LoRaWAN™ Library Plug-in for MPLAB® Code Configurator User’s Guide

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Microchip received ISO/TS-16949:2009 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC® MCUs and dsPIC® DSCs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM CERTIFIED BY DNV

═ ISO/TS 16949 ═

ISBN: 978-1-5224-1294-6

Object of Declaration: LoRaWAN™ Library Plug-in for MPLAB® Code Configurator

EU Declaration of Conformity

This declaration of conformity is issued by the manufacturer.
The development/evaluation tool is designed to be used for research and development in a laboratory environment. This development/evaluation tool is not a Finished Appliance, nor is it intended for incorporation into Finished Appliances that are made commercially available as single functional units to end users under EU EMC Directive 2004/108/EC and as supported by the European Commission's Guide for the EMC Directive 2004/108/EC (8th February 2010).
This development/evaluation tool complies with EU RoHS2 Directive 2011/65/EU.
This development/evaluation tool, when incorporating wireless and radio-telecom functionality, is in compliance with the essential requirement and other relevant provisions of the R&TTE Directive 1999/5/EC and the FCC rules as stated in the declaration of conformity provided in the module datasheet and the module product page available at www.microchip.com.
For information regarding the exclusive, limited warranties applicable to Microchip products, please see Microchip's standard terms and conditions of sale, which are printed on our sales documentation and available at www.microchip.com.
Signed for and on behalf of Microchip Technology Inc. at Chandler, Arizona, USA.

Derek Carlson   11-NOV-16

Derek Carlson   Date
VP Development Tools

NOTES:

Table of Contents

Preface

NOTICE TO CUSTOMERS

All documentation becomes dated, and this manual is no exception. Microchip tools and documentation are constantly evolving to meet customer needs, so some actual dialogs and/or tool descriptions may differ from those in this document. Refer to our website (www.microchip.com) to obtain the latest documentation available.

Documents are identified with a "DS" number. This number is located on the bottom of each page, in front of the page number. The numbering convention for the DS number is "DSXXXXXA", where "XXXXX" is the document number and "A" is the revision level of the document.

For the most up-to-date information on development tools, see the MPLAB® IDE online help. Select the Help menu, and then Topics to open a list of available online help files.

INTRODUCTION

This chapter contains general information that will be useful to know before using the MCC LoRaWAN™ Library Plug-in. Items discussed in this chapter include:

- Document Layout
- Conventions Used in this Guide
- Recommended Reading
- The Microchip Website
- Development Systems Customer Change Notification Service
- Customer Support
- Revision History

DOCUMENT LAYOUT

This document describes how to use the MCC LoRaWAN Library Plug-in as a development tool to emulate and debug firmware on a target board. The document is organized as follows:

- **Chapter 1. "Overview"** – introduces the user to the LoRaWAN Library Plug-in for the MPLAB Code Configuration and presents an overview of the library features.
- **Chapter 2. "LoRaWAN™ Library Plug-in"** – describes the LoRaWAN Library basic and advanced configuration.
- **Chapter 3. "LoRaWAN™ Library Plug-in Running on RN2XX3 Modules"** – gives details on the LoRaWAN configuration on the RN2XX3 modules.
- **Chapter 4. "LoRaWAN™ Stack API"** – describes the APIs available to the user, provided by the LoRaWAN stack.
- **Chapter 5. "Building a LoRaWAN-Based Application"** – describes the LoRaWAN basic operation and offers an example of a custom LoRaWAN-based application.

CONVENTIONS USED IN THIS GUIDE

This manual uses the following documentation conventions:

DOCUMENT CONVENTIONS

Description	Represents	Examples
Arial font:		
Italic characters	Referenced books	*MPLAB IDE User's Guide*
	Emphasized text	...is the *only* compiler...
Initial caps	A window	the Output window
	A dialog	the Settings dialog
	A menu selection	select Enable Programmer
Quotes	A field name in a window or dialog	"Save project before build"
Underlined, italic text with right angle bracket	A menu path	*<u>File>Save</u>*
Bold characters	A dialog button	Click **OK**
	A tab	Click the **Power** tab
N'Rnnnn	A number in verilog format, where N is the total number of digits, R is the radix and n is a digit.	4'b0010, 2'hF1
Text in angle brackets < >	A key on the keyboard	Press <Enter>, <F1>
Courier New font:		
Plain Courier New	Sample source code	`#define START`
	Filenames	`main.c`
	File paths	`c:\mcc18\h`
	Keywords	`_asm, _endasm, static`
	Command-line options	`-Opa+, -Opa-`
	Bit values	`0, 1`
	Constants	`0xFF, 'A'`
Italic Courier New	A variable argument	*`file`*`.o`, where *`file`* can be any valid filename
Square brackets [ ]	Optional arguments	`mcc18 [options]` *`file`* `[options]`
Curly brackets and pipe character: { | }	Choice of mutually exclusive arguments; an OR selection	`errorlevel {0|1}`
Ellipses...	Replaces repeated text	`var_name [, var_name...]`

	Represents code supplied by user	`void main (void)`
		`{ ...`
		`}`

RECOMMENDED READING

This user's guide describes how to use the MCC LoRaWAN Library Plug-in. For the latest information on using other tools, refer to the MPLAB® X IDE home page: www.microchip.com/mplabx/. This resource page contains updated documentation, downloads and links to other MPLAB X compatible tools, plug-ins and much more.

Another recommended reading is the LoRaWAN Specification 1.0. This specification describes the LoRaWAN protocol and it is provided by LoRa® Alliance at http://www.lora-alliance.org.

THE MICROCHIP WEBSITE

Microchip provides online support via our website at www.microchip.com. This website is used as a means to make files and information easily available to customers. Accessible by using your favorite Internet browser, the website contains the following information:

- **Product Support** – Data sheets and errata, application notes, sample programs and labs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip consultant program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

DEVELOPMENT SYSTEMS CUSTOMER CHANGE NOTIFICATION SERVICE

Microchip's customer notification service helps keep customers current on Microchip products. Subscribers will receive e-mail notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest. To register, access the Microchip website at www.microchip.com, click on Customer Change Notification and follow the registration instructions.

The Development Systems product group categories are:

- **Compilers** – The latest information on Microchip C compilers, assemblers, linkers and other language tools. These include all MPLAB C compilers; all MPLAB assemblers (including MPASM™ assembler); all MPLAB linkers (including MPLINK™ object linker); and all MPLAB librarians (including MPLIB™ object librarian).
- **Emulators** – The latest information on Microchip in-circuit emulators.This includes the MPLAB REAL ICE™ and MPLAB ICE 2000 in-circuit emulators.
- **In-Circuit Debuggers** – The latest information on the Microchip in-circuit debuggers. This includes MPLAB ICD 3 in-circuit debuggers and PICkit™ 3 debug express.
- **MPLAB IDE** – The latest information on Microchip MPLAB IDE, the Windows® Integrated Development Environment for development systems tools. This list is focused on the MPLAB IDE, MPLAB IDE Project Manager, MPLAB Editor and MPLAB SIM simulator, as well as general editing and debugging features.
- **Programmers** – The latest information on Microchip programmers. These include production programmers such as MPLAB REAL ICE in-circuit emulator, MPLAB ICD 3 in-circuit debugger and MPLAB PM3 device programmers. Also included are nonproduction development programmers such as PICSTART® Plus and PICkit 2 and 3.

CUSTOMER SUPPORT

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Field Application Engineer (FAE)
- Technical Support

Customers should contact their distributor, representative or field application engineer (FAE) for support. Local sales offices are also available to help customers.

Technical support is available through the website at:

www.microchip.com/support.

REVISION HISTORY

Revision A (November 2016)

Initial release of this document.

Revision B (January 2017)

Minor updates in chapters **2** and **3**.

Added new features to **Section Chapter 4. "LoRaWAN™ Stack API"**.

Added **Section 5.3 "Basic LoRaWAN Class C Stack Operation"**.

Chapter 1. Overview

1.1 INTRODUCTION

The LoRaWAN™ Library Plug-in for the MPLAB® Code Configurator allows for quick and easy C code generation of Microchip's LoRaWAN stack solution for LoRa® Technology end devices.

The LoRaWAN Library Plug-in comes as a Java™ Archive file (`.jar` file extension) and must be added to MPLAB Code Configurator.

LoRaWAN stack currently supports only 8-bit PIC® devices. The minimum requirements for the PIC device to be capable of running LoRaWAN stack are the following:

- 32 kB of Flash memory
- 3 kB of RAM memory
- 1 x SPI
- 6 x GPIOs (three of the GPIOs must be interrupt-capable). If the PIC device has Peripheral Pin Select support, then an additional GPIO is required for the Chip Select of the SPI communication.

This library plug-in uses a Graphical User Interface (GUI) to accomplish the following:

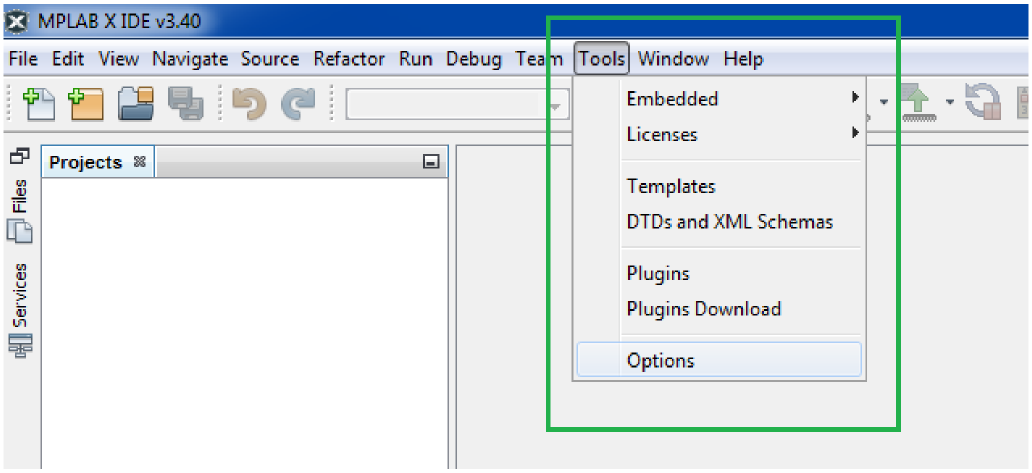
- Select the requirements for the LoRaWAN stack from the device resources (modules present on the device)
- Define the communication channels for the EU 433/868 MHz ISM band
- Enable/Disable communication channels for the NA 915 MHz ISM band
- Adjust various parameters of the LoRaWAN stack in regards to the communication with the server
- Generate the necessary C code to program a PIC® microcontroller

The following chapters of this user's guide cover each component of the LoRaWAN library plug-in and describe the steps to set up a basic LoRaWAN project. For any additional information needed or any queries, contact your local LoRaWAN representative.

1.2 ADD LoRaWAN LIBRARY PLUG-IN TO MCC

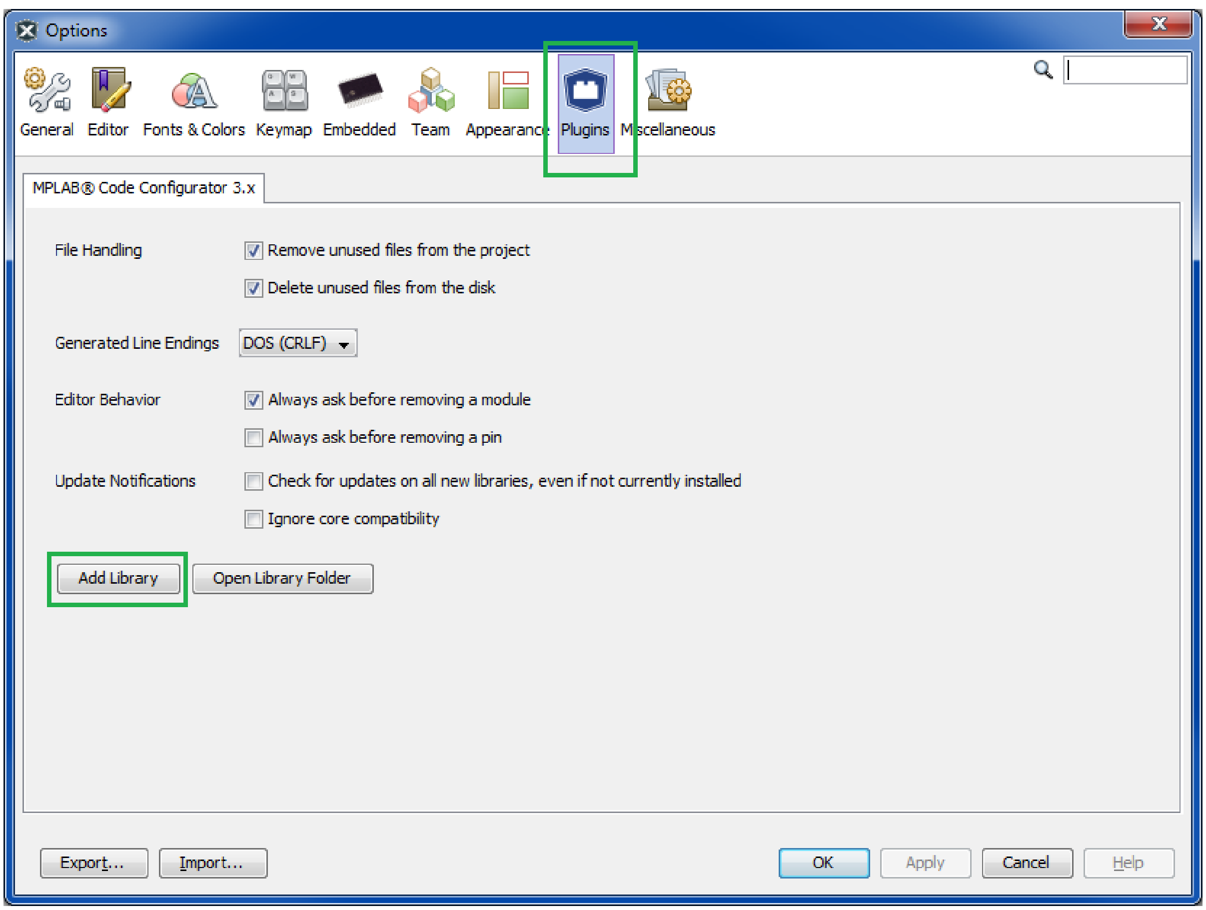
To add the LoRaWAN Library Plug-in, the user must open MPLAB X IDE and click on **Tools -> Options** (see Figure 1-1). The Options window will open.

FIGURE 1-1: ADD LoRaWAN™ LIBRARY PLUG-IN TO MCC



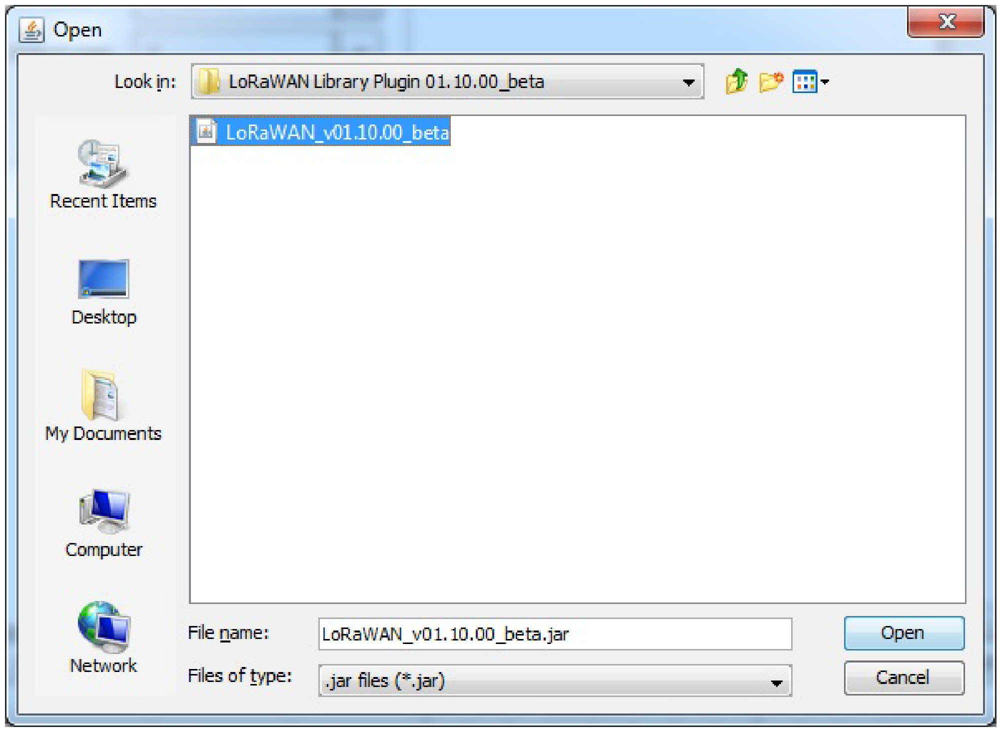
Inside the Options window, click on the **Plug-ins** tab, and then press **Add Library** button (see Figure 1-2).

FIGURE 1-2: ADD LoRaWAN™ LIBRARY PLUG-IN TO MCC – OPTIONS WINDOW



Browse to the location of the LoRaWAN Library Plug-in, select it and click on **Open** to add it to MCC (see Figure 1-3).

FIGURE 1-3: ADD LoRaWAN™ LIBRARY PLUG-IN TO MCC – BROWSE AND OPEN



Note:	If the LoRaWAN™ Library Plug-in has already been added to MCC, a message will pop up, asking the user to confirm the overwrite of the old file.

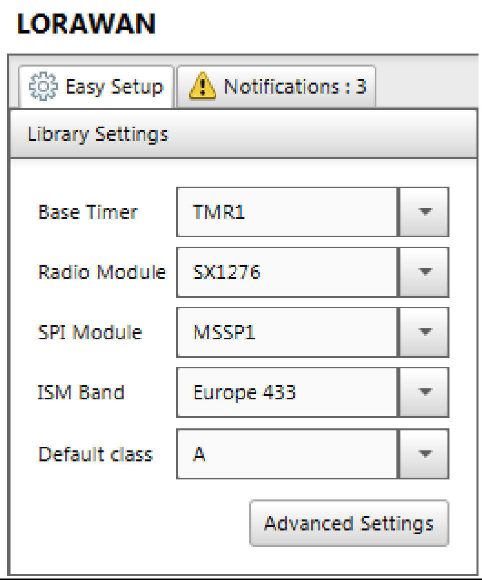
NOTES:

Chapter 2. LoRaWAN™ Library Plug-in

The LoRaWAN Library Plug-in has a wizard menu depicted below in Figure 2-1:

- It provides the minimum setting requirements for basic users
- It offers an Advanced Settings option to access more settings of the LoRaWAN stack

FIGURE 2-1: LoRaWAN™ LIBRARY PLUG-IN



2.1 LoRaWAN BASIC CONFIGURATION

There are five minimum selections that the user must make in order to get a functional LoRaWAN stack. These selections must reflect the LoRaWAN end-device hardware that is used (e.g., the radio transceiver, the MSSP pins of the microcontroller etc.):

1. Base Timer
2. Radio Module
3. SPI Module
4. ISM Band
5. Default Class

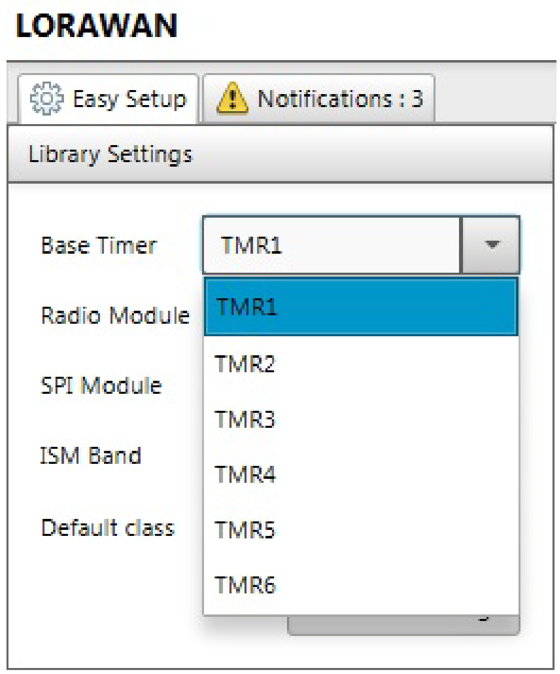
Each of these settings act as a selection from a list of options which is generated according to the microcontroller resources, supported radio modules, or the region of the LoRaWAN stack usage.

2.1.1 LoRaWAN Easy Setup

2.1.1.1 TIMER SELECTION

A timer, which will be used by the LoRaWAN stack, must be selected out of the list of timers available on the microcontroller, as shown in Figure 2-2. The user has to check and ensure a 16-bit timer has been chosen, since this is the specific requirement for the LoRaWAN library.

FIGURE 2-2: LoRaWAN™ BASE TIMER SELECTION



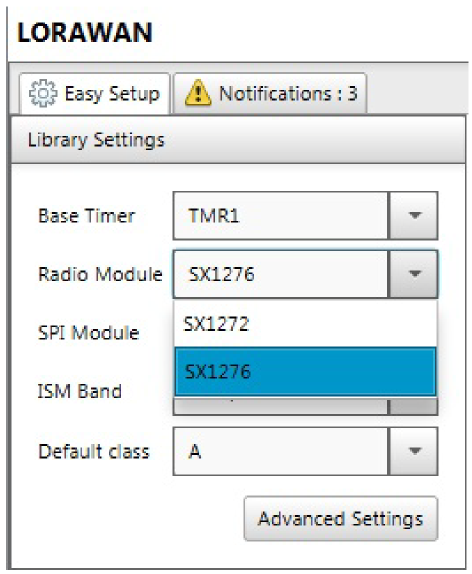
Note:	LoRaWAN stack currently supports only 16-bit timers. The clock source for the timer must be external, running at 32.768 kHz. For more details, see Timer configuration for LoRaWAN stack in **Section 3.2.3.4 "LoRaWAN RN2903 TMR1 Module Configuration"**.

2.1.1.2 RADIO MODULE SELECTION

Currently, the LoRaWAN stack supports two radio transceivers: Semtech SX1272 and Semtech SX1276. The user must select the one present on the end device, as displayed in Figure 2-3.

Note:	SX1272 does not support 433 MHz frequency band.

FIGURE 2-3: LoRaWAN™ RADIO MODULE SELECTION

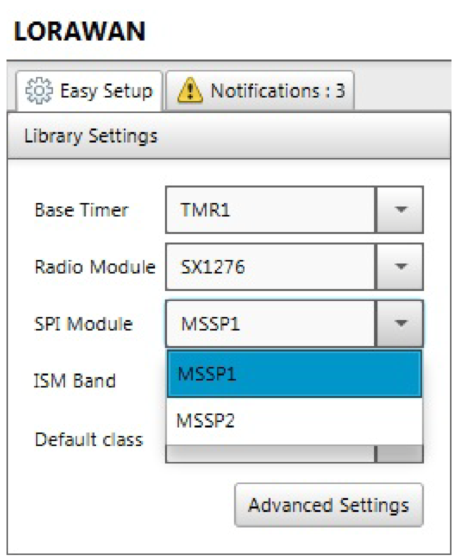


2.1.1.3 SPI MODULE SELECTION

The end device is formed of one PIC device and one radio transceiver. There is a SPI communication needed between these two, where the PIC device is the master and the radio transceiver is the slave. The user must select the MSSP module which will be used for the SPI communication between the PIC microcontroller and the radio (see Figure 2-4).

The selected MSSP module must be configured as Mode 0 (SPI Master).

FIGURE 2-4: LoRaWAN™ SPI MODULE SELECTION



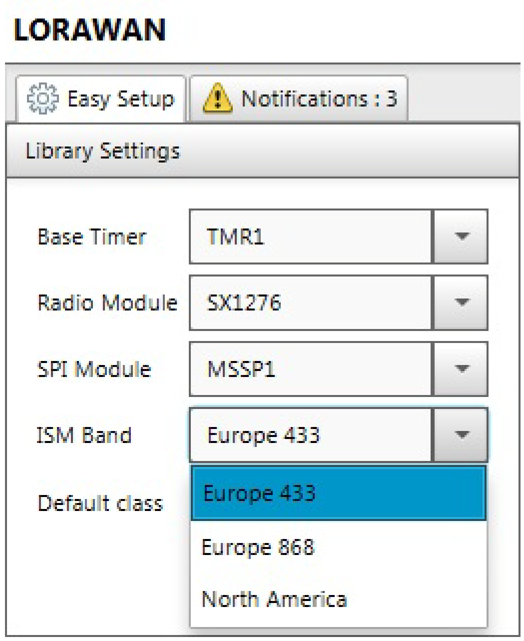
2.1.1.4 ISM BAND SELECTION

There are two regions which are currently supported by the LoRaWAN stack: Europe and North America, with the following specifications:

- Europe is available on 433 and 868 MHz frequency bands
- North America is available on the 915 MHz frequency band

The user must select the desired ISM band (Figure 2-5).

FIGURE 2-5: LoRaWAN™ ISM BAND SELECTION



2.1.1.5 DEFAULT LoRaWAN CLASS SELECTION

Microchip LoRaWAN stack supports both Class A and Class C. The Default class drop-down menu enables user to select which of the two classes shall be active at startup. Changing the class can be made also during runtime by using the class-change API:

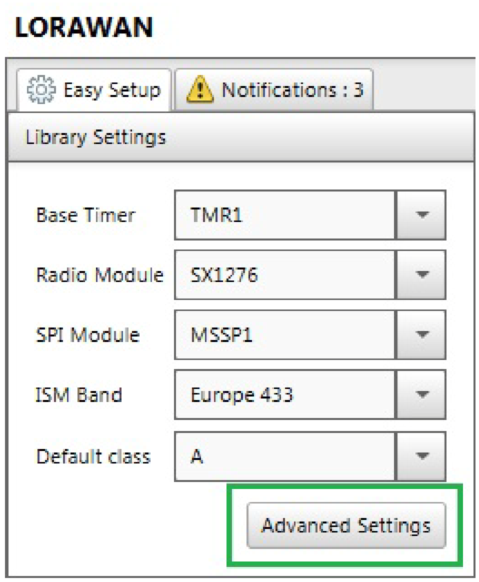
- Class A - device shall be initialized at startup as Class A and shall operate as a Class A device
- Class C - device shall be initialized at startup as Class C and shall operate as a Class C device

Changing the operation Class can be made during runtime, by using the API provided by the LoRaWAN stack (for more details, see **Chapter 4. "LoRaWAN™ Stack API"**).

2.1.2 LoRaWAN Advanced Setup

The advanced settings of the LoRaWAN stack can be accessed by clicking the **Advanced Settings** button, as shown in Figure 2-6. This will take the user to a wizard menu, which allows adding or removing channels for EU 433/868, disable and enable channels for NA 915 and also make various adjustments to the LoRaWAN stack.

FIGURE 2-6: LoRaWAN™ ADVANCED SETTINGS



2.1.2.1 EUROPE 433 MHz ADVANCED SETTINGS

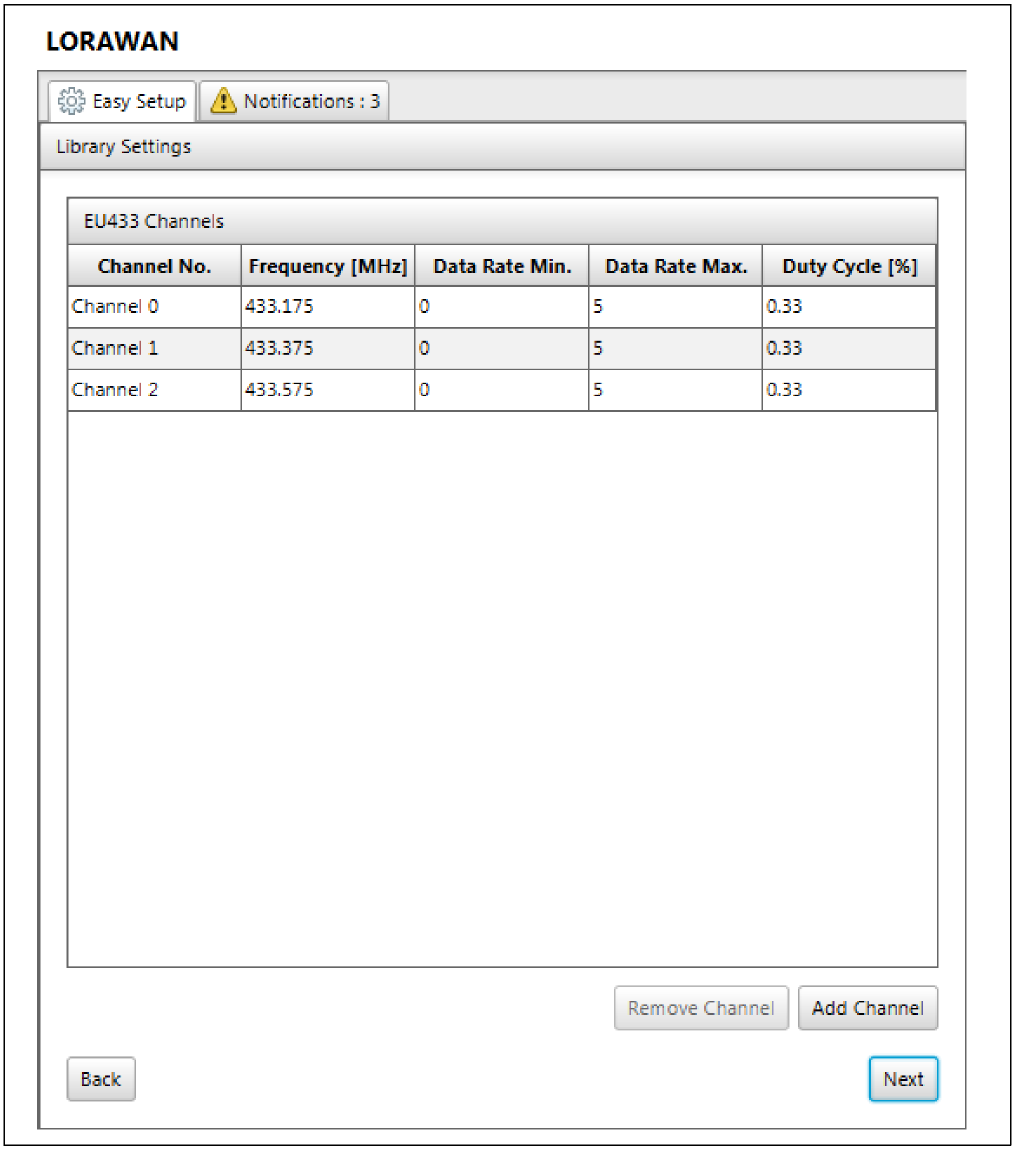
By clicking the **Advanced Settings** button for the EU 433 ISM band, the **Library Settings** tab appears. The first tab of the wizard menu for the Advanced Settings displays the EU 433 Channels table (Figure 2-7). The table contains the following rows:

- Channel Number
- Frequency
- Data Rate Minimum
- Data Rate Maximum
- Duty Cycle

The rows described above represent configurable parameters for each channel. The table already contains the first three default channels. These default channels and their settings are stated in the LoRaWAN Specification V1.0 document. The Frequency for the default channels is fixed and cannot be changed. The minimum and maximum Data Rate Range can be tuned, and the values accepted range between 0 and 7 (Data Rate Maximum must be greater than Data Rate Minimum).

The Duty Cycle needs to be adjusted every time a new channel is created or deleted in order to follow the strict restrictions imposed by the European Telecommunications Standards Institute (ETSI).

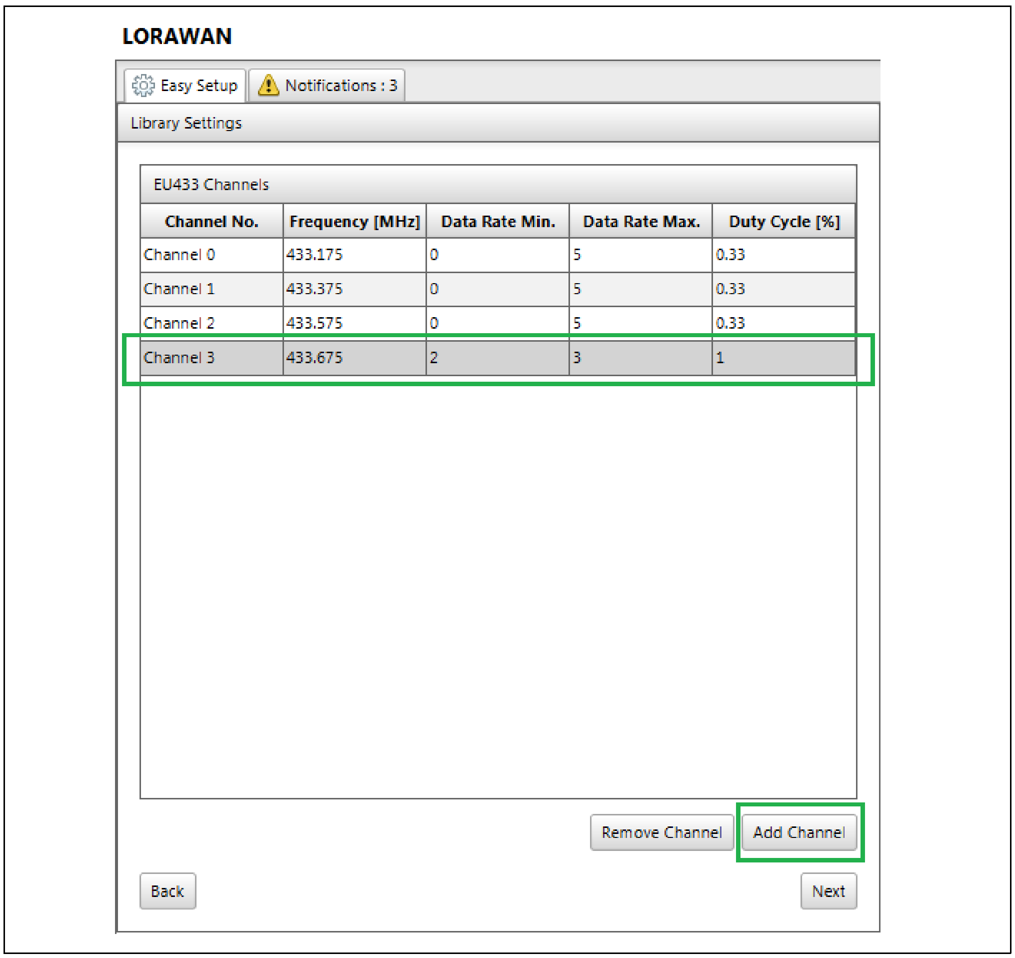
FIGURE 2-7: EU 433 ADVANCED SETTINGS



New channels can be added to the channels table by clicking the **Add Channel** button (Figure 2-8). A maximum number of 13 channels can be added (the maximum number accepted is 16 channels, including the default ones, according to LoRaWAN Specification V1.0). A new entry can be added to the table with the following configurable parameters:

- **Frequency** – by default, the initial value is 433.175 MHz. The accepted range for the EU 433 ISM band is between 433.175 MHz – 434.665 MHz. If the user inserts a value which is not in this range, the cell rejects and signals the invalid value, flashing on a red background.
- **Data Rate** represents the radio parameters used for communication. It is a number from 0 to 7 which corresponds to physical bit rates between 250 bit/s – 50000 bit/s (for more details, refer to the LoRaWAN Specification V1.0 document). Minimum and maximum values can be provided; if the user inserts a value which is not in the range (or if the minimum is greater than the maximum), the cell rejects and signals the invalid value, flashing on a red background.
- **Duty Cycle** represents the time-on-air. E.g.: A 1% duty cycle means that for a one hundred-second period, the end device can transmit for one second and for the rest of 99 seconds it remains on Standby. The accepted range for the duty cycle is between 0 – 100%. If the user inserts a value which is not in this range, the cell rejects and signals the invalid value, flashing on a red background.

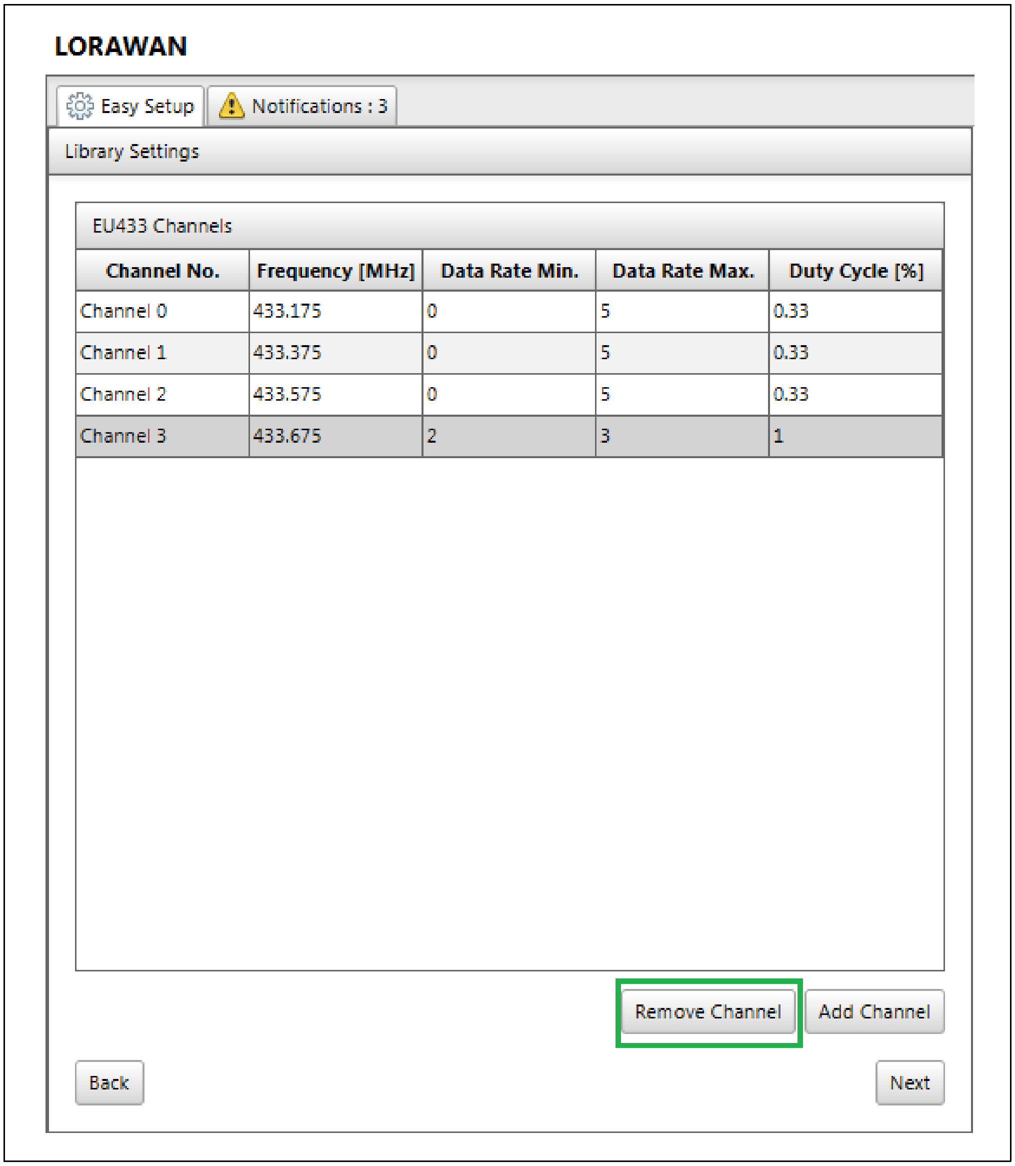
FIGURE 2-8: ADD NEW CHANNEL FOR EUROPE 433 MHz



For more details on the channels and data rates, see LoRaWAN Specification V1.0 document.

Any channel which was previously inserted can be removed from the table by selecting it and clicking the **Remove Channel** button (Figure 2-9). The first three default channels cannot be removed.

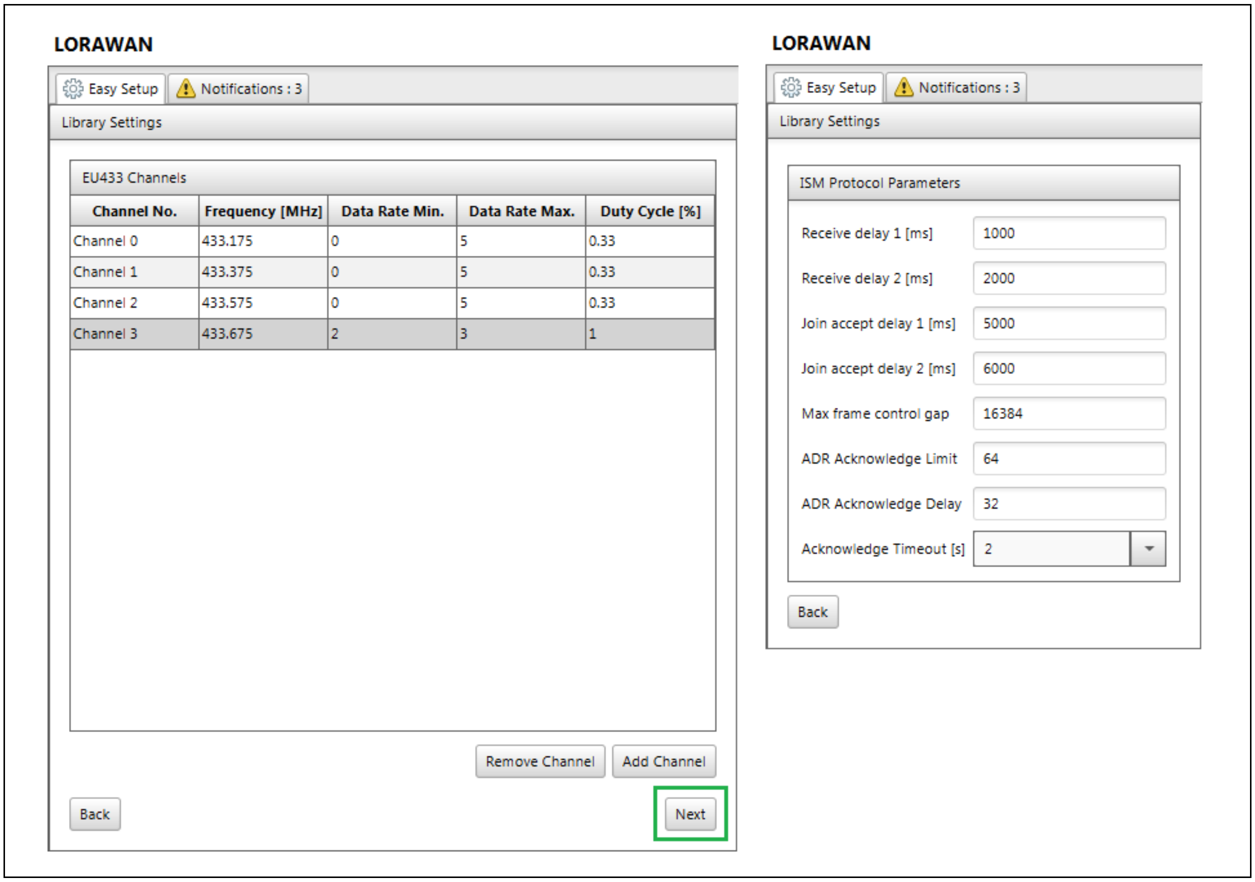
FIGURE 2-9: REMOVE ADDED CHANNEL FOR EUROPE 433 MHz



For more details on the channels and data rates, see LoRaWAN Specification V1.0 document.

The second tab of the EU433 Advanced Settings can be accessed by clicking the **Next** button. This tab allows the user to configure the LoRaWAN stack parameters such as receive delays, join accept delays or acknowledge time out. The parameters have already been loaded with the default values.

FIGURE 2-10: ISM PROTOCOL PARAMETERS



Note: Any change of these values should only be made by users with advanced knowledge of LoRa® Technology. Setting incorrect values to these parameters may result into a non-functional LoRaWAN™ stack.

As soon as the ISM Protocol Parameters setting is complete, the **Back** button can be pressed, taking the user to the previous tab – **EU433 Channels**. By pressing the **Back** button again, the user returns to the Easy Setup view of the LoRaWAN Library.

2.1.2.2 EUROPE 868 MHz ADVANCED SETTINGS

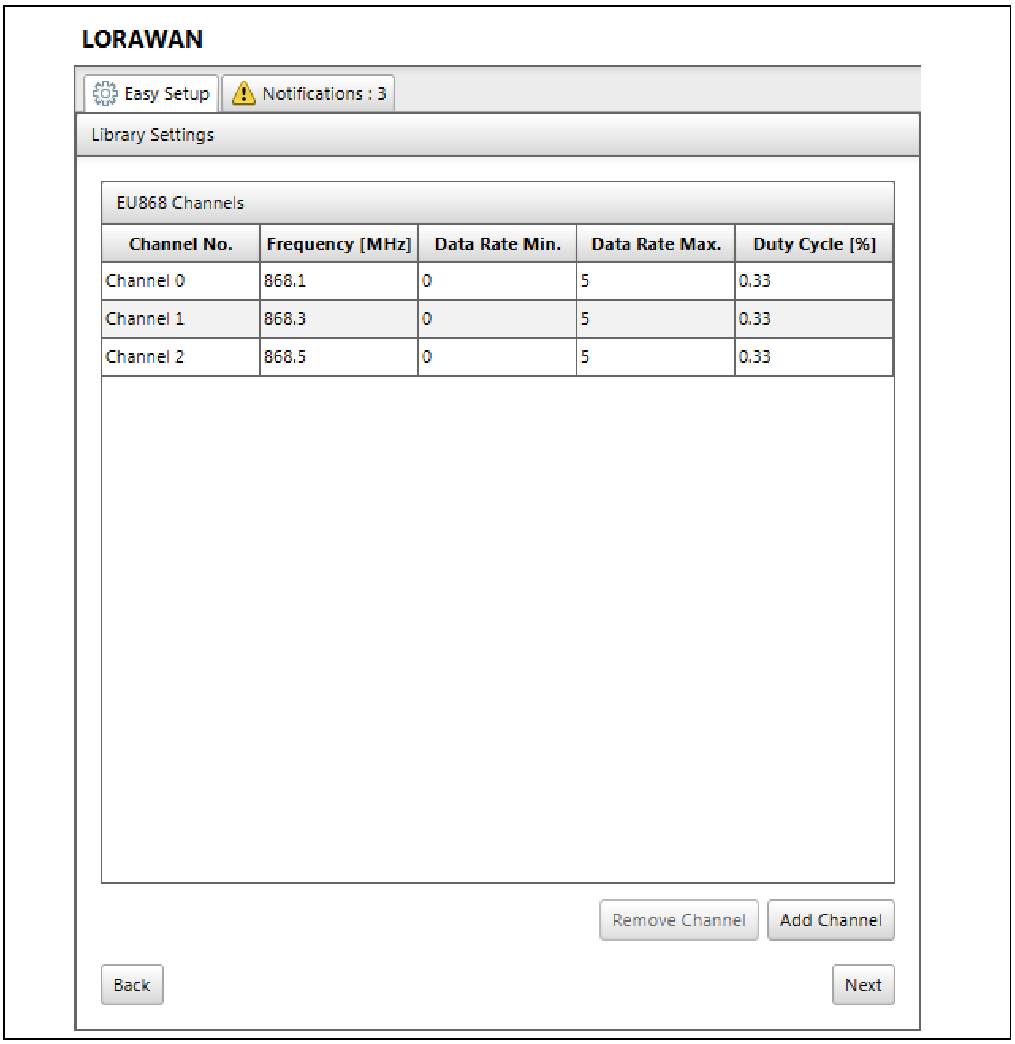
By clicking the **Advanced Settings** button for EU868 ISM band, the **Library Settings** tab appears. The first tab of the wizard menu for the Advanced Settings shows the EU868 channels table, which can be seen in Figure 2-11. The table is formed of the following rows:

- Channel Number
- Frequency
- Data Rate Minimum
- Data Rate Maximum
- Duty Cycle

There are configurable parameters for each channel. The table already contains the first three default channels. These default channels and their settings are stated in the LoRaWAN Specification V1.0 document. The Frequency for the default channels is fixed and cannot be changed. The Data Rate Range Minimum and Maximum can be tuned and the values accepted range between 0 and 7 (Data Rate Maximum must be greater than Data Rate Minimum).

The Duty Cycle needs to be adjusted every time a new channel is created or deleted in order to follow the strict restrictions imposed by the European Telecommunications Standards Institute (ETSI).

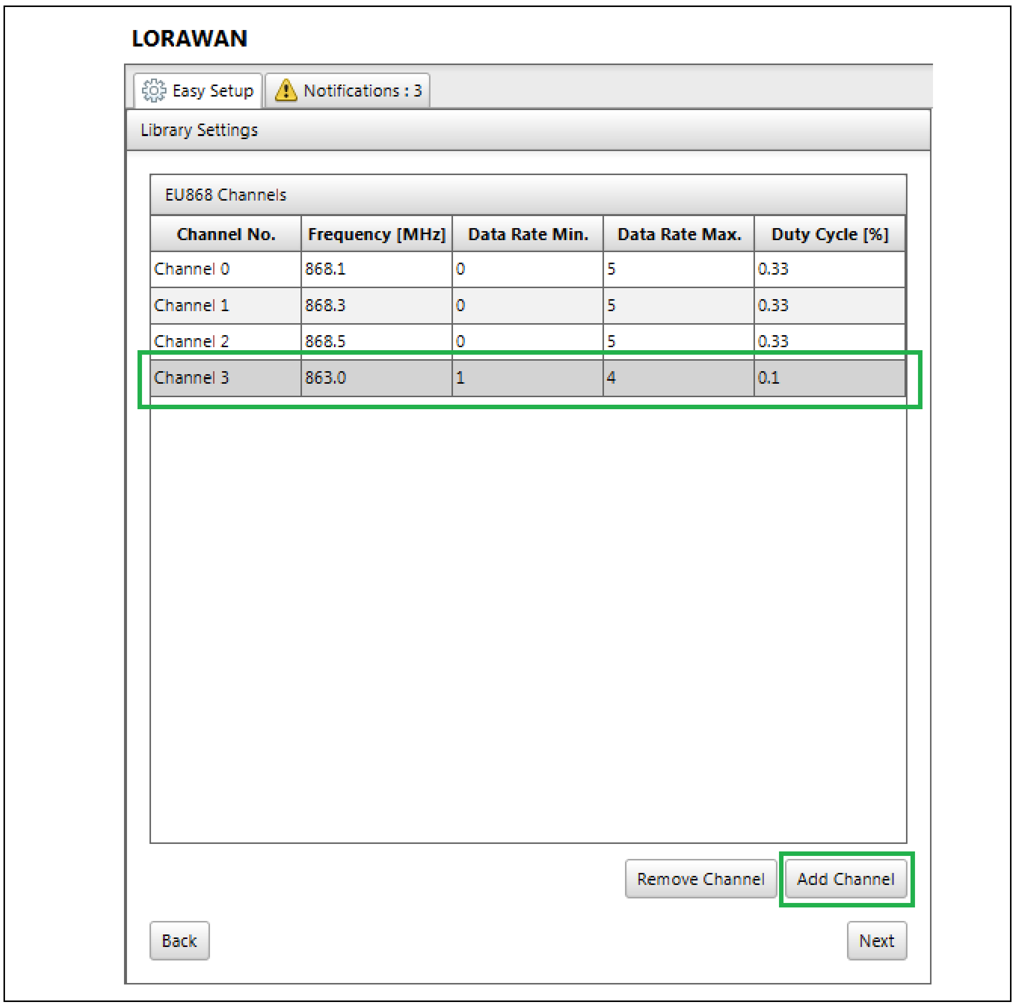
FIGURE 2-11: EUROPE 868 MHz ADVANCED SETTINGS



New channels can be added to the channels table by clicking the **Add Channel** button (Figure 2-12). A maximum number of 13 channels can be added (the maximum number accepted is 16 channels, including the default ones, according to LoRaWAN Specification V1.0). A new entry can be added to the table with the following configurable parameters:

- **Frequency** – by default, the initial value is 863.0 MHz. The accepted range for the EU 868 ISM band is between 863.0 MHz – 870.0 MHz. If the user inserts a value which is not in this range, the cell rejects and signals the invalid value, flashing on a red background.
- **Data Rate** represents the radio parameters used for communication. It is a number from 0 to 7 which corresponds to physical bit rates between 250 bit/s – 50000 bit/s (for more details, refer to the LoRaWAN Specification V1.0 document). Minimum and maximum values can be provided; if the user inserts a value which is not in the range (or if the minimum is greater than the maximum), the cell rejects and signals the invalid value, flashing on a red background.
- **Duty Cycle** represents the time on air. E.g.: A 1% duty cycle means that for a one hundred-second period, the end device can transmit for one second and for the rest of 99 seconds it remains on Standby. The accepted range for the duty cycle is between 0 – 100%. If the user inserts a value which is not in this range, the cell rejects and signals the invalid value, flashing on a red background.

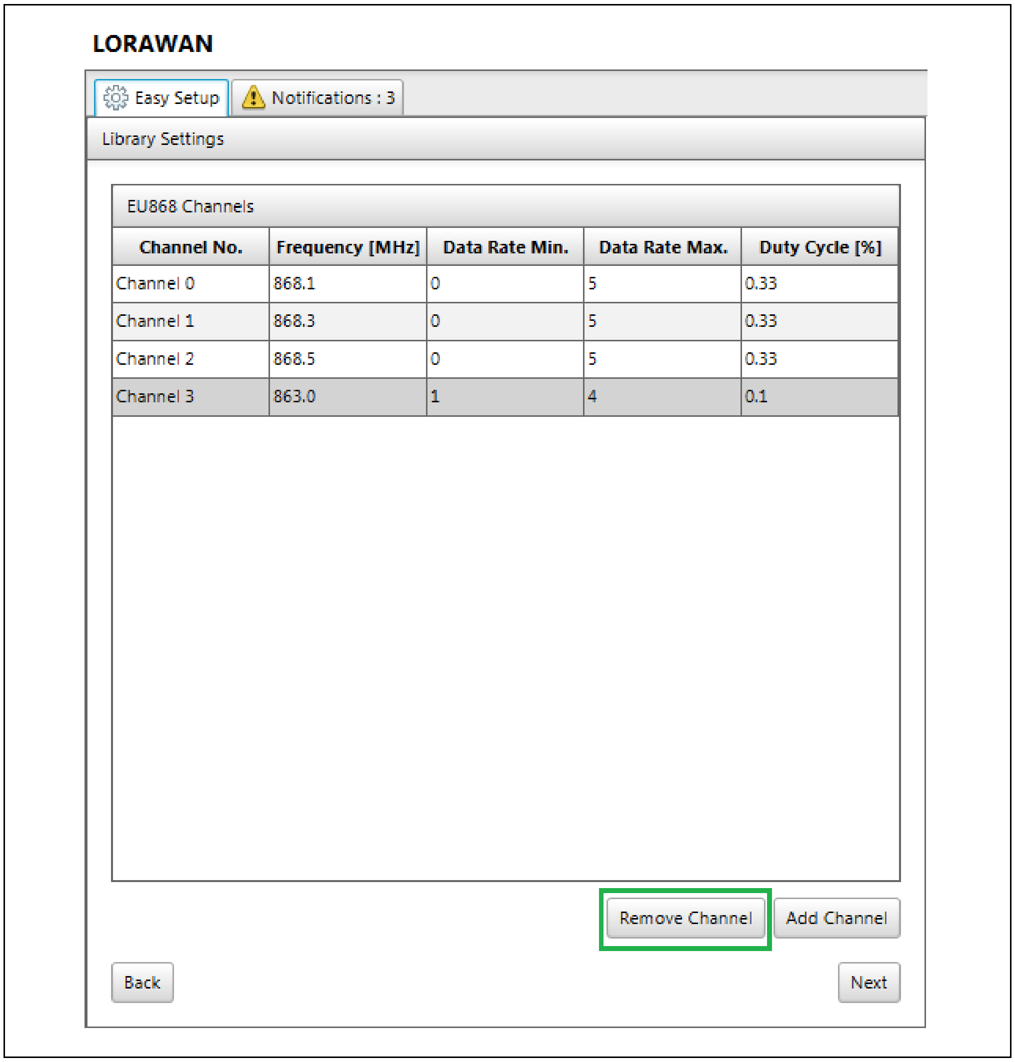
FIGURE 2-12: ADD NEW CHANNEL FOR EUROPE 868 MHz



For more details on the channels and data rates, see LoRaWAN Specification V1.0 document.

Any channel which was previously inserted can be removed from the table by selecting it and clicking the **Remove Channel** button, as displayed in Figure 2-13. The first three default channels cannot be removed.

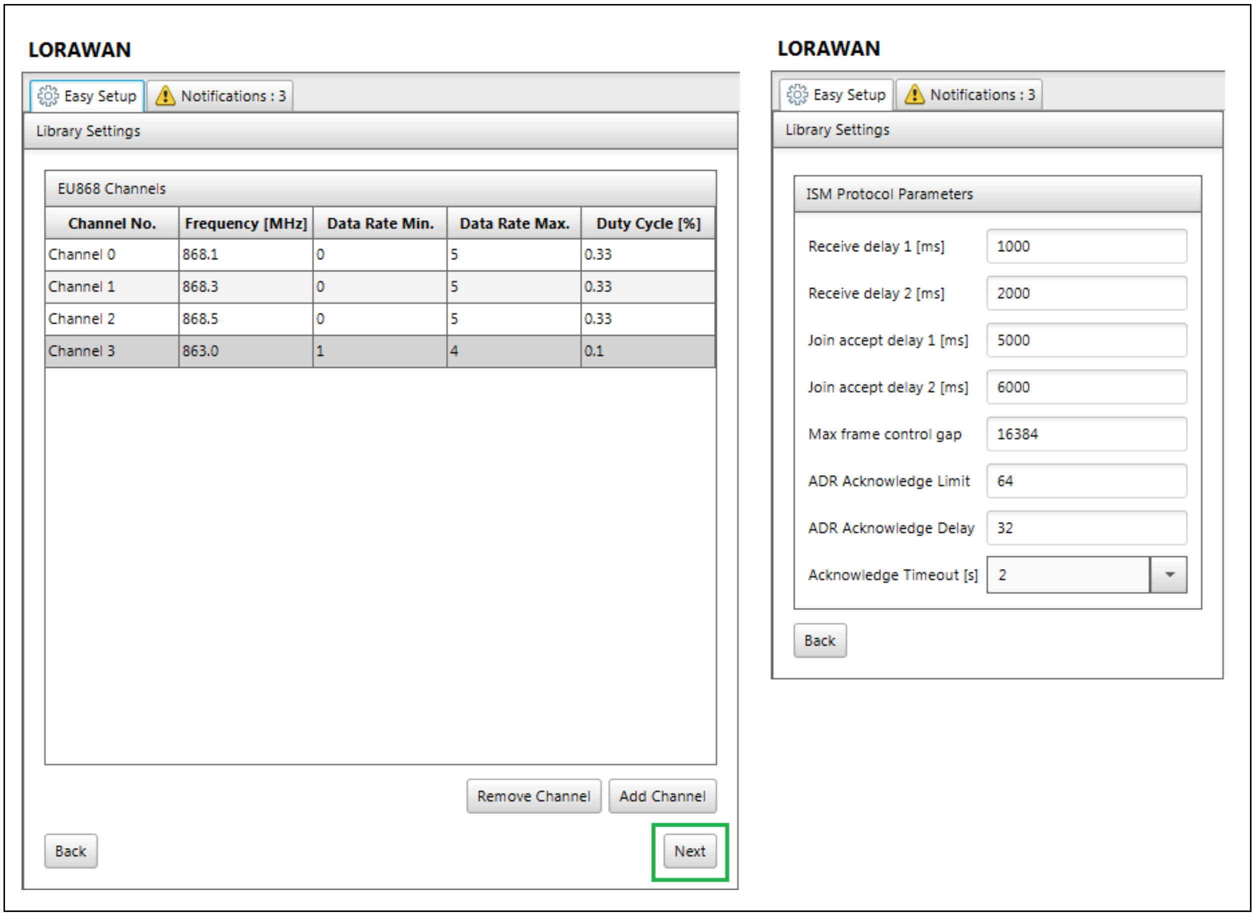
FIGURE 2-13: REMOVE ADDED CHANNEL FOR EUROPE 868 MHz



For more details on the channels and data rates, see LoRaWAN Specification V1.0 document.

The second tab of the EU868 Advanced Settings can be accessed by clicking the **Next** button. This tab allows the user to configure the LoRaWAN stack parameters such as receive delays, join accept delays or acknowledge time out. The parameters have already been loaded with the default values.

FIGURE 2-14: ISM PROTOCOL PARAMETERS



> **Note:** Any change of these values should only be made by users with advanced knowledge of LoRa® Technology. Setting incorrect values to these parameters may result into a non-functional LoRaWAN™ stack.

As soon as the ISM Protocol Parameters setting is complete, the **Back** button can be pressed, taking the user to the previous tab – **EU868 Channels**. By pressing the **Back** button again, the user returns to the Easy Setup view of the LoRaWAN Library.

2.1.2.3 NORTH AMERICA 915 MHz ADVANCED SETTINGS

For North America 915 MHz, there are 72 channels, split into two categories: 0 to 63 and 64 to 71. The frequency for all the channels is stated into the LoRaWAN Specification V1.0 document. By default, the 72 channels are already enabled.

The channels can be disabled by clearing the checkbox on the Active column for each channel and re-enabled by clicking the checkbox for each channel of this **Active** column.

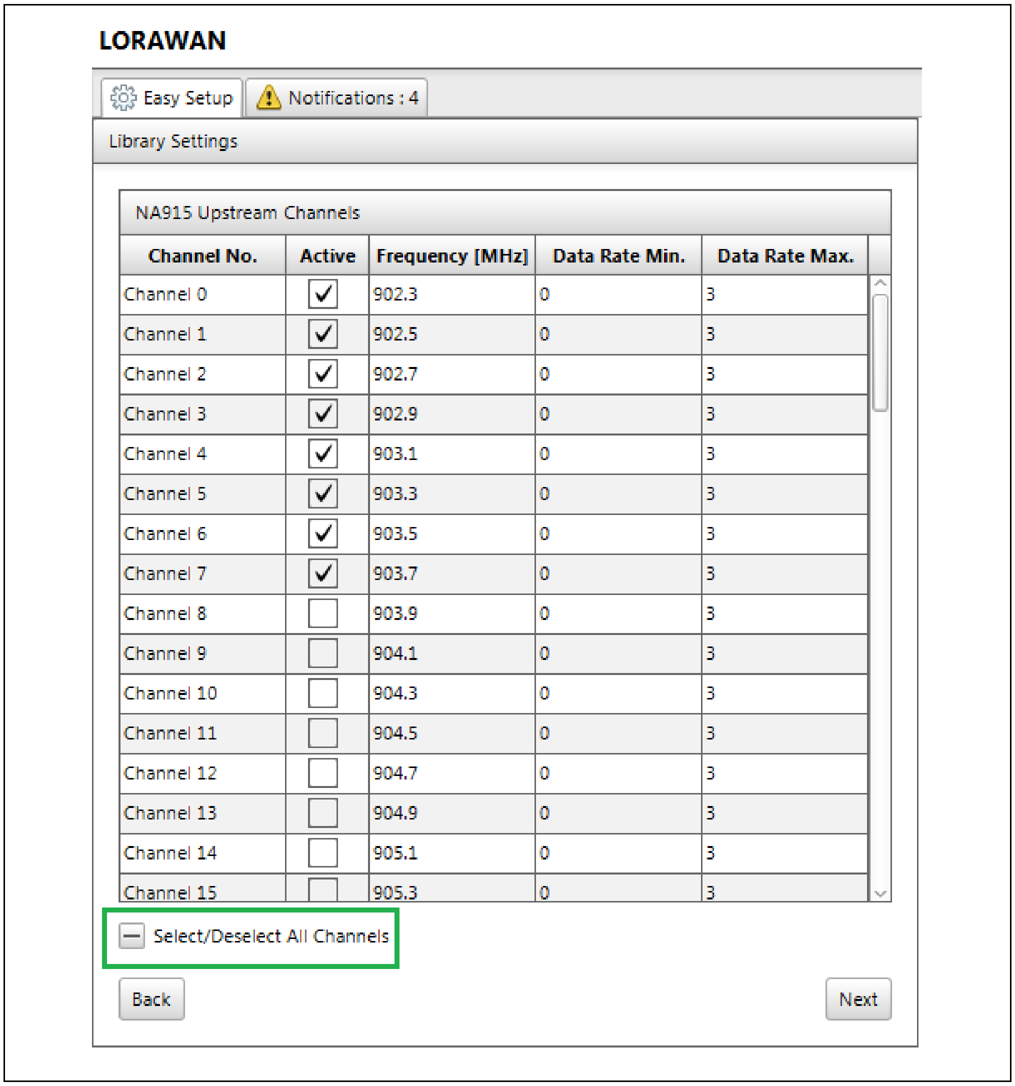
For the Channels 0 – 63, Data Rate Min. and Data Rate Max. values can be changed. The range for the Data Rate (Minimum and Maximum) is 0-3, corresponding to 980 bit/s to 5470 bit/s.

There is a Select/Deselect All Channels button which can be used in order to enable or disable all channels for North America (See Figure 2-15).

For the Channels 64 – 71, the Data Rate is fixed to 4 (corresponding to 12500 bit/s) thus, Data Rate Min. is equal to Data Rate Max. and these cells are not editable.

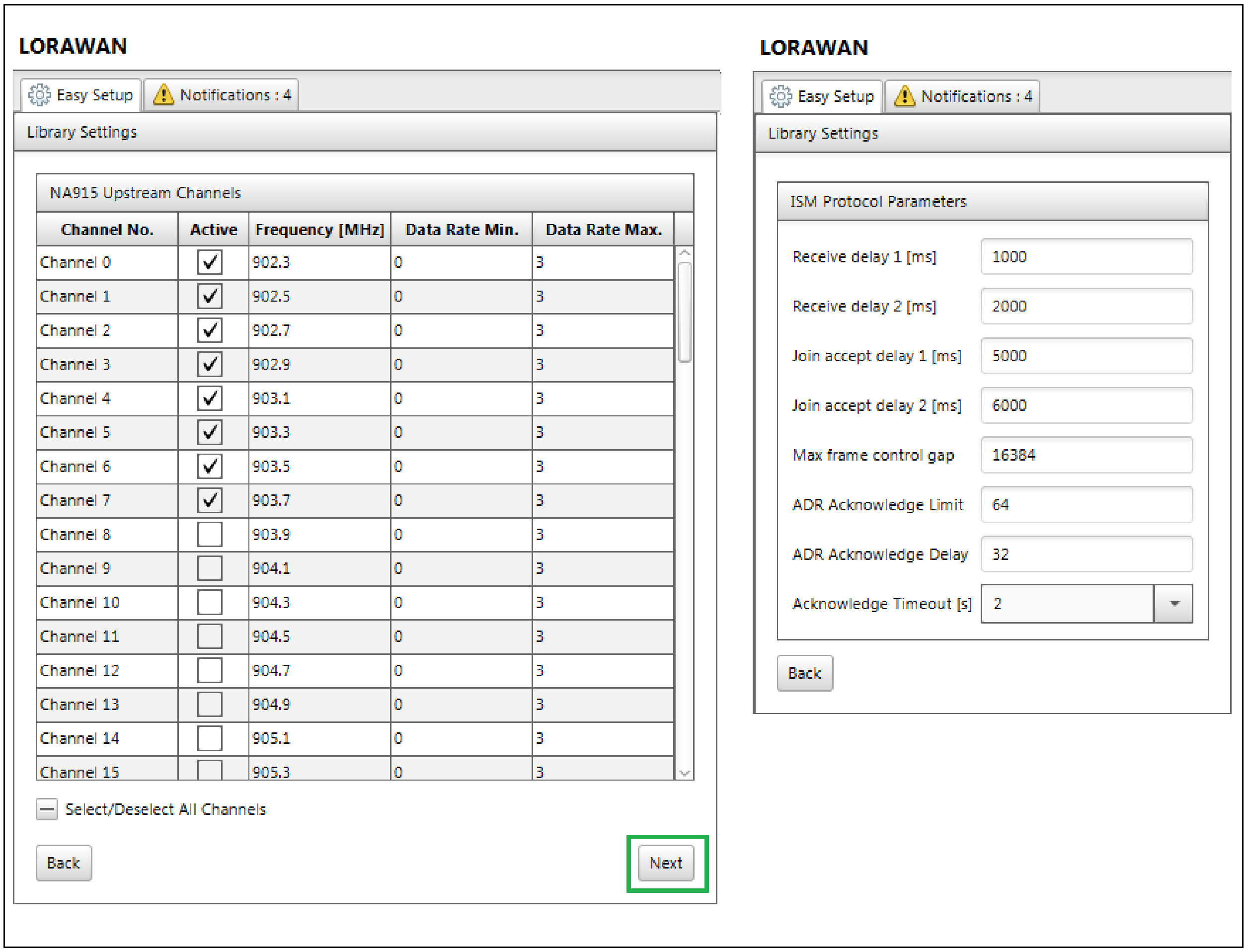
For more details on the channels and data rates, see LoRaWAN Specification V1.0 document.

FIGURE 2-15: ENABLE/DISABLE CHANNELS FOR NORTH AMERICA 915 MHz



The second tab of North America 915 MHz Advanced Settings can be accessed by clicking the **Next** button. This tab allows the user to configure LoRaWAN stack parameters such as receive delays, join accept delays or acknowledge time out. The parameters have already been already loaded with the default values.

FIGURE 2-16: ISM PROTOCOL PARAMETERS



> **Note:** Any change of these values should only be made by users with advanced knowledge of LoRa® Technology. Setting incorrect values to these parameters may result into a non-functional LoRaWAN™ stack.

As soon as the ISM Protocol Parameters setting is complete, the **Back** button can be pressed, taking the user to the previous tab – **NA915 Channels**. By pressing the **Back** button again, the user returns to the Easy Setup view of the LoRaWAN Library.

NOTES:

Chapter 3. LoRaWAN™ Library Plug-in Running on RN2XX3 Modules

3.1 OVERVIEW

The LoRaWAN stack is already present on the off-the-shelf Microchip RN2483 and RN2903 modules. This chapter describes the project configuration created in order to generate the LoRaWAN stack and operate on the Microchip RN2XX3 modules.

The same configuration can be made on both RN2483 and RN2903; however, the ISM Band chosen during LoRaWAN Library Basic Configuration is different.

Both cores of RN2483 and RN2903 modules contain the PIC18LF46K22 device and the Semtech SX1276 radio transceiver. Thus, the project configuration is made for this specific system (PIC18LF46K22 and SX1276).

RN2903 (for North America) will be used in the following example.

Note:	The RN2XX3 modules are sold as standard, as regulatory certified and LoRa Alliance certified modules. These certifications make reference to the standard production firmware and so overwriting with custom MCC firmware will void those certifications.

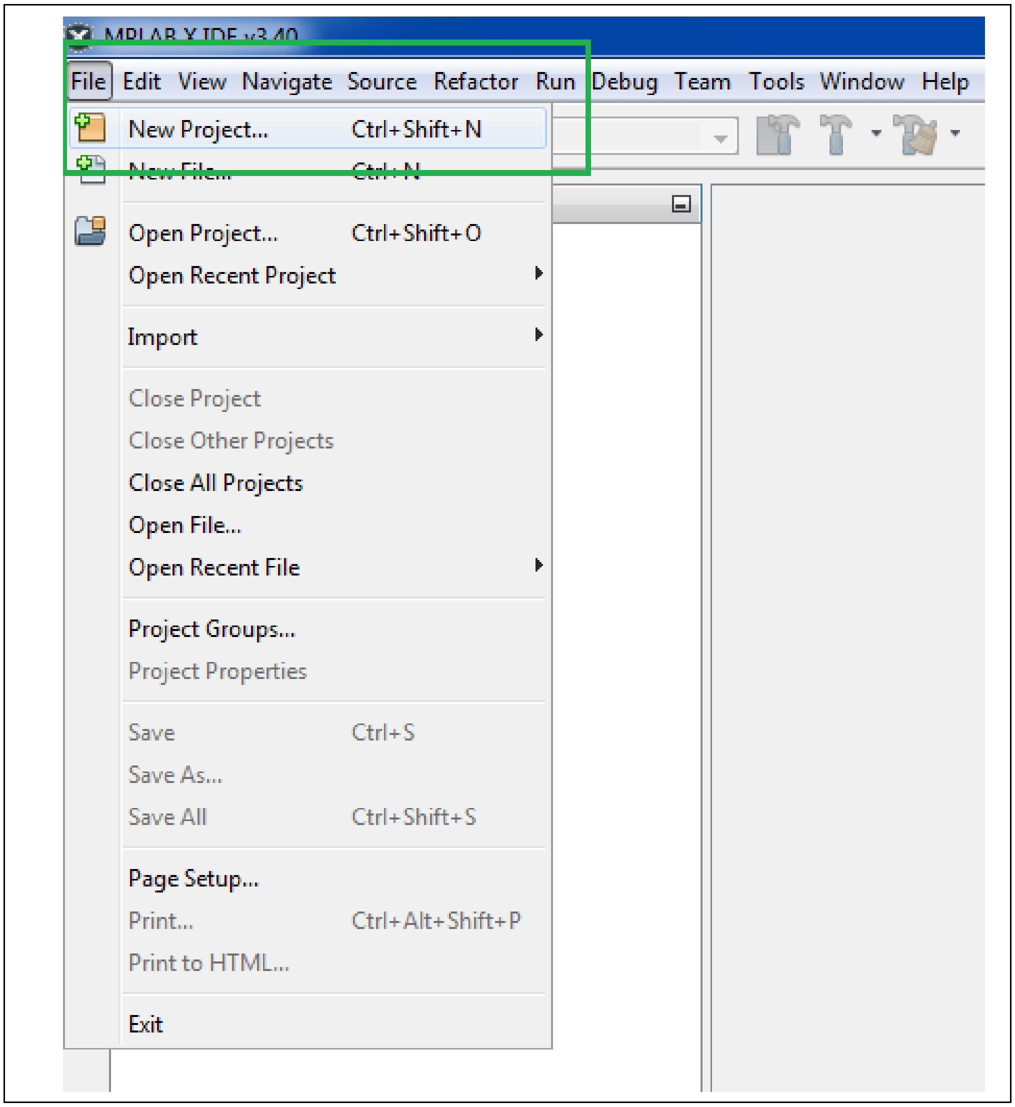
3.2 LoRaWAN CONFIGURATION ON RN2XX3 MODULES

3.2.1 LoRaWAN PROJECT CREATION

A new MPLAB® X project has to be created and set up. The steps are described below:

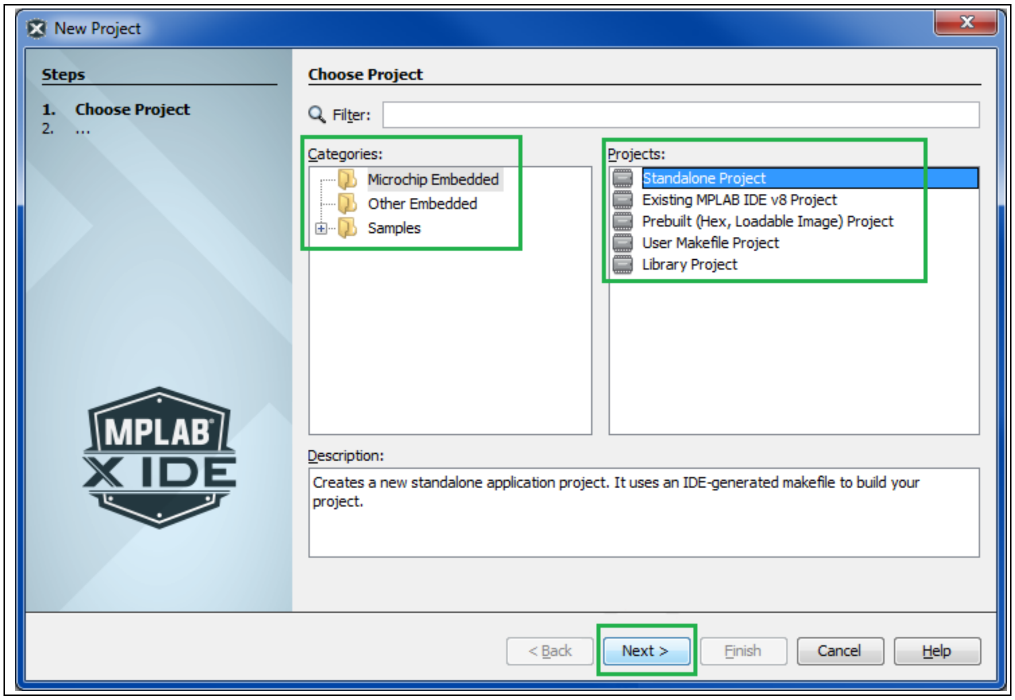
1. In MPLAB® X, click on *File > New Project* (see Figure 3-1).

FIGURE 3-1: MPLAB® X NEW PROJECT SELECTION



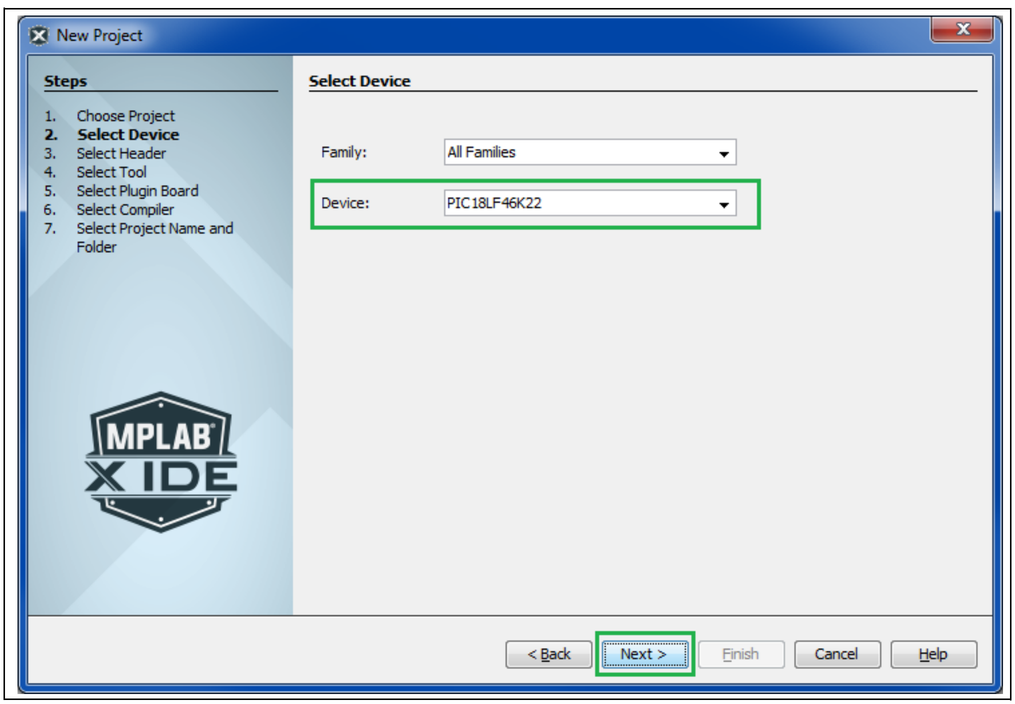
2. Select Microchip Embedded under the Categories window and Standalone Project under the Projects window and click **Next** (see Figure 3-2).

FIGURE 3-2: MPLAB® X NEW PROJECT STANDALONE



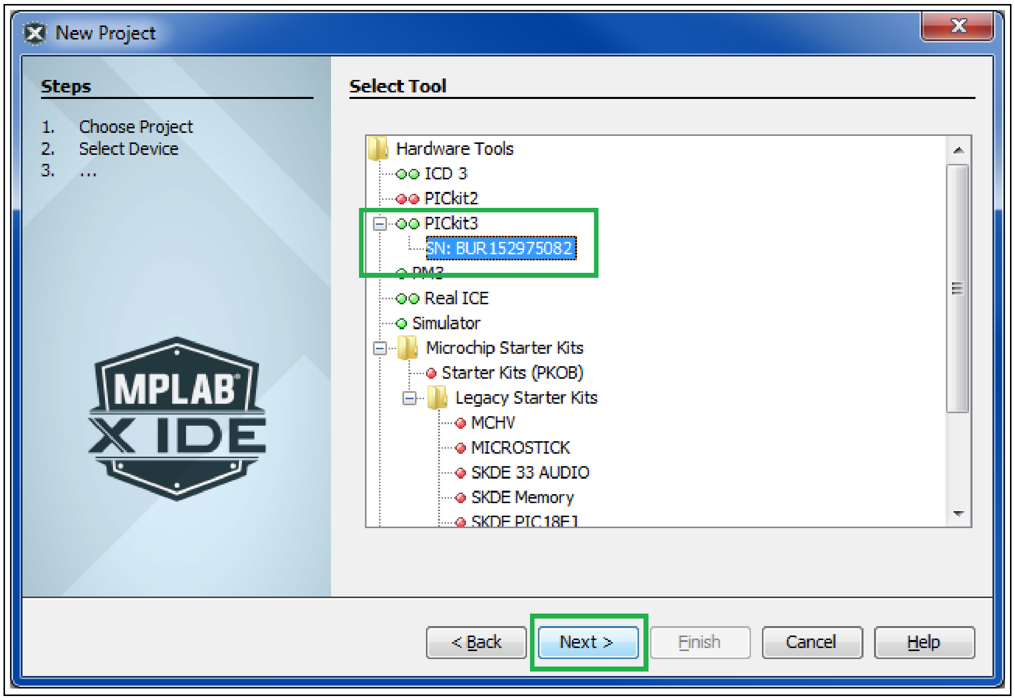
3. Select PIC18LF46K22 from the Device drop-down list and press the **Next** button, as shown in Figure 3-3.

FIGURE 3-3: MPLAB® X NEW PROJECT DEVICE SELECTION



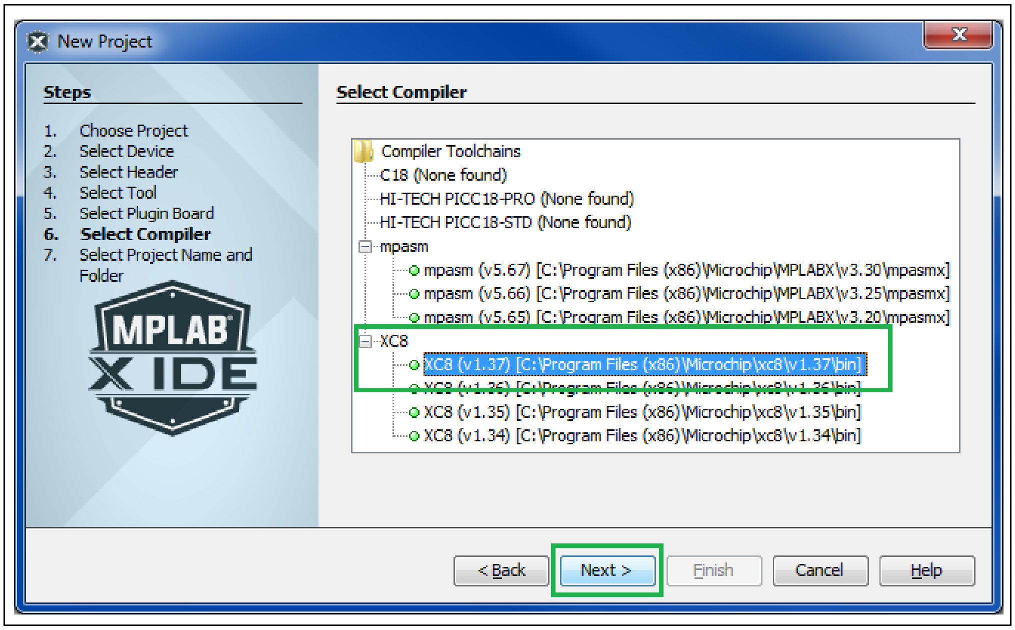
4. Select the programmer used from the Hardware Tools tree and click **Next**, as displayed in Figure 3-4. If no programmer or debug tool is physically connected to the PC, the Simulator can be used instead.

FIGURE 3-4: MPLAB® X NEW PROJECT PROGRAMMER SELECTION



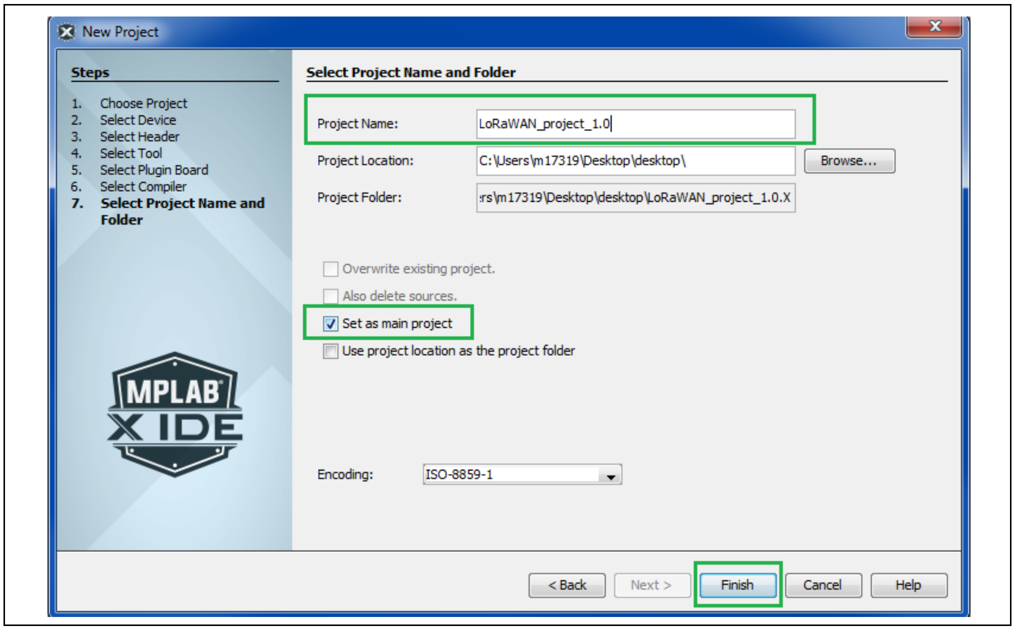
5. Select the compiler to be used. The LoRaWAN Library requires XC8 v1.37 (or later). Select XC8 (v1.37) and click **Next**, as shown in Figure 3-5.

FIGURE 3-5: MPLAB® X NEW PROJECT COMPILER SELECTION



6. Set a name for the project and make sure 'Set as main project' check box is selected. Click the **Finish** button; this will end the new project creation process.

FIGURE 3-6: MPLAB® X NEW PROJECT NAME AND FOLDER

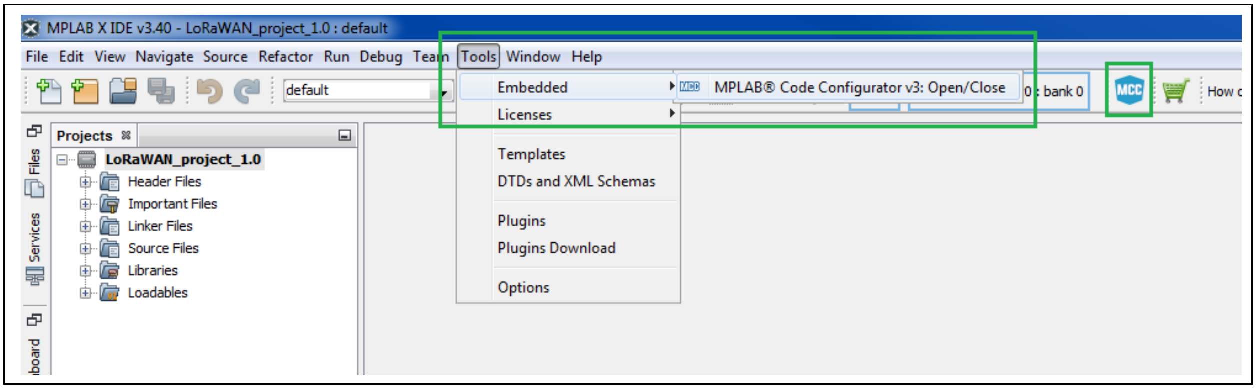


3.2.2 LoRaWAN BASIC CONFIGURATION

The steps for creating the LoRaWAN basic configuration are described below.

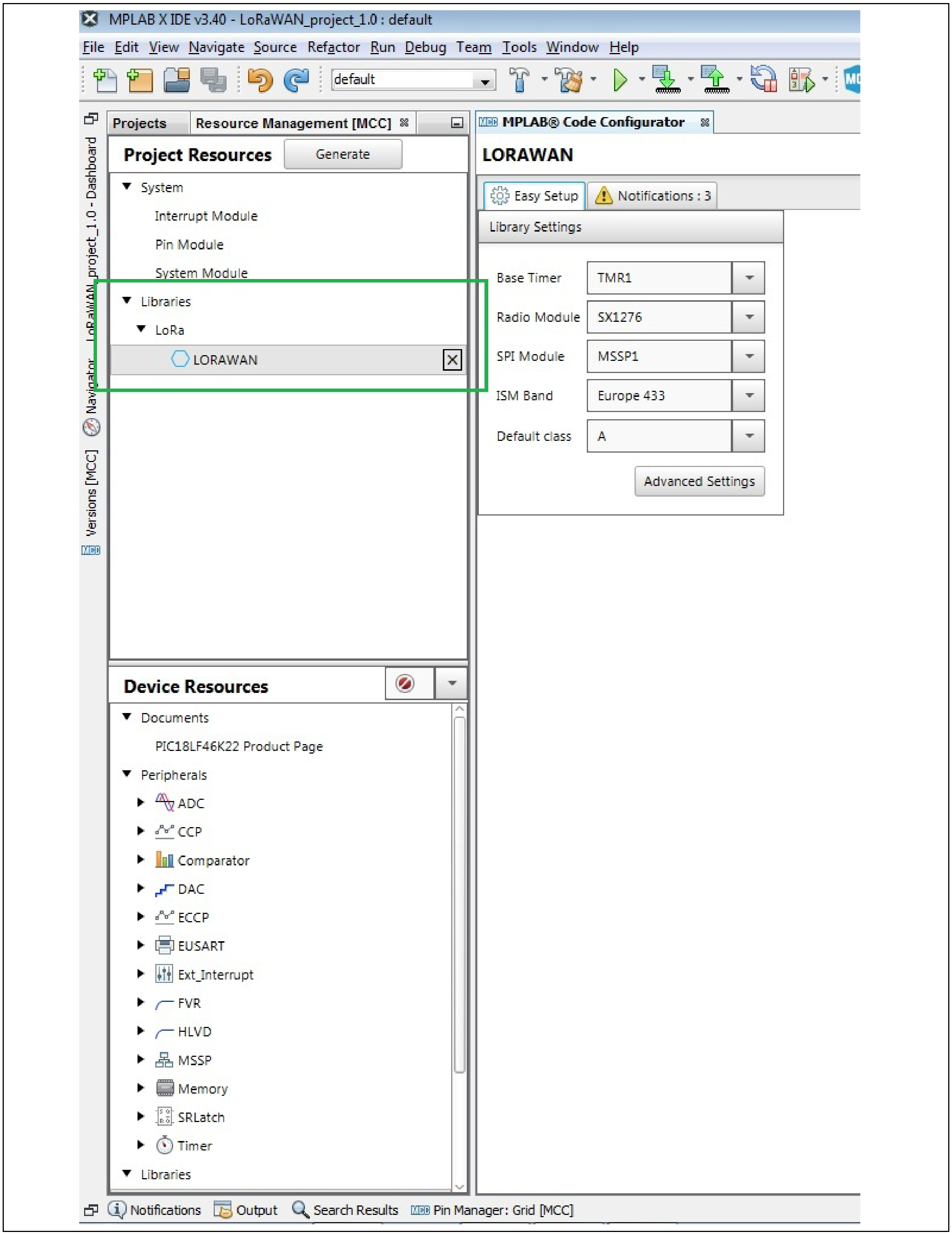
1. Open the MPLAB® Code Configurator (MCC) in order to create the configuration of the project. In MPLAB X, click on _Tools > Embedded > MPLAB®Code Configurator v3 Open/Close_ or click on the MCC icon located on the MPLAB X toolbar (see Figure 3-7).

FIGURE 3-7: MCC OPEN



2. Add the LoRaWAN Library. From the Device Resources window, expand *Libraries > LoRa* and double click **LoRaWAN**. The library will be added to the Project Resources (see Figure 3-8).

FIGURE 3-8: ADD LoRaWAN™ LIBRARY TO PROJECT RESOURCES

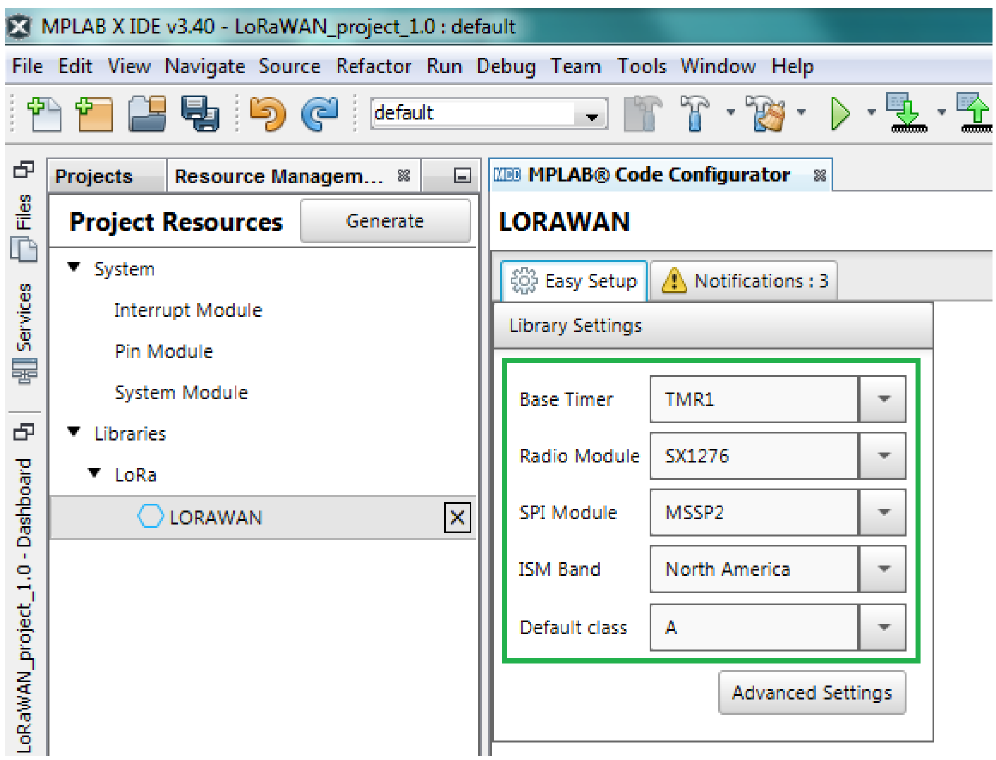


3. Select the resources needed by the LoRaWAN stack (Base Timer, Radio module, SPI and ISM band). Make the following settings for the LoRaWAN Library, as shown in Figure 3-9:

- Base Timer: TMR1
- Radio Module: SX1276
- SPI Module: MSSP2
- ISM Band: North America
- Default class: A

Note: The ISM Band should be selected according to the area where the end device using the LoRaWAN stack will be used. If the user wants to run the LoRaWAN-based application inside Europe, then the ISM band chosen should be Europe 433 MHz (EU433 on the list) or Europe 868 MHz (EU868 on the list).

FIGURE 3-9: LoRaWAN™ LIBRARY NEEDED RESOURCES

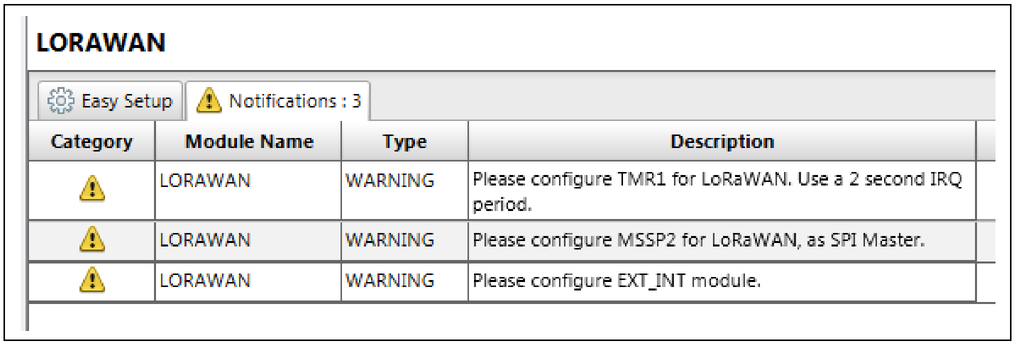


As soon as the above settings are made, there will be three notifications displayed on the **Notifications** tab. Click the tab in order to view these notifications, as they appear in Figure 3-10. They will send warnings to the user about:

- Configuring TMR1 for LoRaWAN to use a 2-second interrupt request period
- Configuring MSSP2 for LoRaWAN in SPI Master mode
- Configuring the EXT_INT (External Interrupt) module

All these settings will be further explained in **Section 3.2.3 “LoRaWAN RN2XX3 Configuration”**.

FIGURE 3-10: LoRaWAN™ LIBRARY BASIC CONFIGURATION WARNINGS



3.2.3 LoRaWAN RN2XX3 Configuration

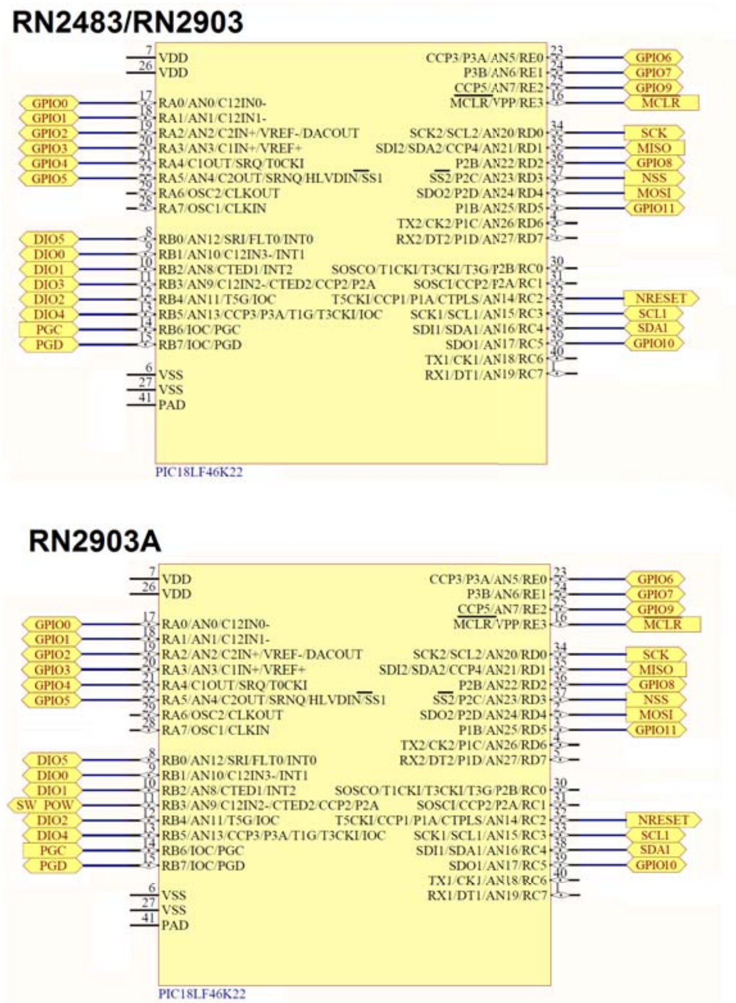
3.2.3.1 OVERVIEW

The configuration described in this chapter is strictly related to the hardware and the physical hardware links present on the RN2XX3 modules (both RN2483 and RN2903).

The hardware links present on the RN2903 are shown below (Figure 3-11). The points of interest are:

- SPI pins (MSSP pins)
- NSS (Chip Select)
- NRESET pin
- DIOx pins
- SW_POW pin (only for RN2903A)

FIGURE 3-11: RN2903/RN2483 HARDWARE LINKS



> **Note:** SCL1 and SDA1 are connected to a 2K I²C Serial EEPROM with EUI-64™ Node Identity (24AA02E64T-I/OT) using MSSP1 configured in I²C mode.
>
> For more details on GIPOx pins, please refer to RN2XX3 data sheet.

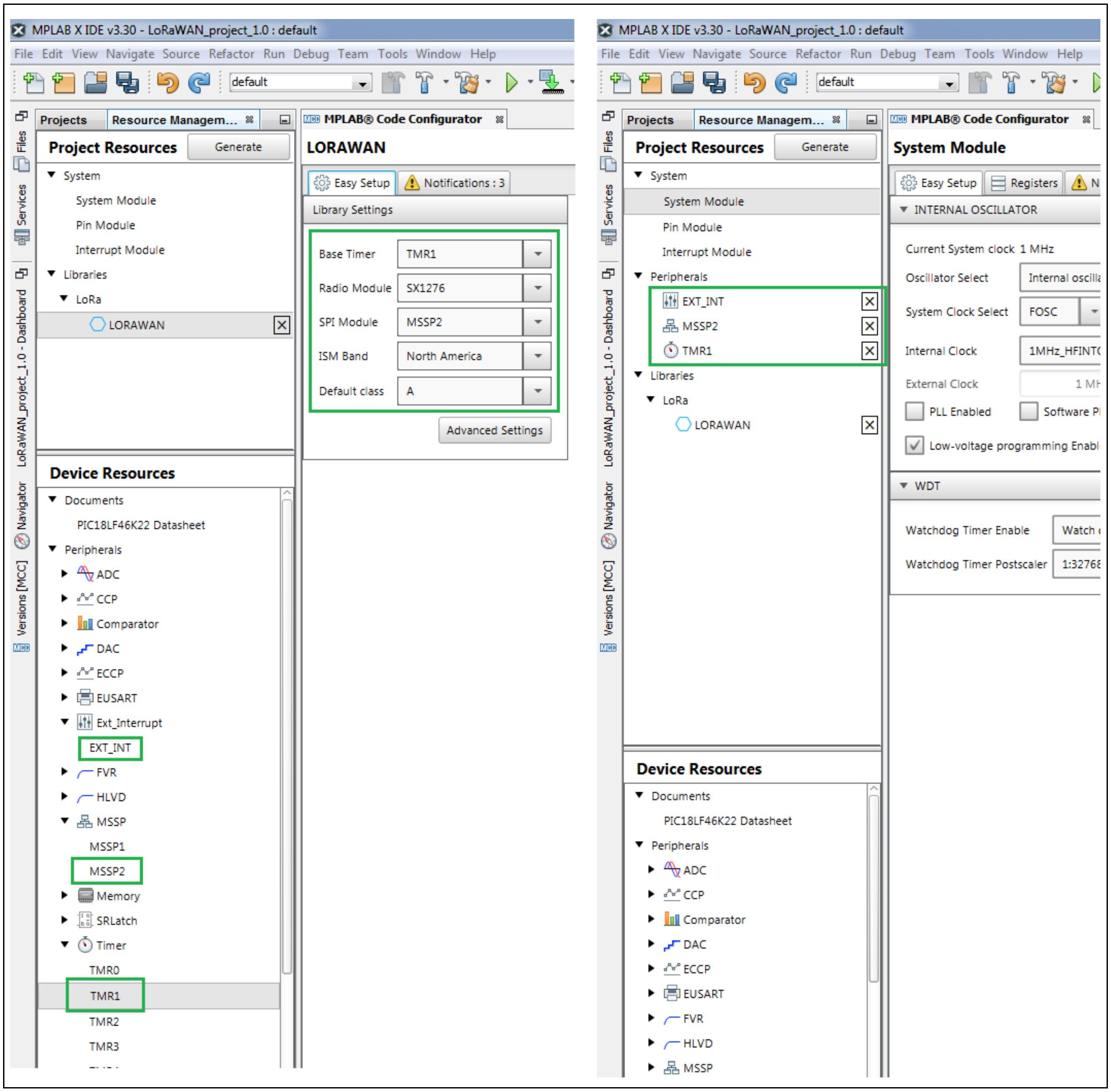
3.2.3.2 ADDING NEEDED RESOURCES TO PROJECT

According to the notifications provided by the LoRaWAN Library during the MCC configuration of the project, the following resources must be added to the project:

- TMR1
- MSSP2
- EXT_INT

The user must add these resources to the Project Resources by double clicking these items in the Device Resources window, as seen in Figure 3-12.

FIGURE 3-12: LoRaWAN™ RN2903 NEEDED RESOURCES



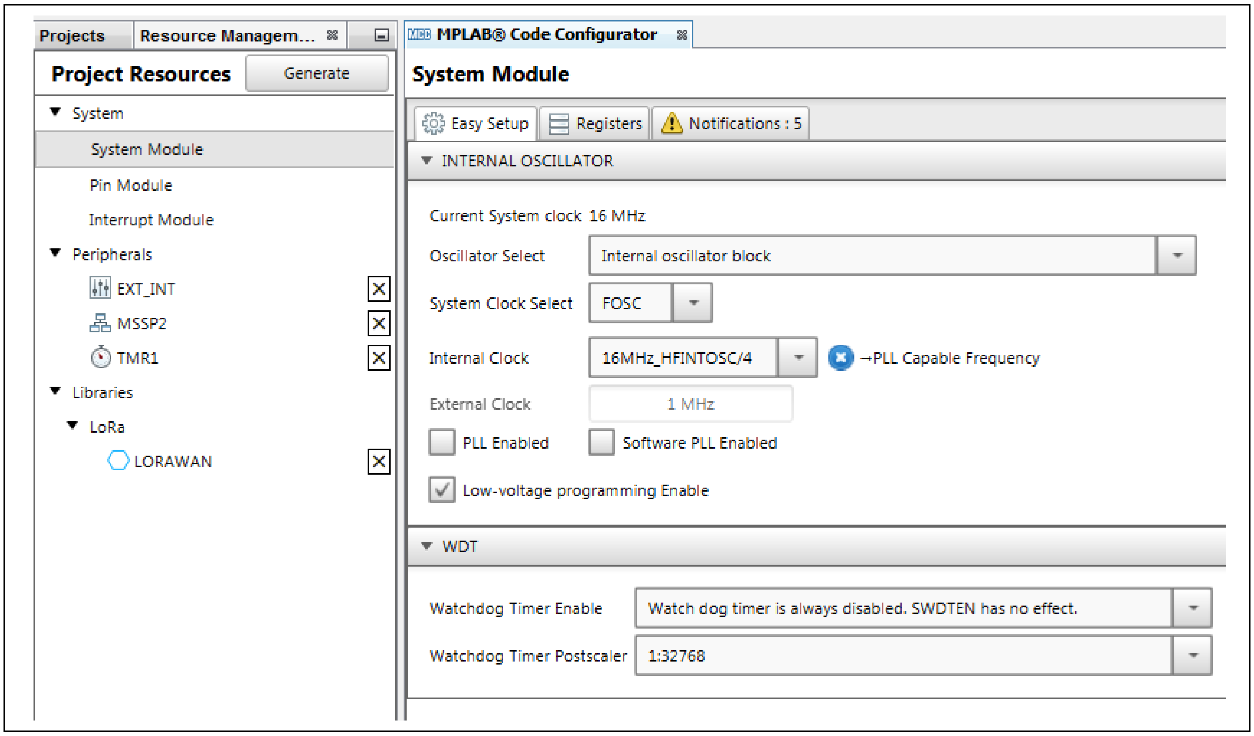
3.2.3.3 LoRaWAN RN2903 System Module Configuration

The user must set the following parameters for the SYSTEM Module (see Figure 3-13):

- Oscillator Select: Internal oscillator block
- System Clock Select: FOSC
- Internal Clock: 16MHz_HFINTOSC/4
- Low-voltage programming Enable (checked)
- Watchdog Timer Enable: Watchdog timer is always disabled; SWDTEN has no effect
- Watchdog Timer Postscaler: 1: 32768

No other modifications are needed on the pre-loaded system configuration done by MCC at start-up.

FIGURE 3-13: LoRaWAN™ RN2903 SYSTEM MODULE CONFIGURATION



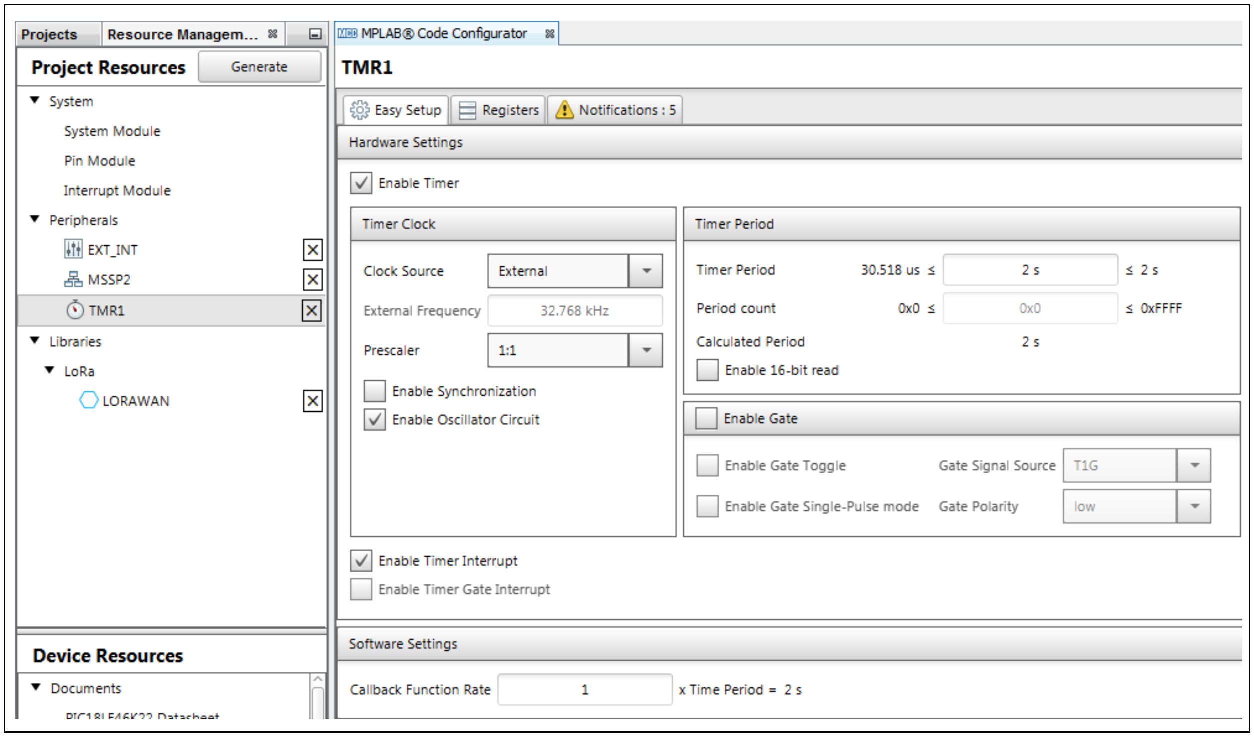
3.2.3.4 LoRaWAN RN2903 TMR1 Module Configuration

The user must set the following parameters for the TMR1 Module (see Figure 3-14):

- Enable Timer: checked
- Timer Clock
  - Clock Source: External @ 32.768 kHz
  - Prescaler: 1:1
  - Enable Synchronization: Not checked
  - Enable Oscillator Circuit: Checked
- Timer Period
  - Timer Period: 2s
  - Period Count: (automatically filled by MCC)
  - Enable 16-bit read: not checked
- Enable Gate: not checked
- Enable Timer Interrupt: checked
- Software Settings
  - Callback Function Rate: 1

No other modifications are needed on the pre-loaded system configuration done by MCC when loading TMR1 resource.

FIGURE 3-14: LoRaWAN™ RN2903 TMR1 MODULE CONFIGURATION



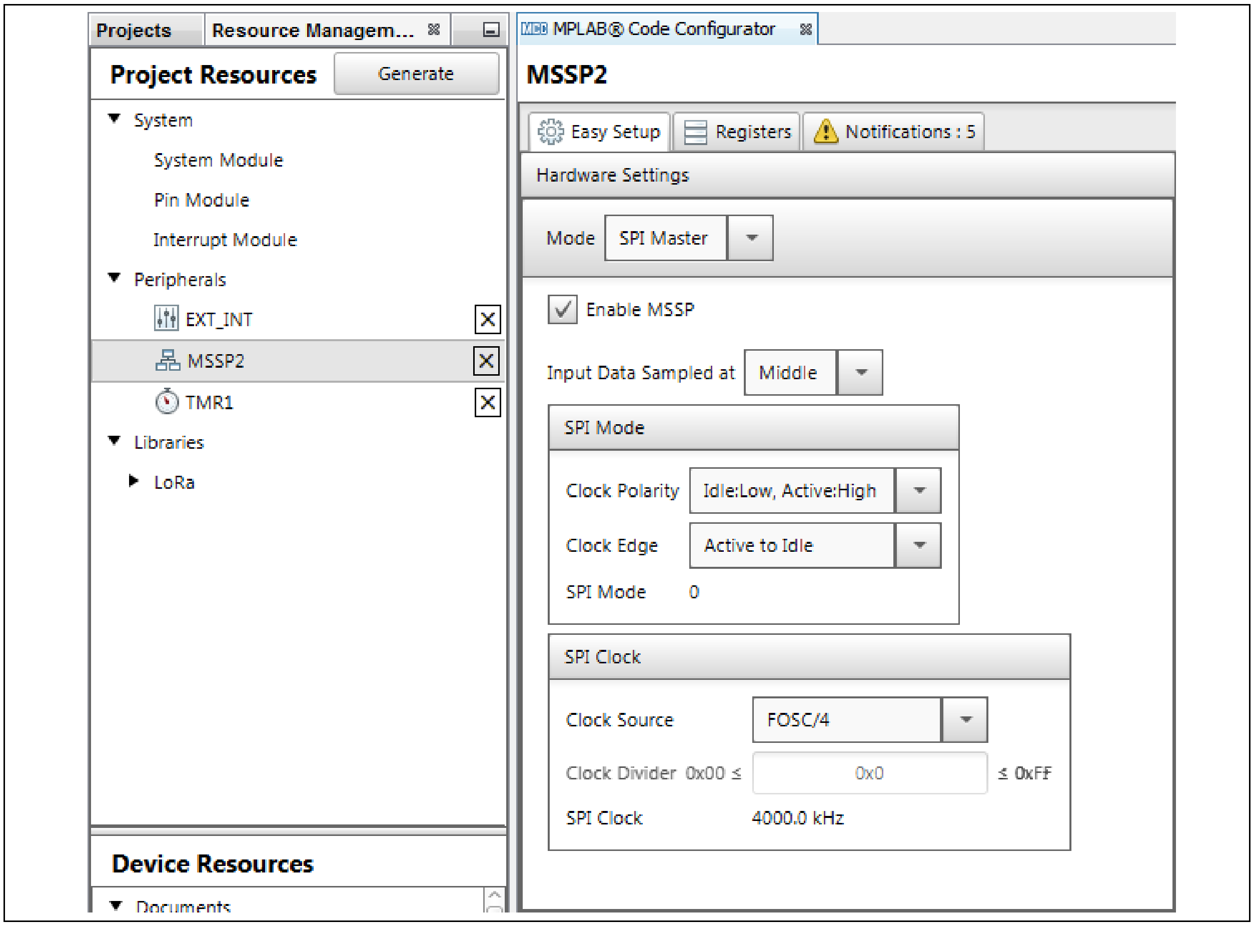
3.2.3.5 LORAWAN RN2903 MSSP2 MODULE CONFIGURATION

The user must set the following parameters for the MSSP2 Module, as shown in Figure 3-15:

- Mode: SPI Master
- Enable MSSP: checked
- Input Data Sampled at: Middle
- SPI Mode
  - Clock Polarity: Idle:Low, Active:High
  - Clock Edge: Active to Idle
  - SPI Mode: 0 (automatically filled by MCC)
- SPI Clock
- Clock Source: FOSC/4
- SPI Clock: 4000.0 kHz (automatically filled by MCC)

No other modifications are needed on the pre-loaded system configuration done by MCC when loading MSSP2 resource.

FIGURE 3-15: LoRaWAN™ RN2903 MSSP2 MODULE CONFIGURATION



3.2.3.6 LoRaWAN RN2903 DIO PINS CONFIGURATION

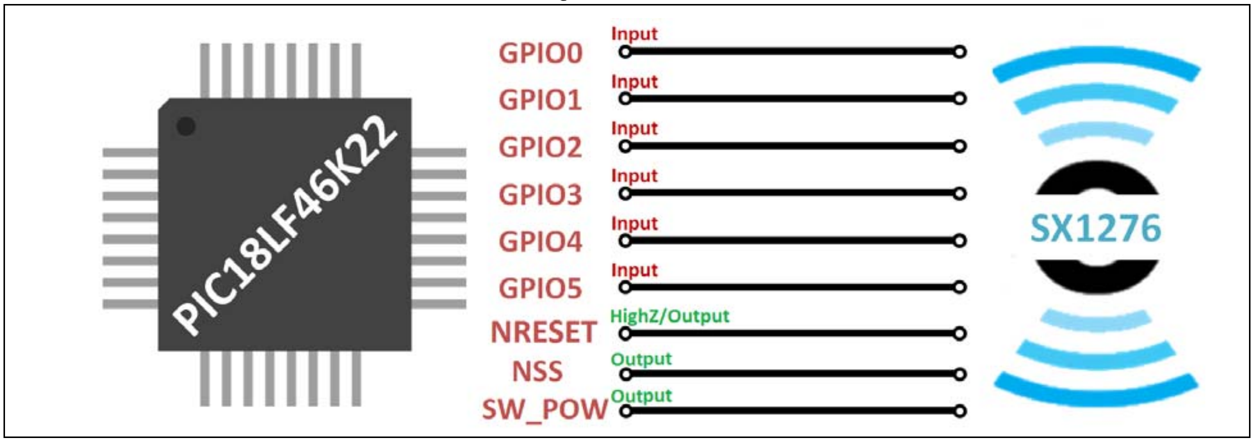
Besides SPI, the LoRaWAN stack further requires six GPIOs (DIOs), in order to communicate with the SX1276 radio transceiver. These GPIOs need to be interrupt-capable and can either have External Interrupt function or IOC (Interrupt-on-Change) function.

Besides these six GPIOs, the LoRaWAN stack also requires a reset control pin (NRESET) and the Chip Select for SPI (NSS) to communicate with the SX1276 radio transceiver and the RF Switch control pin (SW_POW). The RF Switch control pin is required only if RN2903A module is used.

In the ‘Pin Manager: Grid [MCC]’ window, LoRaWAN library makes a filter on the available MCU pins and displays only the interrupt-capable ones.

The configuration below reflects the way the LoRaWAN DIO pins are linked inside the RN2903 module (Figure 3-16).

FIGURE 3-16: LoRaWAN™ RN2903 REQUIRED DIO PINS

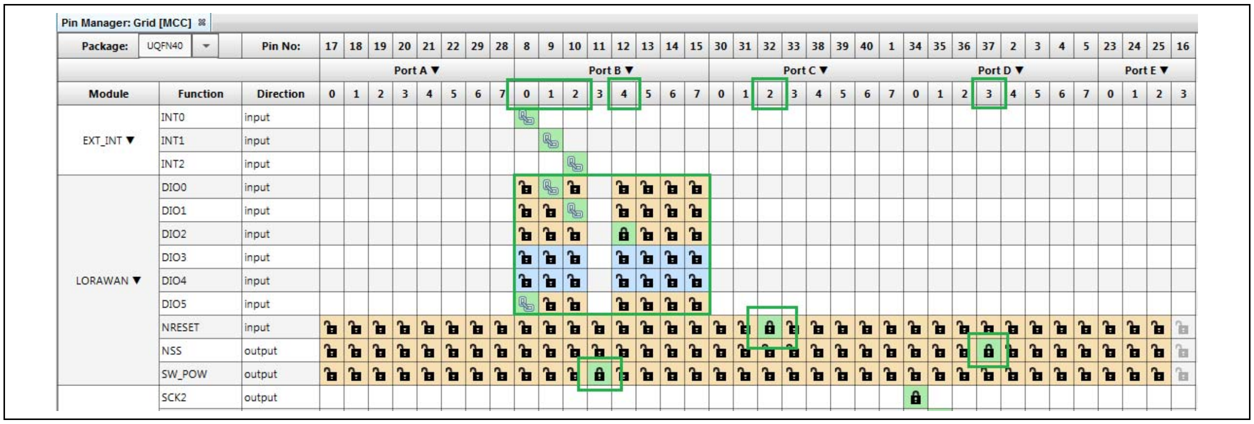


The user must make the following selections, as displayed in Figure 3-17:

- DIO0 – PORTB Pin 1 (RB1)
- DIO1 – PORTB Pin 2 (RB2)
- DIO2 – PORTB Pin 4 (RB4)
- DIO3 – not needed (Reserved for future use)
- DIO4 – not needed (Reserved for future use)
- DIO5 – PORTB Pin 0 (RB0)
- NRESET – PORTC Pin 2 (RC2)
- NSS – PORTD Pin 3 (RD3)
- SW_POW – PORTB Pin 3 (RB3). This pin is needed only for RN2903A module. For RN2483/RN2903, configuration for this pin is not needed.

Note: The NRESET pin needs a special configuration because of the SX127X radio transceiver. The configuration sequence for the NRESET pin is first to be set as High Z and then as an output. For more details, see the Semtech SX127X data sheet. At the time the LoRaWAN Library was developed, MCC was not offering support for High Z pin configuration. In order to get the High Z configuration for the NRESET pin (RC2), the pin must be configured as input. During run-time, the LoRaWAN stack configures the pin as output as soon as the High Z configuration is no longer needed.

FIGURE 3-17: LoRaWAN™ RN2903 DIO PINS CONFIGURATION



Note:	SW_POW pin needs to be configured only if RN2903A module is used. In case RN2903 or RN2483 modules are used, this pin does not need to be configured (if EU433 MHz or EU868 MHz ISM bands are selected, the SW_POW pin does not appear in the "Pin Manager: Grid [MCC]" window).

3.2.3.7 LoRaWAN RN2903 PIN MODULE CONFIGURATION

Note:	The SW_POW pin needs to be configured only if RN2903A module is used. In case RN2903 or RN2483 modules are used, this pin does not need to be configured (if EU433 MHz or EU868 MHz ISM bands are selected, the SW_POW pin does not appear in the "Pin Manager: Grid [MCC]" window).

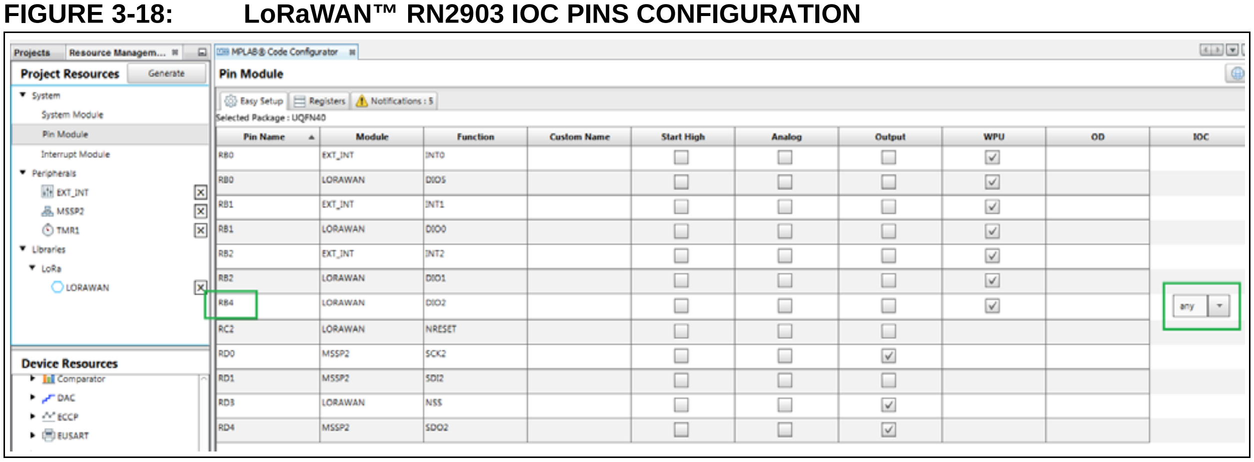
The Pin Module contains the pins of the resources which were added to the project.

There are two modifications which need to be completed:

1. Set the IOC edge for RB4 to any. Click on Pin Module; the setting is on the IOC column of the table, see Figure 3-18):

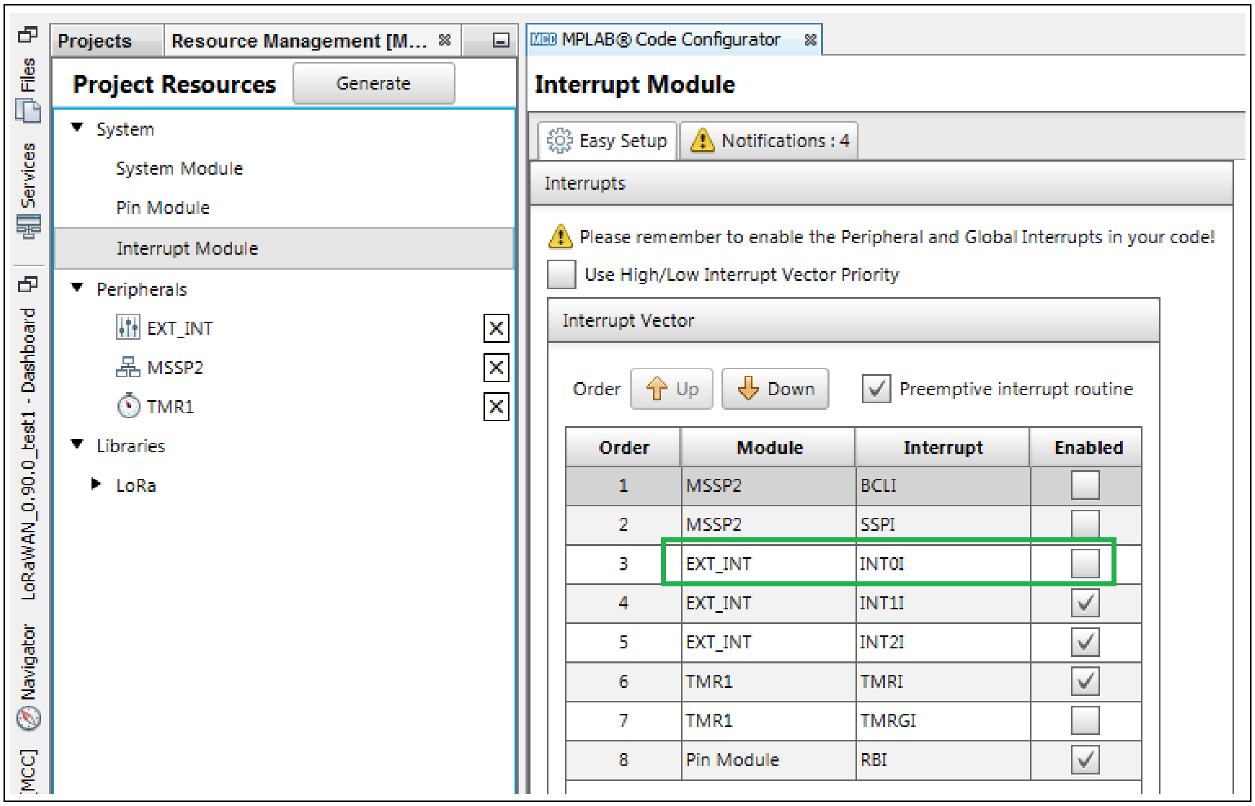
- RB4: IOC any

FIGURE 3-18: LoRaWAN™ RN2903 IOC PINS CONFIGURATION



2. Disable the INT0 interrupt. Because the RB0 pin, corresponding to the INT0 interrupt is only polled by the LoRaWAN stack whenever needed, there is no need to also generate an interrupt on INT0. Click on **Interrupt Module** and clear the EXT_INT - INT0I check box (see Figure 3-19).

FIGURE 3-19: LoRaWAN™ RN2903 INTO DISABLE

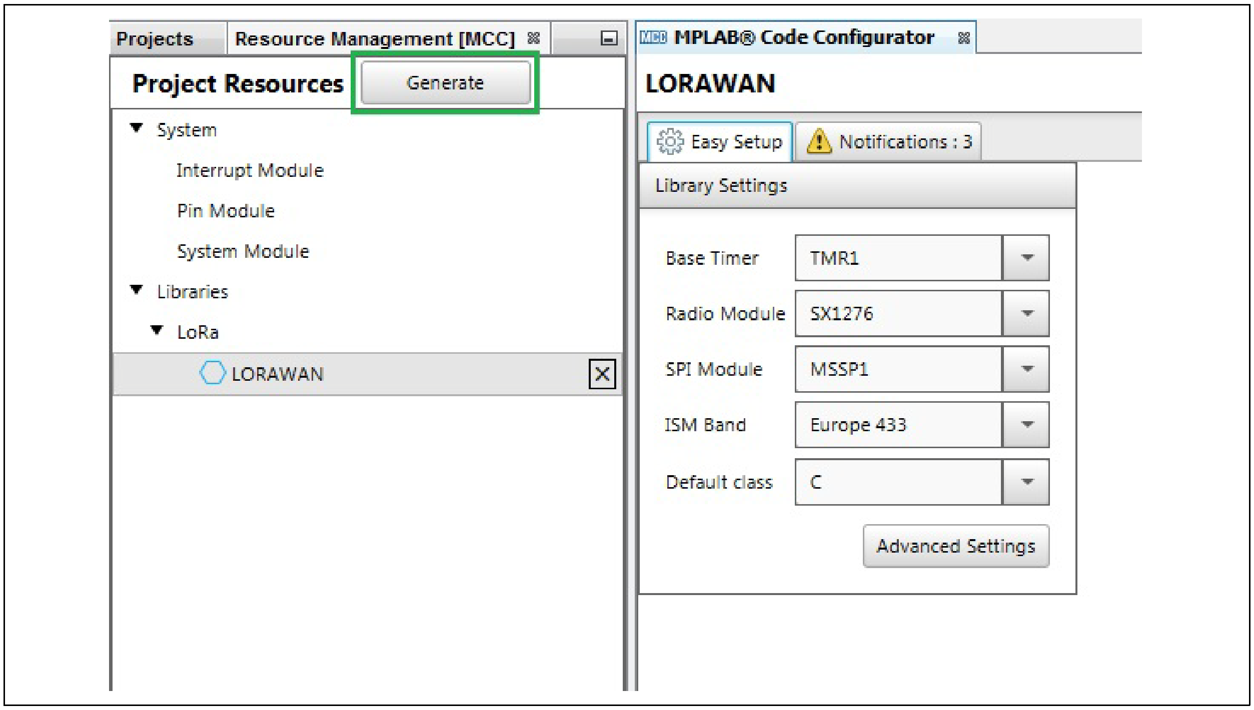


Note: The order of the interrupts in the table might differ from the one in Figure 3-19.

3.3 LoRaWAN™ STACK GENERATION FOR THE RN2XX3 MODULES

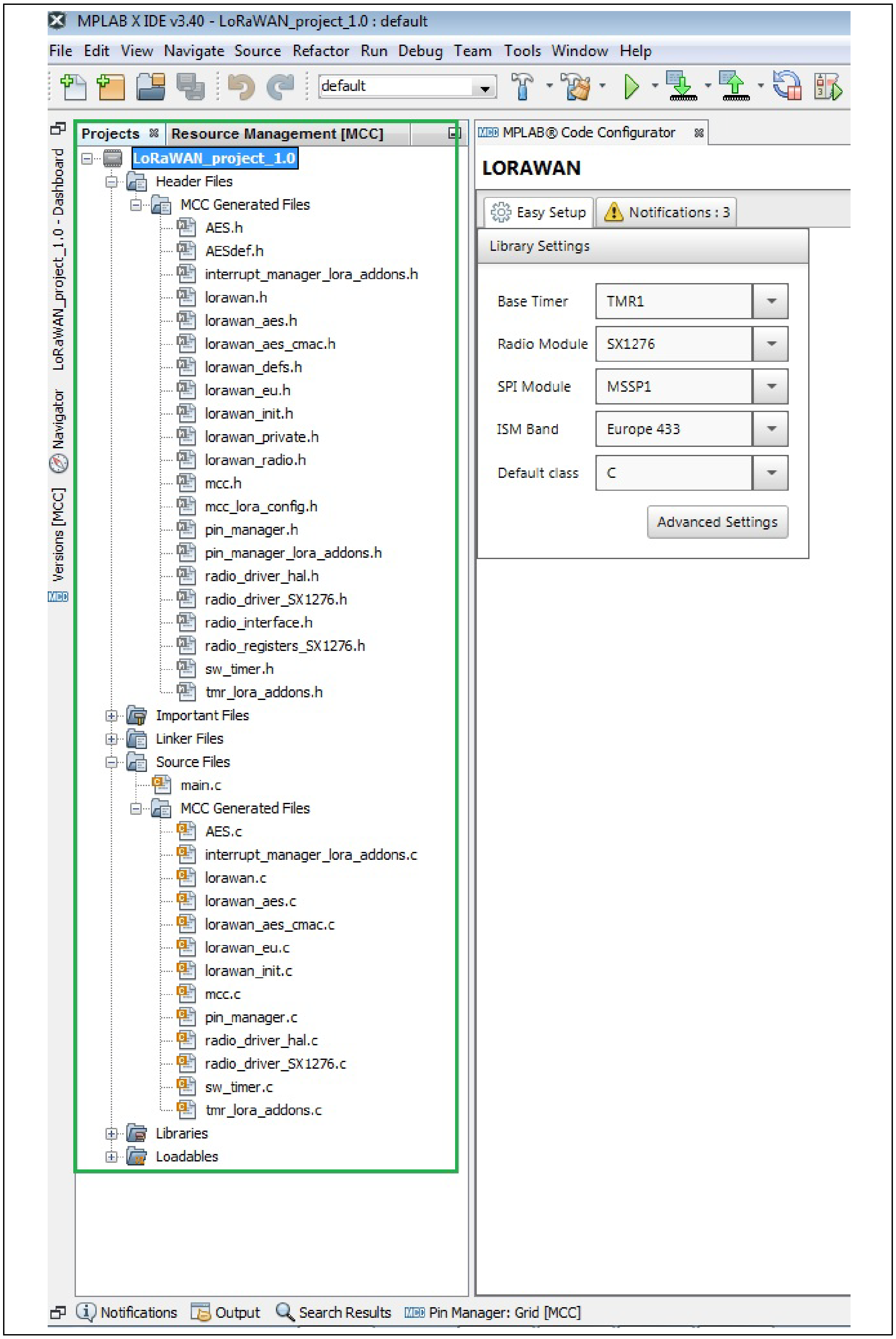
As soon as the project configuration has been made, the LoRaWAN Library can be generated by pressing the **Generate** button, see Figure 3-20.

FIGURE 3-20: LoRaWAN™ STACK GENERATION FOR RN2903



As soon as the Generate button has been pressed, the LoRaWAN stack is available, as shown in Figure 3-21, and the user can start creating the LoRaWAN-based application.

FIGURE 3-21: LoRaWAN™ STACK GENERATED FILES



If the generated files are needed to build a custom LoRaWAN-based application, more information can be found in **Chapter 5. "Building a LoRaWAN-Based Application"**.

Chapter 4. LoRaWAN™ Stack API

4.1 OVERVIEW

The LoRaWAN Stack is customized and generated by the Graphical User Interface of the MPLAB® Code Configurator. An application based on the LoRaWAN stack will use the APIs provided by the stack to make the data transfer using the LoRa® Technology. The following sections explain the Application Programming Interfaces (APIs) of the LoRaWAN stack.

4.2 LoRaWAN ARCHITECTURE

4.2.1 LoRaWAN PROJECT CREATION

A LoRaWAN-based application is organized into four layers:

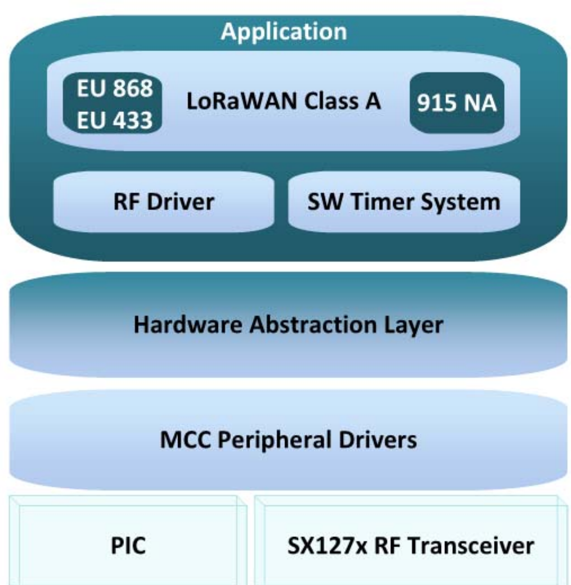
1. The application layer, which contains the user's application(s) (e.g. reading a sensor and processing the data etc.)
2. LoRaWAN Class A Layer:

- LoRaWAN Class A application (for North America or Europe)
- RF Driver
- Software Timer System

3. Hardware Abstraction Layer – provides an interface between peripheral drivers generated by MCC and the Application layer
4. MCC Peripheral Drivers Layer – contains the drivers generated by MCC, according to the project configuration.

Both the LoRaWAN stack and the user custom application run on the PIC device which is present inside the RN2XX3 module.

FIGURE 4-1: LoRaWAN™ STACK ARCHITECTURE



4.2.2 File Structure

The LoRaWAN stack files are split according to the architecture layers. So, the following files are being defined and used:

1. The User Application Layer
   - User application header files

• user_custom_header1.h • user_custom_header2.h • user_custom_header3.h

   - User application source files

• main.c • user_custom_source1.c • user_custom_source2.c • user_custom_source3.c

2. LoRaWAN Stack
   - LoRaWAN header files

• interrupt_manager_lora_addons.h • lorawan.h • lorawan_aes.h • lorawan_aes_cmac.h • lorawan_defs.h • lorawan_init.h • lorawan_na.h • lorawan_eu.h • lorawan_private.h • lorawan_radio.h • AES.h • AESdef.h

   - LoRaWAN source files

• AES.c • interrupt_manager_lora_addons.c • lorawan.c • lorawan_aes.c • lorawan_aes_cmac.c • lorawan_init.c • lorawan_na.c • lorawan_eu.c

- Radio Driver header files

• radio_driver_SX1276.h • radio_driver_SX1272.h • radio_interface.h • radio_registers_SX1276.h • radio_registers_SX1272.h

- Radio Driver source files

• radio_driver_SX1276.h • radio_driver_SX1272.h

- SW Timer System header files

• sw_timer.h • tmr_lora_addons.h

- SW Timer System source files

• sw_timer.c • tmr_lora_addons.c

3. Hardware Abstraction Layer (HAL files)
   - HAL header files

• mcc_lora_config.h • tmr_lora_addons.h • pin_manager_lora_addons.h • radio_driver_hal.h

- HAL source files

• radio_driver_hal.c

4. MCC Peripheral Drivers: all of the files which are generated by MCC for each of the used MCU modules, besides the libraries (e.g.,: SPI2 – `spi2.h` and `spi2.c`; TMR1 – `tmr1.h` and `tmr1.c`).

4.3 APPLICATION PROGRAMMING INTERFACES

4.3.1 Overview

The LoRaWAN stack offers all of the Application Programming Interfaces (APIs) which are needed by the user in the process of building the custom LoRaWAN-based application.

The APIs are defined in the `lorawan.c` file and are declared (together with all the details) in the `lorawan.h` file. Details about the APIs are given in the next sub-chapter.

4.3.2 LoRaWAN APIs

4.3.2.1 LORAWAN_Init

Name	`LORAWAN_Init`
Prototype	`void LORAWAN_Init(RxAppDataCb_t RxPayload, RxJoinResponseCb_t RxJoinResponse)`
Summary	LoRaWAN Initialization function
Description	Initializes LoRaWAN stack and the radio module.
Preconditions	none
Parameters	• `RxPayload` – pointer to function that gets called after the bidirectional communication ended • `RxJoinResponse` – pointer to function that gets called after the activation procedure
Return	none
Example	`LORAWAN_Init(RxData, RxJoinResponse);`

4.3.2.2 LORAWAN_Join

Name	`LORAWAN_Join`
Prototype	`LorawanError_t LORAWAN_Join(ActivationType_t activationTypeNew)`
Summary	LoRaWAN activation procedure
Description	This function starts LoRaWAN activation procedure. This procedure is finished whenever the registered `RxJoinResponse` callback is called.
Preconditions	none
Parameters	• `activationTypeNew` – activation type: `OTAA` or `ABP`
Return	Function returns the status of the operation (`LorawanError_t`)
Example	`LORAWAN_Join(ABP);`

4.3.2.3 LORAWAN_Send

Name	LORAWAN_Send
Prototype	`LorawanError_t LORAWAN_Send (TransmissionType_t confirmed, uint8_t port, void *buffer, uint8_t bufferLength)`
Summary	Bidirectional communication start
Description	This function starts a bidirectional communication process. This procedure is finished whenever the registered `RxPayload` callback is called.
Preconditions	none
Parameters	• `confirmed` – represents the transmission type; can be either UNCNF - unconfirmed or CNF - confirmed (`TransmissionType_t`) • `port` – represents the port on which the transmission is being made; it's a number between 0 and 255 (`uint8_t`) • `buffer` – a data buffer used to store the data to be sent • `bufferLength` – the length in bytes of the data buffer (`uint8_t`)
Return	Function returns the status of the operation (`LorawanError_t`)
Example	`LORAWAN_Send(UNCNF, 4, “Hello World!”, 12);`

4.3.2.4 LORAWAN_SetDeviceEui

Name	LORAWAN_SetDeviceEui
Prototype	`void LORAWAN_SetDeviceEui (uint8_t *deviceEuiNew)`
Summary	Sets the value of the end-device identifier.
Description	This function sets the end-device identifier (DevEUI). The DevEUI is a global end-device ID in IEEE EUI64 address space that uniquely identifies the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `deviceEui` – buffer where the value will be stored.
Return	none

Example	`uint8_t deviceEui[8] = {0x00, 0x04, 0xA3, 0x0B, 0x00, 0x1A, 0x9E, 0xF8};`
	
	`LORAWAN_SetDeviceEui(&deviceEui[0]);`

4.3.2.5 LORAWAN_GetDeviceEui

Name	LORAWAN_GetDeviceEui
Prototype	`void LORAWAN_GetDeviceEui (uint8_t *deviceEui)`
Summary	Gets the value of the end-device identifier.
Description	This function gets the end-device identifier (DevEUI). The DevEUI is a global end-device ID in IEEE EUI64 address space that uniquely identifies the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `deviceEui` - buffer where the value will be stored.
Return	none
Example	`uint8_t deviceEui[8];` `LORAWAN_GetDeviceEui(&deviceEui[0]);`

4.3.2.6 LORAWAN_SetApplicationEui

Name	`LORAWAN_SetApplicationEui`
Prototype	`void LORAWAN_SetApplicationEui (uint8_t *applicationEuiNew)`
Summary	Sets the application identifier.
Description	This function sets the end-device Application identifier (AppEUI). The AppEUI is a global application ID in IEEE EUI64 address space that uniquely identifies the application provider (i.e., owner) of the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationEuiNew` – buffer where AppEUI is stored.
Return	none

Example	`uint8_t applicationEuiNew[8] = {0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88};`
	
	`LORAWAN_SetApplicationEui(&applicationEuiNew[0]);`

4.3.2.7 LORAWAN_GetApplicationEui

Name	`LORAWAN_GetApplicationEui`
Prototype	`void LORAWAN_GetApplicationEui (uint8_t *applicationEui)`
Summary	Gets the value of the application identifier.
Description	This function gets the end-device Application identifier (AppEUI). The AppEUI is a global application ID in IEEE EUI64 address space that uniquely identifies the application provider (i.e., owner) of the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationEui` – buffer where the value will be stored.
Return	none
Example	`uint8_t applicationEui[8];` `LORAWAN_GetApplicationEui(&applicationEui[0]);`

4.3.2.8 LORAWAN_SetDeviceAddress

Name	`LORAWAN_SetDeviceAddress`
Prototype	`void LORAWAN_SetDeviceAddress (uint32_t deviceAddressNew)`
Summary	Sets the end-device address.
Description	This function sets the end-device address (DevAddr). The DevAddr is a 32-bit identifier of the end device within the current network.
Preconditions	none
Parameters	• `deviceAddressNew` – the value of the new address to be set.
Return	none

Example	`uint32_t devAddr = 0x11223344;`
	
	`LORAWAN_SetDeviceAddress(devAddr);`

4.3.2.9 LORAWAN_GetDeviceAddress

Name	LORAWAN_GetDeviceAddress
Prototype	`uint32_t LORAWAN_GetDeviceAddress (void)`
Summary	Returns the end-device address.
Description	This function gets the end-device address (DevAddr). The DevAddr is a 32-bit identifier of the end device within the current network.
Preconditions	none
Parameters	none
Return	device address (`uint32_t` value)

Example	`uint32_t devAddr;`
	
	`devAddr = LORAWAN_GetDeviceAddress();`

4.3.2.10 LORAWAN_SetNetworkSessionKey

Name	LORAWAN_SetNetworkSessionKey
Prototype	`void LORAWAN_SetNetworkSessionKey (uint8_t *networkSessionKeyNew)`
Summary	Sets the network session key.
Description	This function sets the Network Session key (NwkSKey). The NwkSKey is a network session key specific to the end device, used in calculating and verifying the MIC (Message Integrity Code). The NwkSKey is a 16-byte array.
Preconditions	Pointer must be allocated by caller.
Parameters	• `networkSessionKeyNew` – buffer where the value is stored.
Return	none

Example	`uint8_t nwkSKey[16] = {0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C};`
	
	`LORAWAN_SetNetworkSessionKey(nwkSKey);`

4.3.2.11 LORAWAN_GetNetworkSessionKey

Name	`LORAWAN_GetNetworkSessionKey`
Prototype	`void LORAWAN_GetNetworkSessionKey (uint8_t *networkSessionKey)`
Summary	Gets the network session key.
Description	This function gets the Network Session key (NwkSKey). The NwkSKey is a network session key specific to the end device, used in calculating and verifying the MIC (Message Integrity Code). The NwkSKey is a 16-byte array.
Preconditions	Pointer must be allocated by caller.
Parameters	• `networkSessionKey` – buffer where the value will be stored.
Return	none

Example	`uint8_t nwkSKey[16];`
	
	`LORAWAN_GetNetworkSessionKey(nwkSKey);`

4.3.2.12 LORAWAN_SetApplicationSessionKey

Name	`LORAWAN_SetApplicationSessionKey`
Prototype	`void LORAWAN_SetApplicationSessionKey (uint8_t *applicationSessionKeyNew)`
Summary	Sets the application session key.
Description	This function sets the Application Session Key (AppSKey). The AppSKey is an application session key specific to the end device, used to encrypt/decrypt the payload field of the application-specific data messages, and also to calculate/verify an application-level MIC (Message Integrity Code) that may be included in the payload. The AppSKey is a 16-byte array.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationSessionKeyNew` – buffer where the value is stored.
Return	none

Example	`uint8_t appSKey[16] = {0x3C, 0x8F, 0x26, 0x27, 0x39, 0xBF, 0xE3, 0xB7, 0xBC, 0x08, 0x26, 0x99, 0x1A, 0xD0, 0x50, 0x4D};`
	
	`LORAWAN_SetApplicationSessionKey(appSKey);`

4.3.2.13 LORAWAN_GetApplicationSessionKey

Name	`LORAWAN_GetApplicationSessionKey`
Prototype	`void LORAWAN_GetApplicationSessionKey (uint8_t *applicationSessionKey)`
Summary	Gets the application session key.
Description	This function gets the Application Session Key (AppSKey). The AppSKey is an application session key specific to the end device, used to encrypt/decrypt the payload field of the application-specific data messages, and also to calculate/verify an application-level MIC (Message Integrity Code) that may be included in the payload. The AppSKey is a 16-byte array.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationSessionKey` – buffer where the value will be stored.
Return	none

Example	`uint8_t appSKey[16];`
	
	`LORAWAN_GetApplicationSessionKey(appSKey);`

4.3.2.14 LORAWAN_SetApplicationKey

Name	`LORAWAN_SetApplicationKey`
Prototype	`void LORAWAN_SetApplicationKey (uint8_t *applicationKeyNew)`
Summary	Sets the application key.
Description	This function sets the Application Session Key (AppKey). The AppKey is an AES-128 application key specific to the end device, assigned by the application owner to the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationKeyNew` – buffer where the value is stored.
Return	none

Example	`uint8_t appKey[16] = {0x3C, 0x8F, 0x26, 0x27, 0x39, 0xBF, 0xE3, 0xB7, 0xBC, 0x08, 0x26, 0x99, 0x1A, 0xD0, 0x50, 0x4D};`
	
	`LORAWAN_SetApplicationKey(appKey);`

4.3.2.15 LORAWAN_GetApplicationKey

Name	`LORAWAN_GetApplicationKey`
Prototype	`void LORAWAN_GetApplicationKey (uint8_t *applicationKey)`
Summary	Gets the application key.
Description	This function gets the Application Session Key (AppKey). The AppKey is an AES-128 application key specific to the end device, assigned by the application owner to the end device.
Preconditions	Pointer must be allocated by caller.
Parameters	• `applicationKey` – buffer where the value is stored.
Return	none

Example	`uint8_t appKey[16];`
	
	`LORAWAN_GetApplicationKey(appKey);`

4.3.2.16 LORAWAN_SetAdr

Name	`LORAWAN_SetAdr`
Prototype	`void LORAWAN_SetAdr (bool status)`
Summary	Sets the adaptive data rate mode.
Description	This function sets the Adaptive Data Rate (ADR) mode. LoRa network allows end devices to individually use any of the possible data rates, which is referred to as the Adaptive Data Rate (ADR).
Preconditions	none
Parameters	status – true/false
Return	none
Example	`LORAWAN_SetAdr (true);`

4.3.2.17 LORAWAN_GetAdr

Name	LORAWAN_GetAdr
Prototype	`bool LORAWAN_GetAdr (void)`
Summary	Returns the adaptive data rate mode.
Description	This function returns the Adaptive Data Rate (ADR) mode. LoRa network allows end devices to individually use any of the possible data rates, which is referred to as the Adaptive Data Rate (ADR). If the ADR is set, the network will control the data rate of the end device through the appropriate MAC commands. If the ADR is not set, the network will not attempt to control the data rate of the end device, regardless of the signal quality received. If the ADR is set, the network will control the data rate of the end device through the appropriate MAC commands. If the ADR is not set, the network will not attempt to control the data rate of the end device, regardless of the signal quality received.
Preconditions	none
Parameters	none
Return	status – true/false

Example	`bool adrStatus;`
	
	`adrStatus = LORAWAN_GetAdr();`

4.3.2.18 LORAWAN_GetCurrentDataRate

Name	LORAWAN_GetCurrentDataRate
Prototype	`uint8_t LORAWAN_GetCurrentDataRate (void)`
Summary	Returns the data rate mode for the next uplink transmission.
Description	Communication between end devices and gateways is spread out on different frequency channels and data rates. The selection of the data rate is a trade-off between the communication range and the message duration; still, communications with different data rates do not interfere with each other.
Preconditions	none
Parameters	• `valueNew` – new data rate value
Return	Returns the current data rate (`uint8_t`)

Example	`uint8_t dataRateUsed;`
	
	`dataRateUsed = LORAWAN_GetCurrentDataRate();`

4.3.2.19 LORAWAN_SetTxPower

Name	`LORAWAN_SetTxPower`
Prototype	`LorawanError_t LORAWAN_SetTxPower (uint8_t txPowerNew)`
Summary	Sets the TX output power.
Description	The TX output power (TXPower) is region-specific. `txPowerNew` must be provided as an index between 0 - 15. For more details, refer to the LoRaWAN Specification V1.0 document.
Preconditions	none
Parameters	• `txPowerNew` – new TX power value
Return	Returns LoRaWAN Error type (`LorawanError_t`)
Example	`LORAWAN_SetTxPower (4);`

4.3.2.20 LORAWAN_GetTxPower

Name	`LORAWAN_GetTxPower`
Prototype	`uint8_t LORAWAN_GetTxPower (void)`
Summary	Returns the TX output power.
Description	The TX output power (TXPower) is region-specific. `txPower` is returned as an index between 0 - 15. For more details, refer to the LoRaWAN Specification V1.0 document.
Preconditions	none
Parameters	• `txPowerNew` – new TX power value
Return	Returns LoRaWAN Error type (`LorawanError_t`)

Example	`uint8_t txPowerUsed;`
	
	`txPowerUsed = LORAWAN_GetTxPower();`

4.3.2.21 LORAWAN_SetSyncWord

Name	`LORAWAN_SetSyncWord`
Prototype	`void LORAWAN_SetSyncWord (uint8_t syncWord)`
Summary	Sets the synchronization word.
Description	This function sets the current synchronization word used during communication. For more details, refer to the LoRaWAN Specification V1.0 document.
Preconditions	none
Parameters	• `syncWord` – the value for the new sync word
Return	none

Example	`uint8_t syncWordToSet = 0x34;`
	
	`LORAWAN_SetSyncWord(syncWordToSet);`

4.3.2.22 LORAWAN_GetSyncWord

Name	`LORAWAN_GetSyncWord`
Prototype	`uint8_t LORAWAN_GetSyncWord (void)`
Summary	Returns the synchronization word.
Description	This function returns the current synchronization word used during communication. For more details, refer to the LoRaWAN Specification V1.0 document.
Preconditions	none
Parameters	none
Return	The value of the sync word (`uint8_t`)

Example	`uint8_t syncWordUsed;`
	
	`syncWordUsed = LORAWAN_SetSyncWord();`

4.3.2.23 LORAWAN_SetUplinkCounter

Name	`LORAWAN_SetUplinkCounter`
Prototype	`void LORAWAN_SetUplinkCounter (uint32_t ctr)`
Summary	Sets the current uplink counter.
Description	This function sets the current uplink counter used during communication. This may be used to synchronize the uplink counter with the value stored by the server.
Preconditions	none
Parameters	• `ctr` – the value of the new counter to be set
Return	none
Example	`LORAWAN_SetUplinkCounter(1000);`

4.3.2.24 LORAWAN_GetUplinkCounter

Name	`LORAWAN_GetUplinkCounter`
Prototype	`uint32_t LORAWAN_GetUplinkCounter (void)`
Summary	Returns the current uplink counter.
Description	This function returns the current uplink counter used during communication. This may be used to synchronize the uplink counter with the value stored by the server.
Preconditions	none
Parameters	Current uplink counter (`uint32_t`)
Return	none

Example	`uint32_t uplinkCntUsed;`
	
	`LORAWAN_GetUplinkCounter(uplinkCntUsed);`

4.3.2.25 LORAWAN_SetDownlinkCounter

Name	`LORAWAN_SetDownlinkCounter`
Prototype	`void LORAWAN_SetDownlinkCounter (uint32_t ctr`
Summary	Sets the current downlink counter.
Description	This function sets the current downlink counter used during communication. This may be used to synchronize the downlink counter with the value stored by the server.
Preconditions	none
Parameters	• `ctr` – the value of the new counter
Return	none
Example	`LORAWAN_SetDownlinkCounter(1500);`

4.3.2.26 LORAWAN_GetDownlinkCounter

Name	`LORAWAN_GetDownlinkCounter`
Prototype	`uint32_t LORAWAN_GetDownlinkCounter (void`
Summary	Returns the current downlink counter.
Description	This function returns the current downlink counter used during communication. This may be used to synchronize the downlink counter with the value stored by the server.
Preconditions	none
Parameters	Current downlink counter (`uint32_t`)
Return	none

Example	`uint32_t downlinkCntUsed;`
	
	`LORAWAN_GetDownlinkCounter(downlinkCntUsed);`

4.3.2.27 LORAWAN_SetReceiveDelay1

Name	`LORAWAN_SetReceiveDelay1`
Prototype	`void LORAWAN_SetReceiveDelay1 (uint16_t receiveDe-lay1New)`
Summary	Sets the value for the first receive delay (RECEIVE_DELAY1).
Description	This function will set the delay between the transmission and the first Reception window. The delay between the transmission and the second Reception window is calculated in software as the delay between the transmission and the first Reception window + 1000 (in milliseconds).
Preconditions	none
Parameters	• `receiveDelay1New` – value of the new delay (must be provided in milliseconds).
Return	none
Example	`LORAWAN_SetReceiveDelay1(1100);`

4.3.2.28 LORAWAN_GetReceiveDelay1

Name	`LORAWAN_GetReceiveDelay1`
Prototype	`uint16_t LORAWAN_GetReceiveDelay1 (void)`
Summary	Returns the value for the first receive delay (RECEIVE_DELAY1).
Description	This function will return the delay between the transmission and the first Reception window. The delay between the transmission and the second Reception window is calculated in software as the delay between the transmission and the first Reception window + 1000 (in milliseconds).
Preconditions	none
Parameters	none
Return	Value of the receive delay (is returned in milliseconds - `uint16_t`)

Example	`uint16_t receiveDelay1Used;`
	
	`receiveDelay1Used = LORAWAN_GetReceiveDelay1();`

4.3.2.29 LORAWAN_GetReceiveDelay2

Name	`LORAWAN_GetReceiveDelay2`
Prototype	`uint16_t LORAWAN_GetReceiveDelay2 (void)`
Summary	Returns the value for the second receive delay (RECEIVE_DELAY2).
Description	This function will return the delay between the transmission and the second Reception window. The delay between the transmission and the second Reception window is calculated in software as the delay between the transmission and the first Reception window + 1000 (in milliseconds).
Preconditions	none
Parameters	none
Return	Value of the second receive delay (is returned in milliseconds - `uint16_t`)

Example	`uint16_t receiveDelay2Used;`
	
	`receiveDelay2Used = LORAWAN_GetReceiveDelay2();`

4.3.2.30 LORAWAN_SetJoinAcceptDelay1

Name	`LORAWAN_SetJoinAcceptDelay1`
Prototype	`void LORAWAN_SetJoinAcceptDelay1 (uint16_t joinAcceptDelay1New)`
Summary	Sets the value for the first join accept delay (JOIN_ACCEPT_DELAY1).
Description	The network server will respond to the join-request message with a join-accept message if the end device is permitted to join a network. The join-accept message is sent like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2, respectively).
Preconditions	none
Parameters	• `joinAcceptDelay1New` – the value of the new join accept delay; it must be provided in milliseconds
Return	none
Example	`LORAWAN_SetJoinAcceptDelay1(5500);`

4.3.2.31 LORAWAN_GetJoinAcceptDelay1

Name	`LORAWAN_GetJoinAcceptDelay1`
Prototype	`uint16_t LORAWAN_GetJoinAcceptDelay1 (void)`
Summary	Returns the value for the first join accept delay (JOIN_ACCEPT_DELAY1).
Description	The network server will respond to the join-request message with a join-accept message if the end device is permitted to join a network. The join-accept message is sent like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2, respectively).
Preconditions	none
Parameters	none
Return	Value of the first join accept delay (will be returned in milliseconds – `uint16_t`)

Example	`uint16_t joinAcceptDelay1Used;`
	
	`joinAcceptDelay1Used = LORAWAN_GetJoinAcceptDelay1();`

4.3.2.32 LORAWAN_SetJoinAcceptDelay2

Name	`LORAWAN_SetJoinAcceptDelay2`
Prototype	`void LORAWAN_SetJoinAcceptDelay2 (uint16_t joinAcceptDelay2New)`
Summary	Sets the value for the second join accept delay (JOIN_ACCEPT_DELAY2).
Description	The network server will respond to the join-request message with a join-accept message if the end device is permitted to join a network. The join-accept message is sent like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2, respectively).
Preconditions	none
Parameters	• `joinAcceptDelay2New` – the value of the new join accept delay (must be provided in milliseconds)
Return	none
Example	`LORAWAN_SetJoinAcceptDelay2(6500);`

4.3.2.33 LORAWAN_GetJoinAcceptDelay2

Name	`LORAWAN_GetJoinAcceptDelay2`
Prototype	`uint16_t LORAWAN_GetJoinAcceptDelay2 (void)`
Summary	Returns the value for the second join accept delay (JOIN_ACCEPT_DELAY2).
Description	The network server will respond to the join-request message with a join-accept message if the end device is permitted to join a network. The join-accept message is sent like a normal downlink but uses delays JOIN_ACCEPT_DELAY1 or JOIN_ACCEPT_DELAY2 (instead of RECEIVE_DELAY1 and RECEIVE_DELAY2, respectively).
Preconditions	none
Parameters	none
Return	Value of the first join accept delay (will be returned in milliseconds – `uint16_t`)

Example	`uint16_t joinAcceptDelay2Used;`
	
	`joinAcceptDelay2Used = LORAWAN_GetJoinAcceptDelay2();`

4.3.2.34 LORAWAN_SetMaxFcntGap

Name	`LORAWAN_SetMaxFcntGap`
Prototype	`void LORAWAN_SetMaxFcntGap (uint16_t maxFcntGapNew)`
Summary	Sets the value for the maximum frame counter gap (MAX_FCNT_GAP).
Description	Each end device has two frame counters to keep track of the number of data frames sent uplink to the network server (`FCntUp`), incremented by the end device and received by the end-device downlink from the network server (`FCntDown`), which is incremented by the network server. At the receiver side, the corresponding counter is kept in sync with the value received, provided this value received has incremented compared to the current counter value and is lower than the value specified by MAX-_FCNT_GAP after considering the counter rollovers. If this difference is greater than the value of MAX_FCNT_GAP, then too many data frames have been lost and will be subsequently discarded.
Preconditions	none
Parameters	• `maxFcntGapNew` – the value for the new maximum frame counter
Return	none
Example	`LORAWAN_SetMaxFcntGap(3000);`

4.3.2.35 LORAWAN_GetMaxFcntGap

Name	`LORAWAN_GetMaxFcntGap`
Prototype	`uint16_t LORAWAN_GetMaxFcntGap (void)`
Summary	Sets the value for the adaptive data rate acknowledge limit (ADR_ACK_LIMIT).
Description	Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After the ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (ADRACKReq). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame will reset the ADR_ACK_CNT counter.
Preconditions	none
Parameters	none
Return	Value of the maximum frame counter

Example	`uint16_t maxFcntGapUsed;`
	
	`maxFcntGapUsed = LORAWAN_GetMaxFcntGap();`

4.3.2.36 LORAWAN_SetAdrAckLimit

Name	`LORAWAN_SetAdrAckLimit`
Prototype	`void LORAWAN_SetAdrAckLimit (uint8_t adrAckLimitNew)`
Summary	Sets the value for the adaptive data rate acknowledge limit (ADR_ACK_LIMIT).
Description	Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After the ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (ADRACKReq). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame will reset the ADR_ACK_CNT counter.
Preconditions	none
Parameters	• `adrAckLimitNew` - the new value (`uint8_t`)
Return	none
Example	`LORAWAN_SetAdrAckLimit(5);`

4.3.2.37 LORAWAN_GetAdrAckLimit

Name	`LORAWAN_GetAdrAckLimit`
Prototype	`uint8_t LORAWAN_GetAdrAckLimit (void)`
Summary	Returns the value for the adaptive data rate acknowledge limit (ADR_ACK_LIMIT).
Description	Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After the ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (ADRACKReq). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame will reset the ADR_ACK_CNT counter.
Preconditions	none
Parameters	none
Return	Value of the limit (`uint8_t`).
Example	`uint8_t adrAckLimitUsed;` `adrAckLimitUsed = LORAWAN_GetAdrAckLimit();`

4.3.2.38 LORAWAN_SetAdrAckDelay

Name	`LORAWAN_SetAdrAckDelay`
Prototype	`void LORAWAN_SetAdrAckDelay(uint8_t adrAckDelayNew)`
Summary	Sets the value for the adaptive data rate acknowledge delay (ADR_ACK_DELAY).
Description	Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After the ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (ADRACKReq). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame will reset the ADR_ACK_CNT counter.
Preconditions	none
Parameters	• `adrAckDelayNew` - the new value (`uint8_t`)
Return	none
Example	`LORAWAN_SetAdrAckDelay(20);`

4.3.2.39 LORAWAN_GetAdrAckDelay

Name	`LORAWAN_GetAdrAckDelay`
Prototype	`uint8_t LORAWAN_GetAdrAckDelay (void)`
Summary	Returns the value for the adaptive data rate acknowledge delay (ADR_ACK_DELAY).
Description	Each time the uplink frame counter is incremented (for each new uplink, repeated transmissions do not increase the counter), the device increments an ADR_ACK_CNT counter. After the ADR_ACK_LIMIT uplinks (ADR_ACK_CNT >= ADR_ACK_LIMIT) without any downlink response, it sets the ADR acknowledgment request bit (ADRACKReq). The network is required to respond with a downlink frame within the next ADR_ACK_DELAY frames, any received downlink frame following an uplink frame will reset the ADR_ACK_CNT counter.
Preconditions	none
Parameters	none
Return	Value of the delay (`uint8_t`).

Example	`uint8_t adrAckDelayUsed;`
	
	`adrAckDelayUsed = LORAWAN_GetAdrAckDelay();`

4.3.2.40 LORAWAN_SetAckTimeout

Name	LORAWAN_SetAckTimeout
Prototype	`void LORAWAN_SetAckTimeout(uint16_t ackTimeoutNew)`
Summary	Sets the value for the acknowledge timeout (ACK_TIMEOUT).
Description	If an end device does not receive a frame with the ACK bit set in one of the two receive windows immediately following the uplink transmission, it may resend the same frame with the same payload and frame counter, for at least ACK_TIMEOUT seconds after the second reception window.
Preconditions	none
Parameters	• `ackTimeoutNew` - the new value of the time out; the new value must be provided in milliseconds
Return	none
Example	`LORAWAN_SetAckTimeout(2500);`

4.3.2.41 LORAWAN_GetAckTimeout

Name	LORAWAN_GetAckTimeout
Prototype	`uint16_t LORAWAN_GetAckTimeout (void)`
Summary	Returns the value for the acknowledge time out (ACK_TIMEOUT).
Description	If an end device does not receive a frame with the ACK bit set in one of the two receive windows immediately following the uplink transmission, it may resend the same frame with the same payload and frame counter, for at least ACK_TIMEOUT seconds after the second reception window.
Preconditions	none
Parameters	none
Return	The value of the acknowledge time out (`uint16_t`)

Example	`uint16_t ackTimeoutUsed;`
	
	`ackTimeoutUsed = LORAWAN_GetAckTimeout();`

4.3.2.42 LORAWAN_SetNumberOfRetransmissions

Name	LORAWAN_SetNumberOfRetransimissions
Prototype	`void LORAWAN_SetNumberOfRetransmissions (uint8_t numberRetransmissions)`
Summary	Sets the number of retransmissions.
Description	This function sets the number of retransmissions to be used for an uplink confirmed packet, if no downlink acknowledgment is received from the server. At Reset, the number of retransmissions is defaulted to 7.
Preconditions	none
Parameters	• `numberRetransmissions` - the new value (`uint8_t`)
Return	none
Example	`LORAWAN_SetNumberOfRetransimissions(3);`

4.3.2.43 LORAWAN_GetNumberOfRetransmissions

Name	`LORAWAN_GetNumberOfRetransimissions`
Prototype	`uint8_t LORAWAN_GetNumberOfRetransmissions (void)`
Summary	Returns the number of retransmissions.
Description	This function returns the number of retransmissions to be used for an uplink confirmed packet, if no downlink acknowledgment is received from the server.
Preconditions	none
Parameters	none
Return	The number of retransmissions (`uint8_t`)

Example	`uint8_t noOfRetrUsed;`
	
	`noOfRetrUsed = LORAWAN_GetNumberOfRetransmissions();`

4.3.2.44 LORAWAN_SetReceiveWindow2Parameters

Name	`LORAWAN_SetReceiveWindow2Parameters`
Prototype	`LorawanError_t LORAWAN_SetReceiveWindow2Parameters (uint32_t frequency, uint8_t dataRate)`
Summary	Sets the parameters for the second Receive window (RX2).
Description	This function sets the data rate and frequency used for the second Receive window. The configuration of the Receive window parameters should be in concordance with the server configuration.
Preconditions	none
Parameters	• `frequency` – the new frequency (must be provided in Hz) • `dataRate` – the new data rate
Return	Returns LoRaWAN Error type (`LorawanError_t`)
Example	`LORAWAN_SetReceiveWindow2Parameters(868100000, 3);`

4.3.2.45 LORAWAN_GetReceiveWindow2Parameters

Name	`LORAWAN_GetReceiveWindow2Parameters`
Prototype	`void LORAWAN_GetReceiveWindow2Parameters (uint32_t* frequency, uint8_t* dataRate)`
Summary	Gets the parameters for the second Receive window (RX2).
Description	This function gets the data rate and frequency used for the second Receive window. The configuration of the Receive window parameters should be in concordance with the server configuration.
Preconditions	none
Parameters	• `frequency` – pointer to the frequency in Hz (32-bit value) • `data rate` – pointer to the data rate (8-bit value)
Return	none

Example	`uint32_t win2FreqUsed;`
	`uint8_t win2DataRateUsed;`
	
	`LORAWAN_GetReceiveWindow2Parameters(&win2FreqUsed, &win2DataRateUsed);`

4.3.2.46 LORAWAN_SetBattery

Name	`LORAWAN_SetBattery`
Prototype	`void LORAWAN_SetBattery (uint8_t batteryLevelNew)`
Summary	Sets the battery.
Description	This function sets the battery level required for the Device Status Answer frame in use with the LoRaWAN protocol. The level is a decimal number representing the level of the battery, from 0 to 255; 0 stands for external power, 1 for low level, 254 for high level and 255 implies the end device was not able to measure the battery level.
Preconditions	none
Parameters	• `batteryLevelNew` - the new level value
Return	none
Example	`LORAWAN_SetBattery(0);`

4.3.2.47 LORAWAN_GetPrescaler

Name	`LORAWAN_GetPrescaler`
Prototype	`uint16_t LORAWAN_GetPrescaler (void)`
Summary	Returns the duty cycle prescaler value.
Description	This function returns the duty cycle prescaler. The value of the prescaler can be configured ONLY by the SERVER through the use of the Duty Cycle Request frame. Upon the reception of this command from the server, the duty cycle prescaler is changed for all enabled channels.
Preconditions	none
Parameters	none
Return	The value of the prescaler (`uint16_t`)

Example	`uint16_t prescalerUsed;`
	
	`prescalerUsed = LORAWAN_GetPrescaler();`

4.3.2.48 LORAWAN_SetAutomaticReply

Name	`LORAWAN_SetAutomaticReply`
Prototype	`void LORAWAN_SetAutomaticReply (bool status)`
Summary	Sets the Automatic Reply mode state.
Description	This function sets the state of the automatic reply. By enabling the automatic reply, the module will transmit a packet without a payload immediately after a confirmed downlink message is received, or when the Frame Pending bit has been set by the server. If set to OFF, no automatic reply will be transmitted.
Preconditions	none
Parameters	• `status` – on/off (true/false)
Return	none
Example	`LORAWAN_SetAutomaticReply(false);`

4.3.2.49 LORAWAN_GetAutomaticReply

Name	LORAWAN_GetAutomaticReply
Prototype	`bool LORAWAN_GetAutomaticReply (void)`
Summary	Returns the Automatic Reply mode state.
Description	This function returns the state of the automatic reply. By enabling the automatic reply, the module will transmit a packet without a payload immediately after a confirmed downlink message is received, or when the Frame Pending bit has been set by the server. If set to OFF, no automatic reply will be transmitted.
Preconditions	none
Parameters	none
Return	Returns the mode (true/false) for the automatic reply

Example	`bool autoReplyStatus;`
	
	`autoReplyStatus = LORAWAN_GetAutomaticReply();`

4.3.2.50 LORAWAN_GetStatus

Name	LORAWAN_GetStatus
Prototype	`uint32_t LORAWAN_GetStatus (void)`
Summary	Returns the status of the module.
Description	This function will return the current status of the module. The value returned is a bit mask represented in hexadecimal form. For the significance of the bit mask, refer to *“RN2903 LoRa™ Technology Module Command Reference User’s Guide”* (DS40001811).
Preconditions	none
Parameters	none
Return	The status of the module (`uint32_t`)

Example	`uint32_t moduleStatus;`
	
	`moduleStatus = LORAWAN_GetStatus();`

4.3.2.51 LORAWAN_GetLinkCheckMargin

Name	`LORAWAN_GetLinkCheckMargin`
Prototype	`uint8_t LORAWAN_GetLinkCheckMargin (void)`
Summary	Returns a decimal number representing the demodulation margin.
Description	This function will return the demodulation margin as received in the last Link Check Answer frame. Refer to the LoRaWAN Specification 1.0 document for the description of the values.
Preconditions	none
Parameters	none
Return	Margin value (`uint8_t`)
Example	`uint8_t linkCheckMarginUsed;` `linkCheckMarginUsed = LORAWAN_GetLinkCheckMargin();`

4.3.2.52 LORAWAN_GetLinkCheckGwCnt

Name	`LORAWAN_GetLinkCheckGwCnt`
Prototype	`uint8_t LORAWAN_GetLinkCheckGwCnt (void)`
Summary	Returns a decimal number representing the number of gateways.
Description	This function will return the number of gateways that successfully received the last Link Check Request frame command, as received in the last Link Check Answer.
Preconditions	none
Parameters	none
Return	Number of gateways (`uint8_t`)

Example	`uint8_t linkCheckGwCntUsed;`
	
	`linkCheckGwCntUsed = LORAWAN_GetLinkCheckGwCnT();`

4.3.2.53 LORAWAN_GetFrequency

Name	LORAWAN_GetFrequency
Prototype	`uint32_t LORAWAN_GetFrequency (uint8_t channelId)`
Summary	Returns the frequency of the given channel.
Description	This function returns the frequency on the requested “channelId”, entered in decimal form.
Preconditions	none
Parameters	• `channelId` – the channel requested (`uint8_t`)
Return	The frequency of the given channel (value returned is in Hz - `uint32_t`)

Example	`uint32_t freqUsed;`
	
	`freqUsed = LORAWAN_GetFrequency(4);`

4.3.2.54 LORAWAN_SetDataRange

Name	LORAWAN_SetDataRange
Prototype	`LorawanError_t LORAWAN_SetDataRange (uint8_t channelId, uint8_t dataRangeNew)`
Summary	Sets new Data Rate Range for the given channel.
Description	This function sets the operating data rate range, minimum to maximum, for the given “channelId”. Thus, the module can vary data rates between the minimum and maximum range. Refer to the LoRaWAN Specification V1.0 document for the actual values of the data rates and the corresponding spreading factors (SF).
Preconditions	none
Parameters	• `channelId` – the channel requested • `dataRangeNew` – the first four MSBs represent the maximum value and the last four LSB contain the minimum value.
Return	none
Example	Setting channel 13 Data Rate Range between 1 and 3: `MacSetDataRange(13, 0x31); // (0x31 -> 0011 0001)`

4.3.2.55 LORAWAN_GetDataRange

Name	`LORAWAN_GetDataRange`
Prototype	`uint8_t LORAWAN_GetDataRange (uint8_t channelId)`
Summary	Returns the Data Rate Range of a given channel.
Description	This function returns the operating data rate range, minimum to maximum, for the given "channelId".
Preconditions	none
Parameters	• `channelId` – the channel requested
Return	Returns the minimum and maximum Data Rate Range (`uint8_t`). The first four MSBs represent the maximum value and the last four bits, LSBs, contain the minimum value.

Example	`uint8_t dataRangeUsed;`
	
	`dataRangeUsed = LORAWAN_GetDataRange(3);`

4.3.2.56 LORAWAN_SetChannelIdStatus

Name	`LORAWAN_SetChannelIdStatus`
Prototype	`LorawanError_t LORAWAN_SetChannelIdStatus (uint8_t channelId, bool statusNew)`
Summary	Sets to a given channel a new status.
Description	This function sets the operation of the given "channelId".
Preconditions	none
Parameters	• `channelId` – a decimal number representing the channel number • `statusNew` – value representing the state, on/off (true/false)
Return	Returns LoRaWAN Error Type (`LorawanError_t`)
Example	`LORAWAN_SetChannelIdStatus(5, DISABLED);`

4.3.2.57 LORAWAN_GetChannelIdStatus

Name	`LORAWAN_GetChannelIdStatus`
Prototype	`bool LORAWAN_GetChannelIdStatus (uint8_t channelId)`
Summary	Returns the status of a given channel.
Description	This function returns the status of the given "channelId".
Preconditions	none
Parameters	• `channelId` – a decimal number representing the channel number
Return	Channel status – enabled/disabled (true/false)

Example	`bool channel3Status;`
	
	`channel3Status = LORAWAN_GetChannelIdStatus(3);`

4.3.2.58 LORAWAN_Pause

Name	LORAWAN_Pause
Prototype	`uint32_t LORAWAN_Pause (void)`
Summary	Pauses the LoRaWAN stack.
Description	This function pauses the LoRaWAN stack functionality to allow the transceiver (radio) configuration. By using “`mac pause`”, the radio commands can be generated between a LoRaWAN protocol uplink application, and the LoRaWAN protocol Receive windows. This function will reply within the time interval in milliseconds that the transceiver can be used without affecting the LoRaWAN functionality.
Preconditions	none
Parameters	none
Return	Returns the number in milliseconds representing how much it can be paused without affecting the functionality. Returns ‘`0`’ if it cannot be paused, the maximum value when in Idle mode.

Example	`uint32_t timeToPauseInMs;`
	
	`timeToPauseInMs = LORAWAN_Pause();`

4.3.2.59 LORAWAN_Resume

Name	LORAWAN_Resume
Prototype	`void LORAWAN_Resume (void)`
Summary	Resumes the LoRaWAN stack functionality.
Description	This function resumes the LoRaWAN stack functionality, in order to continue normal functionality after being paused.
Preconditions	none
Parameters	none
Return	none
Example	`LORAWAN_Resume();`

4.3.2.60 LORAWAN_LinkCheckConfigure

Name	`LORAWAN_LinkCheckConfigure`
Prototype	`void LORAWAN_LinkCheckConfigure (uint16_t period)`
Summary	Sets the time interval for the link check process.
Description	This function sets the time interval for the link check process to be triggered periodically. A <value> of '`0`' will disable the link check process. When the time interval expires, the next application packet that will be sent to the server will include a link check MAC command. Refer to the LoRaWAN Specification V1.0 document for more information on the link check configuration.
Preconditions	none
Parameters	• `period` – the new period value; it must be provided in seconds
Return	none
Example	`LORAWAN_LinkCheckConfigure(10);`

4.3.2.61 LORAWAN_ForceEnable

Name	`LORAWAN_ForceEnable`
Prototype	`void LORAWAN_ForceEnable (void)`
Summary	Disables the Silent Immediately state.
Description	The network can issue a certain command that would require the end device to go silent immediately. This mechanism disables any further communication of the module, isolating it effectively from the network. Using this function, after this network command has been received, the connectivity of the modules is restored, allowing it to send data.
Preconditions	none
Parameters	none
Return	none
Example	`LORAWAN_ForceEnable();`

4.3.2.62 LORAWAN_Reset (for EU433/EU868 ISM bands)

Name	`LORAWAN_Reset`
Prototype	`void LORAWAN_Reset (IsmBand_t ismBandNew)`
Summary	This function will automatically reset the software LoRaWAN stack and initialize it with the parameters for the selected ISM band.
Description	This API will set default values for most of the LoRaWAN parameters. Everything set prior to this command will lose its set value, being reinitialized to the default value, including setting the cryptographic keys to '`0`'.
Preconditions	none
Parameters	• `ismBandNew` – the new band (`IsmBand_t`).
Return	none
Example	`LORAWAN_Reset(ISM_EU868);`

4.3.2.63 LORAWAN_Reset (for NA915 ISM band)

Name	LORAWAN_Reset
Prototype	void LORAWAN_Reset (void)
Summary	This function will automatically reset the software LoRaWAN stack and initialize it with the parameters for the selected ISM band.
Description	This API will set default values for most of the LoRaWAN parameters. Everything set prior to this command will lose its set value, being reinitialized to the default value, including setting the cryptographic keys to '0'.
Preconditions	none
Parameters	none
Return	none
Example	LORAWAN_Reset();

4.3.2.64 LORAWAN_SetFrequency (only for EU433/EU868)

Name	LORAWAN_SetFrequency
Prototype	LorawanError_t LORAWAN_SetFrequency (uint8_t channelId, uint32_t frequencyNew)
Summary	This function sets the frequency on the requested "channelId" to a new value.
Description	This API will set default values for most of the LoRaWAN parameters. Everything set prior to this command will lose its set value, being reinitialized to the default value, including setting the cryptographic keys to '0'.
Preconditions	none
Parameters	• channelId – the given channel • frequencyNew – the new frequency value (the value must be provided in Hz).
Return	Returns LoRaWAN Error Type (LorawanError_t)

Example	uint8_t channel4 = 4;
	uint32_t freqCh4 = 868500000;
	
	LORAWAN_SetFrequency(channel4, freqCh4);

4.3.2.65 LORAWAN_SetDutyCycle (only for EU433/EU868)

Name	LORAWAN_SetDutyCycle
Prototype	LorawanError_t LORAWAN_SetDutyCycle (uint8_t channelId, uint16_t dutyCycleValue)
Summary	This function sets the duty cycle.
Description	This function sets the duty cycle on a given channel.
Preconditions	none
Parameters	• channelId – the given channel • dutyCycleValue – value of the duty cycle; dutyCycleValue = (100 / X) - 1, where X is the duty cycle in percentage. For more details, refer to the LoRaWAN Specification V0.1 document.
Return	Returns LoRaWAN Error Type (LorawanError_t)

Example	uint8_t channel5 = 5;
	uint16_t dutyCycle5 = 9;
	
	LORAWAN_SetDutyCycle(channel5, dutyCycle5);

4.3.2.66 LORAWAN_GetDutyCycle (only for EU433/EU868)

Name	LORAWAN_GetDutyCycle
Prototype	uint16_t LORAWAN_GetDutyCycle (uint8_t channelId)
Summary	This function returns the duty cycle for a given channel.
Description	This function returns the value of the duty cycle for a given channel. The returned value is calculated using the formula: dutyCycleValue = (100/X) – 1, where X is the duty cycle in percentage. For more details, refer to the LoRaWAN Specification V1.0 document.
Preconditions	none
Parameters	• channelId – the given channel.
Return	Returns the requested channel duty cycle (uint16_t)

Example	uint8_t channel6 = 6;
	uint16_t dutyCycleUsed;
	
	dutyCycleUsed = LORAWAN_GetDutyCycle(channel6);

4.3.2.67 LORAWAN_GetISMBand (only for EU433/EU868)

Name	LORAWAN_GetISMBand
Prototype	uint8_t LORAWAN_GetIsmBand(void)
Summary	Returns the configured ISM Band.
Description	This function returns the configured ISM Band.
Preconditions	none
Parameters	none
Return	Returns ISM Band Type (ISMBand_t)

Example	`uint8_t ISMBandUsed;`
	
	`ISMBandUsed = LORAWAN_GetISMBand();`

4.3.2.68 LORAWAN_Mainloop

Name	LORAWAN_Mainloop
Prototype	void LORAWAN_Mainloop (void)
Summary	LoRaWAN main loop function
Description	This function is used for running the system timers and checking the DIO pins. It must be called in the while(1) loop inside <main> function (once per loop).
Preconditions	none
Parameters	none
Return	none

Example	`while(1)`
	`{`
	`    LORAWAN_Mainloop();`
	`}`

4.3.2.69 LORAWAN_SetClass

Name	LORAWAN_SetClass
Prototype	void LORAWAN_SetClass (LoRaClass_t deviceClass)
Summary	Sets LoRaWAN device class
Description	This function sets LoRaWAN stack class to A or C.
Preconditions	none
Parameters	• deviceClass - the new class (LoRaClass_t)
Return	none
Example	LORAWAN_SetClass(CLASS_C)

4.3.2.70 LORAWAN_GetClass

Name	`LORAWAN_GetClass`
Prototype	`LoRaClass_t LORAWAN_GetClass (void);`
Summary	Returns LoRaWAN device class
Description	This function returns LoRaWAN stack class.
Preconditions	none
Parameters	none
Return	Returns LoRaWANClass Type (`LoRaClass_t`)
Example	`LoRaClass_t deviceClass;` `deviceClass = LOWRAWAN_GetClass();`

4.3.2.71 LORAWAN_SetMcast

Name	`LORAWAN_SetMcast`
Prototype	`LorawanError_t LORAWAN_SetMcast(bool status);`
Summary	Set the status of multicast
Description	This function enables or disables the multicast operation.
Preconditions	When enabling the multicast, its parameters (`mcastNetworkSessionKey`, `mcastApplicationSessionKey`, `mcastDeviceAddressNew`) must be set and the network must be joined.
Parameters	• `status` - true or false (`bool`)
Return	Returns LoRaWAN Error Type (`LorawanError_t`)
Example	`LORAWAN_SetMcast (true)`

4.3.2.72 LORAWAN_GetMcast

Name	`LORAWAN_GetMcast`
Prototype	`bool LORAWAN_GetMcast(void);`
Summary	Returns the status of multicast.
Description	This function returns the status of multicast.
Preconditions	none
Parameters	none
Return	status - `true/false(bool)`
Example	`bool status;` `status = LORAWAN_GetMcast();`

4.3.2.73 LORAWAN_SetMcastApplicationSessionKey

Name	LORAWAN_SetMcastApplicationSessionKey
Prototype	`void LORAWAN_SetMcastApplicationSessionKey (uint8_t *mcastApplicationSessionKeyNew);`
Summary	Sets the multicast application session key.
Description	This function sets the Multicast Application Session Key (`McastAppSKey`). The `McastAppSKey` is an application session key specific to a group of end devices, used to encrypt/decrypt the payload field of the application-specific multicast data messages, and also to calculate/verify an application-level MIC (Message Integrity Code) that may be included in the payload. The `McastAppSKey` is a 16-byte array.
Preconditions	Pointer must be allocated by caller
Parameters	• `mcastApplicationSessionKeyNew` – buffer where the value is stored.
Return	none

Example	`uint8_t mcastAppSKey[16] = {0x3C, 0x8F, 0x26, 0x27, 0x39,`
	`0xBF, 0xE3, 0xB7, 0xBC, 0x08, 0x26, 0x99, 0x1A, 0xD0,0x50, 0x4D};`
	`LORAWAN_SetApplicationSessionKey(mcastAppSKey);`

4.3.2.74 LORAWAN_GetMcastApplicationSessionKey

Name	LORAWAN_GetMcastApplicationSessionKey
Prototype	`void LORAWAN_GetMcastApplicationSessionKey (uint8_t *mcastApplicationSessionKeyNew);`
Summary	Gets the multicast application session key.
Description	This function gets the Multicast Application Session Key (`McastAppSKey`). The `McastAppSKey` is an application session key specific to a group of end devices, used to encrypt/decrypt the payload field of the application-specific multicast data messages, and also to calculate/verify an application-level MIC (Message Integrity Code) that may be included in the payload. The `McastAppSKey` is a 16-byte array.
Preconditions	Pointer must be allocated by caller
Parameters	• `mcastApplicationSessionKey` – buffer where the value will be stored.
Return	none
Example	`uint8_t mcastAppSKey[16];` `LORAWAN_GetApplicationSessionKey(mcastAppSKey);`

4.3.2.75 LORAWAN_SetMcastNetworkSessionKey

Name	`LORAWAN_SetMcastNetworkSessionKey`
Prototype	`void LORAWAN_SetMcastNetworkSessionKey (uint8_t *mcastNetworkSessionKeyNew);`
Summary	Sets the multicast network session key.
Description	This function sets the Multicast Network Session key (`McastNwkSKey`).
Preconditions	Pointer must be allocated by caller
Parameters	• `mcastNetworkSessionKeyNew` – buffer where the value is stored.
Return	none

Example	`uint8_t mcastNwkSKey[16] = {0x2B, 0x7E, 0x15, 0x16, 0x28,`
	`0xAE, 0xD2, 0xA6, 0xAB, 0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C};`
	`LORAWAN_SetMcastNetworkSessionKey(mcastNwkSKey);`

4.3.2.76 LORAWAN_GetMcastNetworkSessionKey

Name	`LORAWAN_GetMcastNetworkSessionKey`
Prototype	`void LORAWAN_GetMcastNetworkSessionKey (uint8_t *mcastNetworkSessionKeyNew);`
Summary	Gets the multicast network session key.
Description	This function gets the Multicast Network Session key (`McastNwkSKey`).
Preconditions	Pointer must be allocated by caller
Parameters	• `mcastNetworkSessionKey` – buffer where the value will be stored.
Return	none
Example	`uint8_t mcastNwkSKey[16];` `LORAWAN_GetMcastNetworkSessionKey(mcastNwkSKey);`

4.3.2.77 LORAWAN_SetMcastDeviceAddress

Name	`LORAWAN_SetMcastDeviceAddress`
Prototype	`void LORAWAN_SetMcastDeviceAddress (uint32_t mcastDeviceAddressNew);`
Summary	Sets a group of end-devices multicast address.
Description	This function sets the group of end-devices multicast address (`McastDevAddr`). The `McastDevAddr` is a 32-bit identifier of a group of end devices within the current network.
Preconditions	none
Parameters	• `mcastDeviceAddressNew` – the value of the new address to be set.
Return	none
Example	`uint32_t mcastDevAddr = 0x11223344;` `LORAWAN_SetMcastDeviceAddress (mcastDevAddr);`

4.3.2.78 LORAWAN_GetMcastDeviceAddress

Name	`LORAWAN_GetMcastDeviceAddress`
Prototype	`uint32_t LORAWAN_GetMcastDeviceAddress (void);`
Summary	Returns the group of end-devices multicast address.
Description	This function gets the group of end-devices multicast address (`McastDevAddr`). The `McastDevAddr` is a 32-bit identifier of a group of end devices within the current network.
Preconditions	none
Parameters	none
Return	multicast device address (`uint32_t value`)
Example	`uint32_t mcastDevAddr;` `mcastDevAddr = LORAWAN_GetMcastDeviceAddress();`

4.3.2.79 LORAWAN_SetMcastDownCounter

Name	`LORAWAN_SetMcastDownCounter`
Prototype	`void LORAWAN_SetMcastDownCounter(uint32_t newCnt);`
Summary	Sets the multicast downlink counter.
Description	This function sets the multicast downlink counter. It can be used for devices that enter an already existing multicast group.
Preconditions	none
Parameters	• `newCnt` – the new downlink counter value (`uint32_t`)
Return	none
Example	`uint32_t newCnt = 12345;` `LORAWAN_SetMcastDownCounter(newCnt);`

4.3.2.80 LORAWAN_GetMcastDownCounter

Name	`LORAWAN_GetMcastDownCounter`
Prototype	`uint32_t LORAWAN_GetMcastDownCounter();`
Summary	Returns the multicast downlink counter.
Description	This function returns the multicast downlink counter.
Preconditions	none
Parameters	none
Return	multicast downlink counter (`uint32_t`)
Example	`uint32_t newCnt;` `newCnt = LORAWAN_SetMcastDownCounter();`

4.3.2.81 LORAWAN_GetState

Name	`LORAWAN_GetState`
Prototype	`uint8_t LORAWAN_GetState(void)`
Summary	Function returns the LoRaWAN stack state.
Description	This function returns the state of LoRaWAN stack. The possible LoRaWAN Class A/Class C states are the following: • IDLE • TRANSMISSION_OCCURRING • BEFORE_RX1 • RX1_OPEN • BETWEEN_RX1_RX2 • RX2_OPEN • RETRANSMISSION_DELAY • ABP_DELAY • CLASS_C_RX2_1_OPEN • CLASS_C_RX2_2_OPEN
Preconditions	none
Parameters	none
Return	Returns state of LoRaWAN stack (`uint8_t`)
Example	`uint8_t stateOfLorawan;` `stateOfLorawan = LORAWAN_GetState(void);`

Chapter 5. Building a LoRaWAN-Based Application

5.1 OVERVIEW

The LoRaWAN stack files generated in **Section 3.3 "LoraWAN™ Stack Generation for the RN2XX3 Modules"** can be used to build a custom LoRaWAN-based application.

All the code that belongs to the custom application must be placed outside the LoRaWAN stack files (e.g., the code can be placed in `main.c` or any other file which is not a LoRaWAN-related file).

> **Note:** The code inside the LoRaWAN generated files must not be modified. Doing so might result in a non-functional LoRaWAN stack.

This chapter describes how to build a simple LoRaWAN-based application that continuously sends a text message (a string) to the server. Before being able to send data, the end device must be initialized, configured for activation and activated (must join the network). These four states are described in the following sub-chapters.

5.2 BASIC LoRaWAN CLASS A STACK OPERATION

5.2.1 Overview

There are four states which can be identified in regards to the basic LoRaWAN Class A Stack operations:

1. Initialization
2. Configuration for Activation
3. Activation
4. Communication

The basic operations are described in the following sub-chapters.

5.2.2 Initialization

In the Initialization operation, the stack must be initialized. This is done by calling the `LORAWAN_Init` function. This function should be called inside the main function, before the `while(1)` infinite loop, immediately after enabling peripheral and global interrupt.

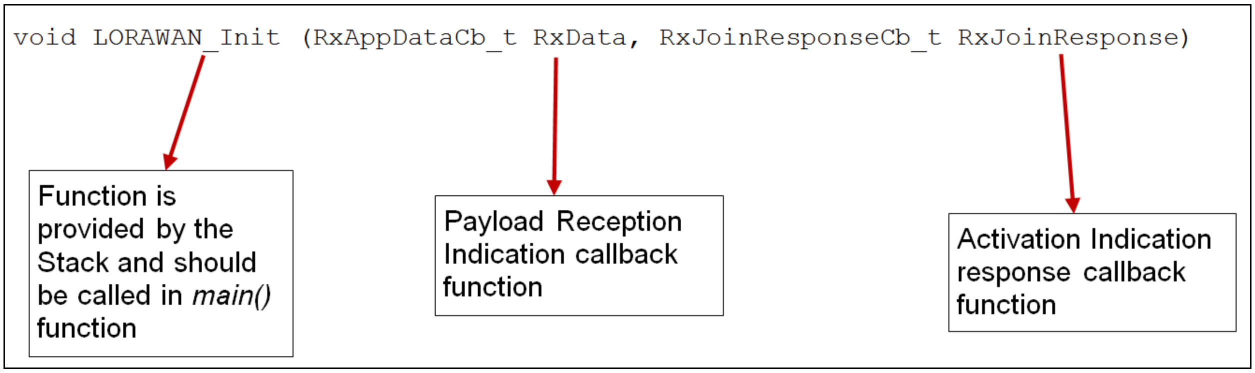
For more details on how to use this function, see **Section 5.6 "LoRaWAN-Based Custom Application Example"**.

The `LORAWAN_Init` function initializes the LoRaWAN stack and the radio transceiver. Two callback functions must be defined and used as parameters for `LORAWAN_Init` function.

The two callback functions that must be defined are used for the following:

- Payload Reception Indication
- Activation Indication Response

FIGURE 5-1: LoRaWAN_Init FUNCTION



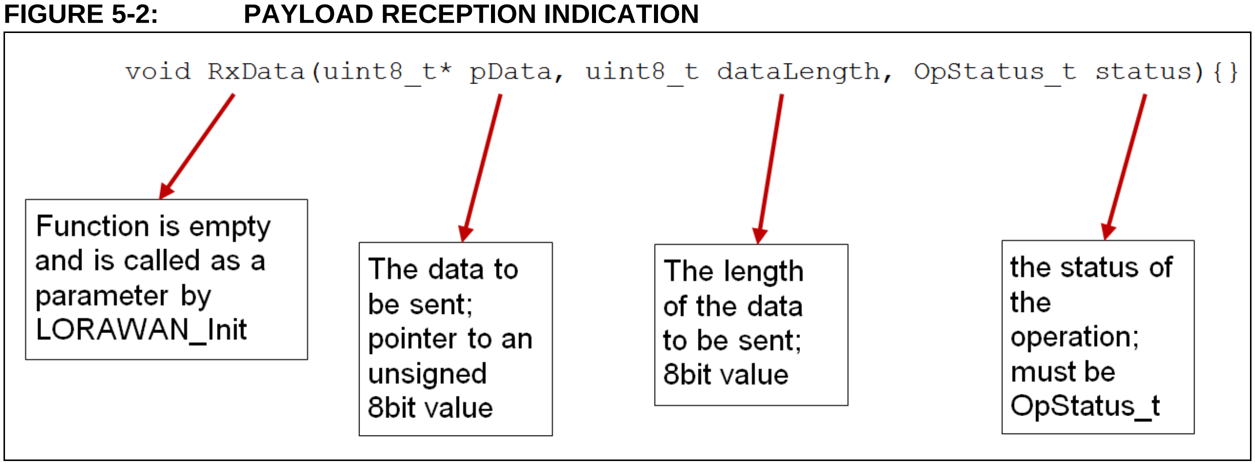
For details on the callback functions which are used by `LORAWAN_Join`, go to sections **Section 5.2.2.1 "Payload Reception Indication (Payload Reception Callback)"** and **Section 5.2.2.2 "Activation Indication Response"**.

5.2.2.1 PAYLOAD RECEPTION INDICATION (PAYLOAD RECEPTION CALLBACK)

The Payload Reception Callback can be an empty function, which has the following parameters:

- `uint8_t* pData` – a pointer to a 8-bit data; this is the actual data to be sent out
- `uint8_t dataLength` – the length of the data to be sent out
- `OpStatus_t status` – the operation status. This is actually the one indicating if the reception has occurred or not. The return of the status variable can be:
  - MAC_NOT_OK – LoRaWAN operation failed
  - MAC_OK – LoRaWAN operation successful
  - RADIO_NOT_OK – Radio operation failed
  - RADIO_OK – Radio operation successful
  - INVALID_BUFFER_LENGTH – a retransmission was tried and the buffer is too large because there was a Spreading Factor change.

FIGURE 5-2: PAYLOAD RECEPTION INDICATION

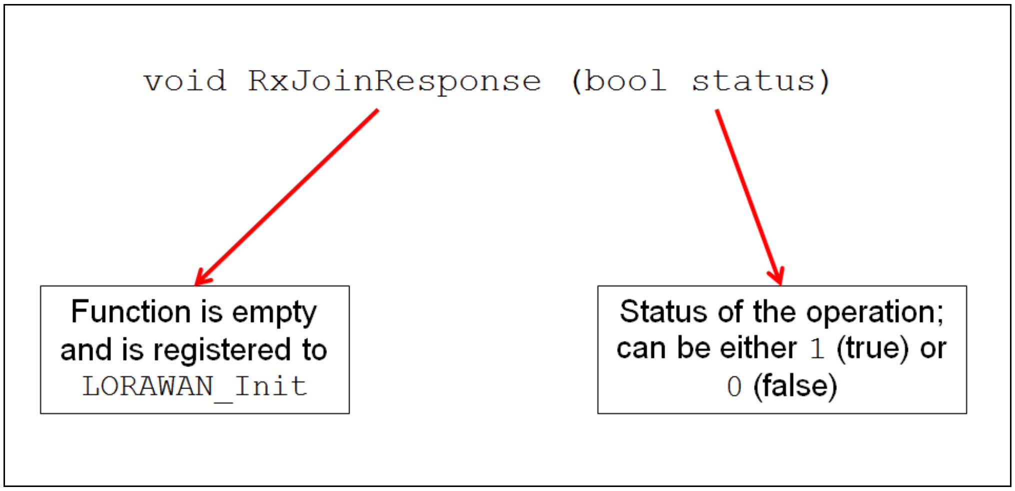


5.2.2.2 ACTIVATION INDICATION RESPONSE

The Activation Indication callback function can be an empty function which must have the following parameter:

- bool status – a binary value indicating the activation response from the network:
  - status = `1` (true) – the device has connected to the network
  - status = `0` (false) – the device has not connected to the network

FIGURE 5-3: ACTIVATION INDICATION RESPONSE



5.2.3 Configuration for activation

In the Configuration for Activation phase, the server keys must be passed to the LoRaWAN stack. The activation can be done in two ways:

- Activation-By-Personalization (ABP)
- Over-the-Air Activation (OTAA)

5.2.3.1 CONFIGURATION FOR ACTIVATION-BY-PERSONALIZATION

In order to connect the end device to the server using the Activation-By-Personalization procedure, there are two keys and one unique identifier which need to be passed to the LoRaWAN stack with the help of three APIs. The two keys and the unique identifier are:

- Network Session Key – The NwkSKey is a network session key specific to the end device. It is used by both the network server and the end device to calculate and verify the MIC (message integrity code) of all data messages to ensure data integrity. It is further used to encrypt and decrypt the payload field of a network management message.
- Application Session Key – The AppSKey is an application session key specific to the end device. It is used by both the application server and the end device to encrypt and decrypt the payload field of application-specific data messages.
- Device Address – The DevAddr consisting of 32 bits identifies the end device within the current network. Its format is shown in Figure 5-4.

FIGURE 5-4: DEVICE ADDRESS

Bit#	[31..25]	[24..0]
DevAddr bits	NwkID	NwkAddr

The Most Significant seven bits are used as network identifier (NwkID) to separate addresses of territorially overlapping networks of different network operators and to resolve roaming issues. The Least Significant 25 bits, the network address (NwkAddr) of the end device, can be arbitrarily assigned by the network manager.

Network Session Key Example

NWSKEY: `0x2B7E151628AED2A6ABF7158809CF4F3C`

The Network Session Key is chosen by the user and must be the same on the end device and the server. In order to provide this info to the end device, the NWSKEY must be split into bytes and defined as an array of 16 values.

FIGURE 5-5: NETWORK SESSION KEY SPLIT EXAMPLE

2B	7E	15	16	28	AE	D2	A6	AB	F7	15	88	09	CF	4F	3C
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	byte 14	byte 15	byte 16

The NWKSKEY can be passed to the LoRaWAN stack with the call of LORAWAN_SetNetworkSessionKey API:

```
LORAWAN_SetNetworkSessionKey(nwkSKey);
```

> **Note:** The parameter nwkSKey must be previously defined as an array of 16 elements (8-bit values).

Application Session Key Example

APPSKEY: `0x3C8F262739BFE3B7BC0826991AD0504D`

The Application Session Key is chosen by the user and must be the same on the end device and the server. In order to provide this info to the end device, the APPSKEY must be split into bytes and defined as an array of 16 values.

FIGURE 5-6: APPLICATION SESSION KEY SPLIT EXAMPLE

3C	8F	26	27	39	BF	E3	B7	BC	08	26	99	1A	D0	50	4D
byte 1	byte 2	byte 3	byte 4	byte 5	byte 6	byte 7	byte 8	byte 9	byte 10	byte 11	byte 12	byte 13	byte 14	byte 15	byte 16

The APPSKEY can be passed to the LoRaWAN stack with the call of LORAWAN_SetApplicationSessionKey API:

```
LORAWAN_SetApplicationSessionKey(appSKey);
```

> **Note:** The parameter AppSKey must be previously defined as an array of 16 elements (8-bit values).

Device Address example: DEVADDR: `0x11223344`

The Device Address is chosen by the user and it must be the same on the end device and the server. The DEVADDR can be passed to the LoRaWAN stack with the call of LORAWAN_SetDeviceAddress API:

```
LORAWAN_SetDeviceAddress(devAddr);
```

> **Note:** The parameter devAddr must be previously defined as a 32-bit value variable or constant.

5.2.3.2 CONFIGURATION FOR OVER-THE-AIR ACTIVATION

In order to connect the end device to the server using the Over-The-Air Activation procedure, three pieces of info need to be passed to the LoRaWAN stack with the help of three APIs:

- Application identifier (AppEUI) – The AppEUI is a global application ID in IEEE EUI64 address space that uniquely identifies the application provider (i.e., owner) of the end device. The AppEUI is stored in the end device before the activation procedure is executed.
- End-device identifier (DevEui) – The DevEUI is a global end-device ID in IEEE EUI64 address space that uniquely identifies the end device.
- Application key (AppKey) – The AppKey is an AES-128 application key specific to the end device assigned by the application owner to the end device and most likely derived from an application-specific root key exclusively known to and under the control of the application provider. Whenever an end device joins a network via Over-The-Air Activation, the AppKey is used to derive the session keys NwkSKey and AppSKey specific for that end device to encrypt and verify network communication and application data.

The AppEUI can be passed to the LoRaWAN stack by using LORAWAN_SetApplicationEui API:

```
LORAWAN_SetApplicationEui(applicationEuiNew);
```

Note:	Parameter applicationEuiNew must be previously defined as an 8-byte array.

The DevEUI can be passed to the LoRaWAN stack by using LORAWAN_SetDeviceEui API:

```
LORAWAN_SetDeviceEui (deviceEuiNew);
```

Note:	Parameter deviceEuiNew must be previously defined as an 8-byte array.

The AppKey can be passed to the LoRaWAN stack by using LORAWAN_SetApplicationKey API:

```
LORAWAN_SetApplicationKey (applicationKeyNew);
```

Note:	Parameter applicationKeyNew must be previously defined as an 16-byte array.

5.2.4 Activation

In the Activation phase, the end device sends a join request in ABP mode or OTAA mode to the server.

5.2.4.1 ACTIVATION-BY-PERSONALIZATION

Under certain circumstances, the end devices can be activated by personalization. Activation-By-Personalization directly ties an end device to a specific network bypassing the join request - join accept procedure.

Activating an end device by personalization means that the DevAddr and the two session keys NwkSKey and AppSKey are directly stored into the end device instead of the DevEUI, AppEUI and the AppKey. The end device is equipped with the required information for participating in a specific LoRa network when started.

Each device should have a unique set of NwkSKey and AppSKey. Compromising the keys of one device should not compromise the security of the communications of other devices. The process to build those keys should be such that the keys cannot be derived in any way from publicly available information (like the node address, for example).

In order to connect the end device to the network in Activation-By-Personalization mode, the following API and parameter must be used:

```
LORAWAN_Join(ABP);
```

This API must be called inside main function, before `while(1)` loop. For more details on building the software based on the LoRaWAN stack, see **Section 5.6 "LoRaWAN-Based Custom Application Example"**.

5.2.4.2 OVER-THE-AIR ACTIVATION

For Over-The-Air Activation, end devices must follow a join procedure prior to participating in data exchanges with the network server. An end device has to go through a new join procedure every time it has lost the session context information.

The join procedure requires the end device to be personalized with the following information before its starts the join procedure: a globally unique end-device identifier (DevEUI), the application identifier (AppEUI), and an AES-128 key (AppKey).

In order to connect the end device to the network in Over The Air Activation mode, the following API and parameter must be used:

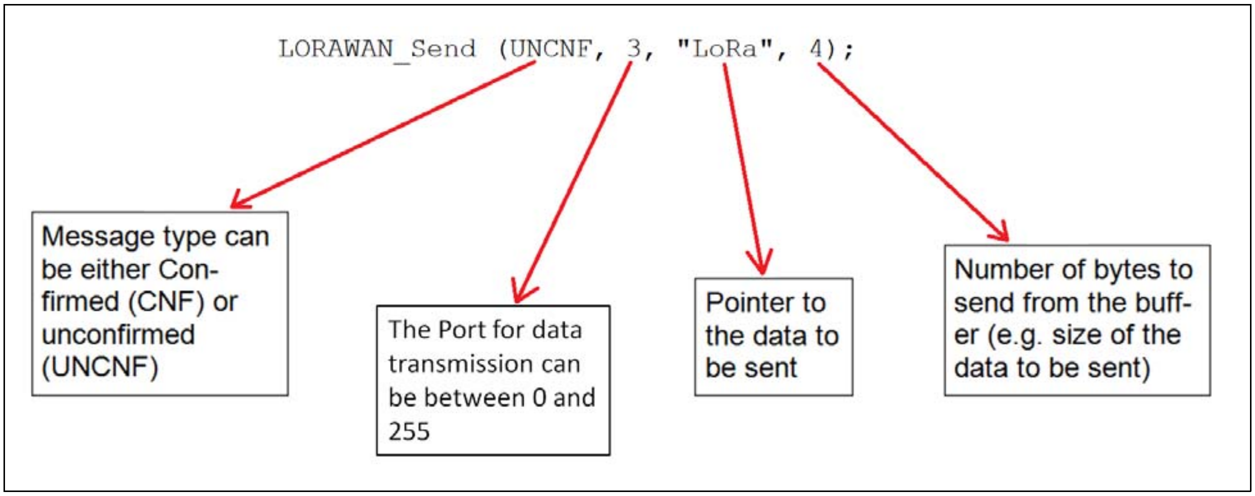
```
LORAWAN_Join(OTAA);
```

5.2.5 Communication

Once the device has been connected to the network, the user's custom application is able to send data to the server. Sending data can be done using LORAWAN_Send API:

```
LORAWAN_Send (UNCNF, 3, "LoRa", 4);
```


FIGURE 5-7: LORAWAN_Send API



5.3 BASIC LoRaWAN CLASS C STACK OPERATION

5.3.1 Overview

The end devices implementing the Class C option are used for applications that do not have to consider current consumption and therefore do not need to minimize reception time.

The Class C end device shall open RX2 windows as often as possible. The end device listens during RX2 window when it is neither sending nor receiving during RX1 window, according to Class A definition.

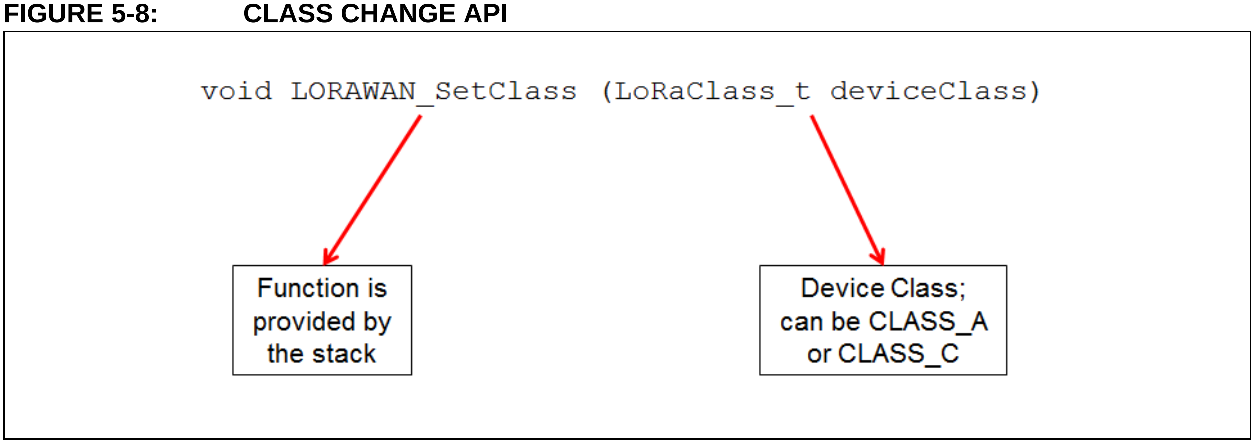
For more detailed information regarding Class C parameters please refer to LoRaWAN Specification document.

For a device that starts in Class C the initialization is the same as in the case of Class A. For details on the procedure, see **Section 5.2 "Basic LoRAWAN Class A Stack Operation"**.

5.3.2 Class Change

A device can start as either a Class A or Class C device. If there is a need to change the operation class during runtime, this is possible with the use of LORAWAN_SetClass API. For more details, see section **Section 4.3 "Application Programming Interfaces"**.

FIGURE 5-8: CLASS CHANGE API



5.3.3 Class C Continuous Receive

After the device has changed or started in Class C, to enter in continuous receive, a successful uplink to the server must be completed. For more information on the communication procedure go to **Section 5.2.5 "Communication"**.

5.3.4 Class C Multicast Messages

Messages can be "unicast" or "multicast". The multicast messages are sent to multiple end devices within a multicast group. All devices of a multicast group must share the same multicast address and associated encryption keys. Class C devices may receive multicast downlink frames. The multicast address and associated network session key and application session key must come from the application layer. The MCC LoRaWAN Library offers several APIs which enable the user to develop a LoRaWAN-based application, which also includes a Class C multicast option. For more details, see **Section 4.3 "Application Programming Interfaces"**.

5.4 BUILD A CUSTOM LORAWAN-BASED APPLICATION

5.4.1 Overview

All the APIs which were described in the previous sub-chapters must be called in a specific order inside the `main.c` file which is automatically generated by MCC.

5.4.2 Interrupts Enable

The Global Interrupts and Peripheral Interrupts must be enabled in `main.c` file. In order to enable them, the lines of code for the Global Interrupts and Peripheral Interrupts must be uncommented in main.c file, as shown in Example 5-1:

EXAMPLE 5-1: MAIN.C FILE

```
// Enable the Global Interrupts
   INTERRUPT_GlobalInterruptEnable();

// Enable the Peripheral Interrupts
   INTERRUPT_PeripheralInterruptEnable();
```

5.4.3 LoRaWAN Mainloop Function

The `LORAWAN_Mainloop()` function is running the LoRaWAN stack, so it must be called in `while(1)` infinite loop. All other user-defined functions of the custom application must be called after the `LORAWAN_Mainloop()` function call in `while(1)` infinite loop.

EXAMPLE 5-2: LoRaWAN™ MAINLOOP FUNCTION

```
.........................
while (1)
    {
       LORAWAN_Mainloop();
       ..................................................
       ..................................................
    }
.........................
```


5.5 LoRaWAN STACK ENCRYPTION

The LoRaWAN stack uses AES-128 encryption. The encryption scheme used is based on the generic algorithm described in IEEE 902.15.4/2006 Annex B [IEEE802154] using AES with a key length of 128 bits. For more details regarding the encryption/decryption of the LoRaWAN stack, refer to LoRaWAN Specification V1.0 document.

Microchip cannot distribute the AES.h and AES.c encryption files. Due to differences in licensing terms, the AES engine used by LoRaWAN stack needs to be added separately. The user is responsible of getting the encryption files. The two files must be manually replaced in the project, because the ones which are generated together with the stack only contain an error informing the user about the missing AES encryption.

The encryption files can be downloaded from the following location: http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=SW300052.

After downloading Data Encryption Libraries V2.6.zip and extracting AES.h and AES.c files, the user must make sure the files do not contain any keywords which are not supported by XC compilers (e.g., “rom” keyword must be deleted because it is not supported by XC8 compiler).

5.6 LORAWAN-BASED CUSTOM APPLICATION EXAMPLE

The LORAWAN_Mainloop() function is running the LoRaWAN stack, so it must be called in while(1) infinite loop. All other user-defined functions of the custom application must be called after the LORAWAN_Mainloop() function call in while(1) infinite loop.

The following code example is based on the MCC generated code using LoRaWAN Library (see **Chapter 3. "LoRaWAN™ Library Plug-in Running on RN2XX3 Modules"**) and can be run on the RN2XX3 modules.

EXAMPLE 5-3: MAIN.C FILE DEMO CODE


```
#include "mcc_generated_files/mcc.h"

uint8_t nwkSKey[16] = {0x2B, 0x7E, 0x15, 0x16, 0x28, 0xAE, 0xD2, 0xA6, 0xAB,
0xF7, 0x15, 0x88, 0x09, 0xCF, 0x4F, 0x3C};
uint8_t appSKey[16] = {0x3C, 0x8F, 0x26, 0x27, 0x39, 0xBF, 0xE3, 0xB7, 0xBC,
0x08, 0x26, 0x99, 0x1A, 0xD0, 0x50, 0x4D};
uint32_t devAddr = 0x1100000F;

void RxData(uint8_t* pData, uint8_t dataLength, OpStatus_t status)
{}

void RxJoinResponse(bool status)
{}

void main(void)
{
    // Initialize the device
    SYSTEM_Initialize();

    // If using interrupts in PIC18 High/Low Priority Mode you need to enable
    // the Global High and Low Interrupts
    // If using interrupts in PIC Mid-Range Compatibility Mode you need to
    // enable the Global and Peripheral Interrupts
    // Use the following macros to:

    // Enable high priority global interrupts
    //INTERRUPT_GlobalInterruptHighEnable();

    // Enable low priority global interrupts.
    //INTERRUPT_GlobalInterruptLowEnable();

    // Disable high priority global interrupts
    //INTERRUPT_GlobalInterruptHighDisable();
```


EXAMPLE 5-4: MAIN.C FILE DEMO CODE (CONTINUED)

```
    // Disable low priority global interrupts.
    //INTERRUPT_GlobalInterruptLowDisable();

    // Enable the Global Interrupts
    INTERRUPT_GlobalInterruptEnable();

    // Enable the Peripheral Interrupts
    INTERRUPT_PeripheralInterruptEnable();

    // Disable the Global Interrupts
    //INTERRUPT_GlobalInterruptDisable();

    // Disable the Peripheral Interrupts
    //INTERRUPT_PeripheralInterruptDisable();

    LORAWAN_Init(RxData, RxJoinResponse);

    LORAWAN_SetNetworkSessionKey(nwkSKey);
    LORAWAN_SetApplicationSessionKey(appSKey);
    LORAWAN_SetDeviceAddress(devAddr);

    LORAWAN_Join(ABP);

    while (1)
    {
        // Add your application code
        LORAWAN_Mainloop();

        // All other function calls of the user-defined
        // application must be made here
        LORAWAN_Send(UNCNF, 2, "LoRa", 4);
    }
}
```

NOTES:

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://www.microchip.com/support
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Austin, TX
Tel: 512-257-3370

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Novi, MI
Tel: 248-848-4000

Houston, TX
Tel: 281-894-5983

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453
Tel: 317-536-2380

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608
Tel: 951-273-7800

Raleigh, NC
Tel: 919-844-7510

New York, NY
Tel: 631-435-6000

San Jose, CA
Tel: 408-735-9110
Tel: 408-436-4270

Canada - Toronto
Tel: 905-695-1980
Fax: 905-695-2078

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon

Hong Kong
Tel: 852-2943-5100
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Dongguan
Tel: 86-769-8702-9880

China - Guangzhou
Tel: 86-20-8755-8029

China - Hangzhou
Tel: 86-571-8792-8115
Fax: 86-571-8792-8116

China - Hong Kong SAR
Tel: 852-2943-5100
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-3326-8000
Fax: 86-21-3326-8021

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8864-2200
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

ASIA/PACIFIC

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-3019-1500

Japan - Osaka
Tel: 81-6-6152-7160
Fax: 81-6-6152-9310

Japan - Tokyo
Tel: 81-3-6880- 3770
Fax: 81-3-6880-3771

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-5778-366
Fax: 886-3-5770-955

Taiwan - Kaohsiung
Tel: 886-7-213-7830

Taiwan - Taipei
Tel: 886-2-2508-8600
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

Finland - Espoo
Tel: 358-9-4520-820

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

France - Saint Cloud
Tel: 33-1-30-60-70-00

Germany - Garching
Tel: 49-8931-9700

Germany - Haan
Tel: 49-2129-3766400

Germany - Heilbronn
Tel: 49-7131-67-3636

Germany - Karlsruhe
Tel: 49-721-625370

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Germany - Rosenheim
Tel: 49-8031-354-560

Israel - Ra'anana
Tel: 972-9-744-7705

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Italy - Padova
Tel: 39-049-7625286

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Norway - Trondheim
Tel: 47-7289-7561

Poland - Warsaw
Tel: 48-22-3325737

Romania - Bucharest
Tel: 40-21-407-87-50

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

Sweden - Gothenberg
Tel: 46-31-704-60-40

Sweden - Stockholm
Tel: 46-8-5090-4654

UK - Wokingham
Tel: 44-118-921-5800
Fax: 44-118-921-5820