MPLAB Harmony Graphics Composer User's Guide

MPLAB Harmony Integrated Software Framework

MPLAB Harmony Graphics Composer User's Guide

This section provides user information about using the MPLAB Harmony Graphics Composer (MHGC).

Introduction

This user's guide provides information on the MPLAB Harmony Graphics Composer (MHGC), also referred to as the graphics composer, which is included in your installation of MPLAB Harmony. MHGC is tightly coupled with the Aria User Interface Library to facilitate rapid prototyping and optimization of the application's graphical user interface (GUI).

Description

The MPLAB Harmony Graphics Composer (MHGC), also referred to as the graphics composer, is a graphics user interface design tool that is integrated as part of the MPLAB Harmony Configurator (MHC). MHGC is tightly coupled with the Aria User Interface Library to facilitate rapid prototyping and optimization of the application's graphical user interface (GUI). The tool provides a "What you see is what you get" (WSYWIG) environment for users to design the graphics user interface for their application. Refer to *Volume V: MPLAB Harmony Framework Reference > Graphics Library Help > Aria User Interface Library* for more information.

The MPLAB Harmony Graphics Composer (MHGC) Tool Suite and the Aria User Interface Library provide the following benefits to developers:

- Enhanced User Experience – Libraries and tools are easy to learn and use.
- Intuitive MHGC Window Tool – Flexible window docking/undocking. Undo/Redo and Copy/Paste support. Tree-based design model. Display design canvas control including zooming.
- Tight Integration Experience – Graphics design & code generator tools are tightly integrated, providing rapid prototyping and optimization of look and feel
- Powerful User Interface (UI) Library – Provides graphics objects and touch support
- Multi-Layer UI design – Supported in the MHGC tool and Aria Library
- Complete Code Generation – Can generate code for library initialization, library management, touch integration, color schemes and event handling with a single click
- Supports Performance and Resource Optimization – Draw order, background caching, and advanced color mode support improve performance
- Resource optimization – Measures Flash memory usage and can direct resources to external memory if needed. Global 8-bit color look-up table (LUT) supports reduced memory footprint. Heap Estimator tool, which helps to manage the SRAM memory footprint.
- Text localization – Easily integrate international language characters into a design and seamlessly change between defined languages at run-time
- Easy to Use Asset Management – Tools provide intuitive management of all graphics assets (fonts, images, text strings)
- Image Optimization – Supports cropping, resizing, and color mode tuning of images
- Expanded Color Mode Support – The graphics stack can manage frame buffers using 8-bit to 32-bit color
- Powerful Asset Converter – Inputs several image formats, auto converts from input format to several popular internal asset formats, performs auto palette generation for image compression, supports run-length encoding. Supports automatic font character inclusion & rasterization.
- Event Management – Wizard-based event configuration. Tight coupling to enable touch user events and external logical events to change the graphics state machine and graphics properties.
- Abstract Hardware Support – Graphics controllers and accelerators can be added or removed without any change to the application

Glossary of Terms

Throughout this user's guide the following terms are used:

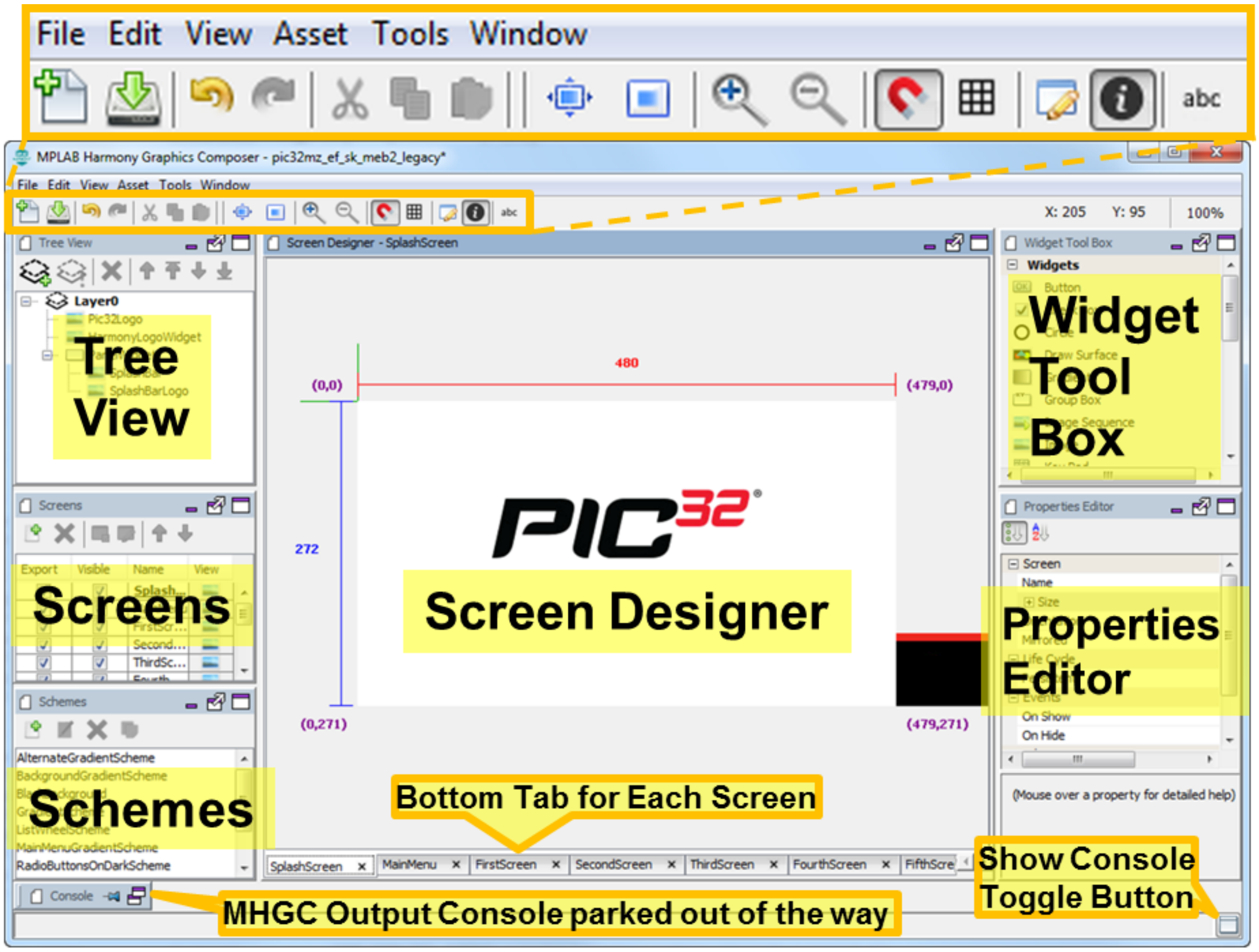
Acronym or Term	Description
Action	A specific task to perform when an event occurs.
Asset	An image, font, or binary data blob that is used by a user interface.
Event	A notification that a specific occurrence has taken place.
Resolution	The size of the target device screen in pixels.
Screen	A discreet presentation of organized objects.
Tool	An interface used to create objects.
UI	Abbreviation for User Interface.
Widget	A graphical object that resides on the user interface screen.

Graphics Composer Window User Interface

This section describes the layout of the different windows and tool panels available through MHGC.

Description

MHGC is launched from the MHC toolbar Launch Utility menu. Launching the Graphics Composer creates a new screen. Shown below is the MHGC screen for the Aria Showcase demonstration. (If you don't see this screen layout, reset the screen by selecting *Window > Reset Dock Areas* from the window's menus.)


File Edit View Asset Tools Window
MPLAB Harmony Graphics Composer - pic32mz_ef_sk_meb2_legacy*
X: 205 Y: 95 100%
Tree View
Screen Designer - SplashScreen
Widget Tool Box
Layer0
Tree View
Screens
Screens
Schemes
Schemes
Screen Designer
Widget Tool Box
Properties Editor
Properties Editor
Bottom Tab for Each Screen
Show Console Toggle Button
MHGC Output Console parked out of the way
SplashScreen MainMenu FirstScreen SecondScreen ThirdScreen FourthScreen FifthScre
Console
(Mouse over a property for detailed help)
(0,0) 480 (479,0) 272 (0,271) (479,271)

Panels

By default, there are five active panels and one minimize panel on this screen:

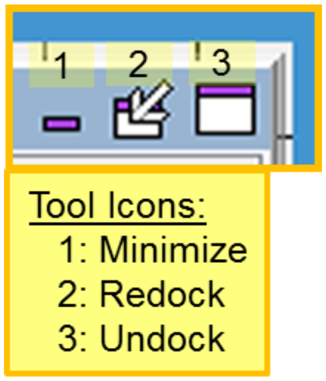
- Screen Designer – Shows the screen design for the selected screen. Tabs on the bottom of the Screen Designer panel show the available screens.
- Tree View – Shows the layer and widget hierarchy for the current screen.
- Screens – Manages screens in the application.
- Schemes – Manages coloring schemes in the application.

Note: In v2.03b of MPLAB Harmony, a third tab named Options, along with Screens and Schemes was available. These properties are now located within the *File > Settings* menu.

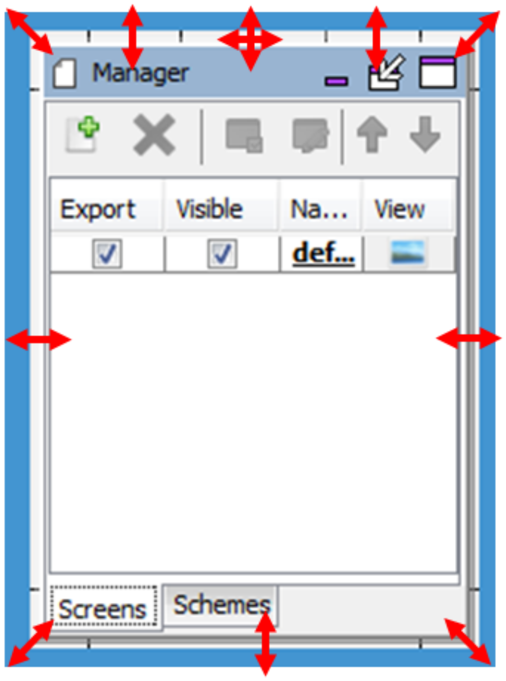
- Widget Tool Box – Available graphics widgets are shown on this panel. Widgets are added to the screen by selecting an icon and dragging or clicking. Widget properties are discussed in the Widget Properties section below.
- Properties Editor – All properties for the currently selected object are shown in this panel.
- The MHGC Output console is parked at the bottom of the Screen Designer window. This console panel can be used to debug problems when the Graphics Composer boots up or during its operation.

Each of the panels has a window tool icon at the upper right corner. Minimizing a panel parks it on the screen just like the Output Console. Undocking the panel creates a new, free floating window. Redocking returns a previously undocked window to its original location on the Screen

Designer window.

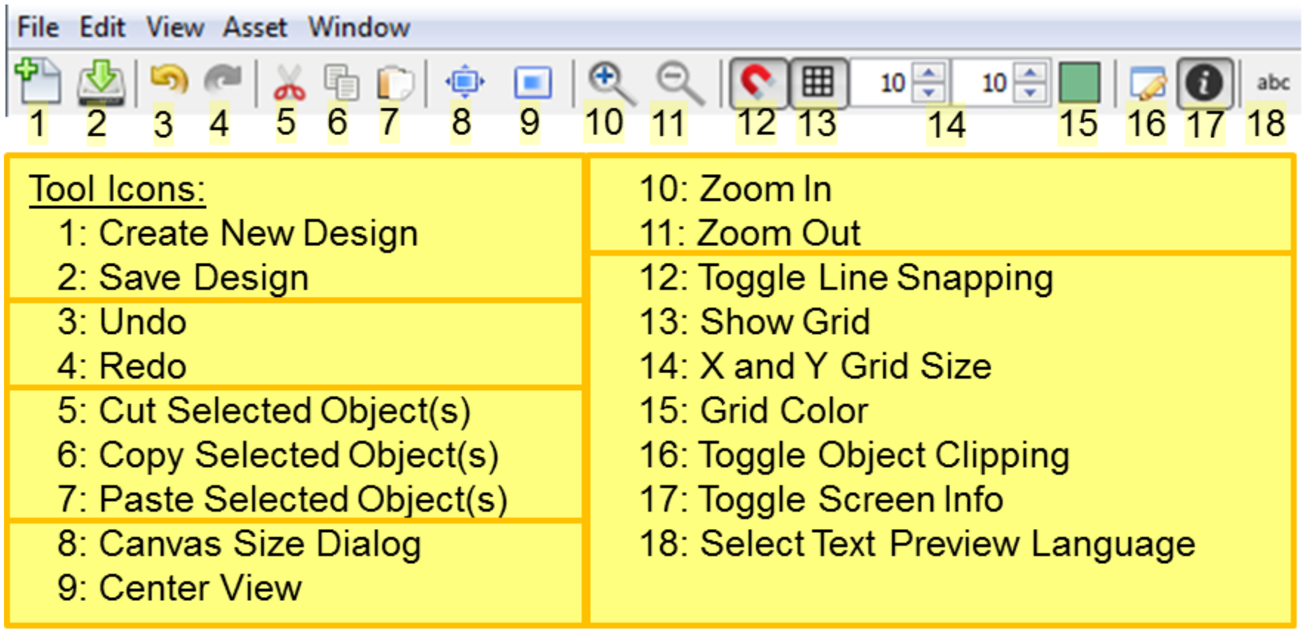


When a panel is undocked, its edges become active and support moving or manipulating the panel as an independent window.



Tool Bar

There are 18 tool bar icons on the Screen Designer Window, as described in the following figure.



Tool Icons:

1: Create New Design
2: Save Design
3: Undo
4: Redo
5: Cut Selected Object(s)
6: Copy Selected Object(s)
7: Paste Selected Object(s)
8: Canvas Size Dialog
9: Center View
10: Zoom In
11: Zoom Out
12: Toggle Line Snapping
13: Show Grid
14: X and Y Grid Size
15: Grid Color
16: Toggle Object Clipping
17: Toggle Screen Info
18: Select Text Preview Language

Create New Design brings up a New Project Wizard dialog that allows you to select anew the screen size, color mode, memory size, and project type. This will erase the currently displayed design.

Save Design saves the current graphics design.

Note: The target configuration's `configuration.xml` will not be updated to reflect these changes in the graphics design until one of the following events happens:

1. The application is regenerated in MHC,
2. The target configurations are changed in the MPLAB X IDE,
3. MPLAB X IDE is exited.

In items 2 and 3 you will be prompted to save the new configuration.

Undo and **Redo** manipulate changes in the screen design into internal MHC memory.

Cut/Copy/Paste support the manipulation of graphics objects (widgets).

Canvas Size Dialog brings up a dialog window allowing changes in the pixel width and height of the Screen Designer panel. (Note: Dimensions smaller than the display's dimensions are ignored).

Center View centers the panel's view of the screen.

Zoom In and Zoom Out allow you to change the scale of the Screen Designer's display of the current window. Currently this only supports coarse zooming (powers of two zooms in and out).

Toggle Line Snapping enables/disables line snapping when moving objects (widgets).

Show Grid turns the Screen Designer pixel grid on/off.

X and Y Grid Size adjust the pixel grid.

Grid Color selects the pixel grid color.

Toggle Object Clipping turns object clipping on/off.

Toggle Screen Info turns the display of screen information (X and Y axes) on/off.

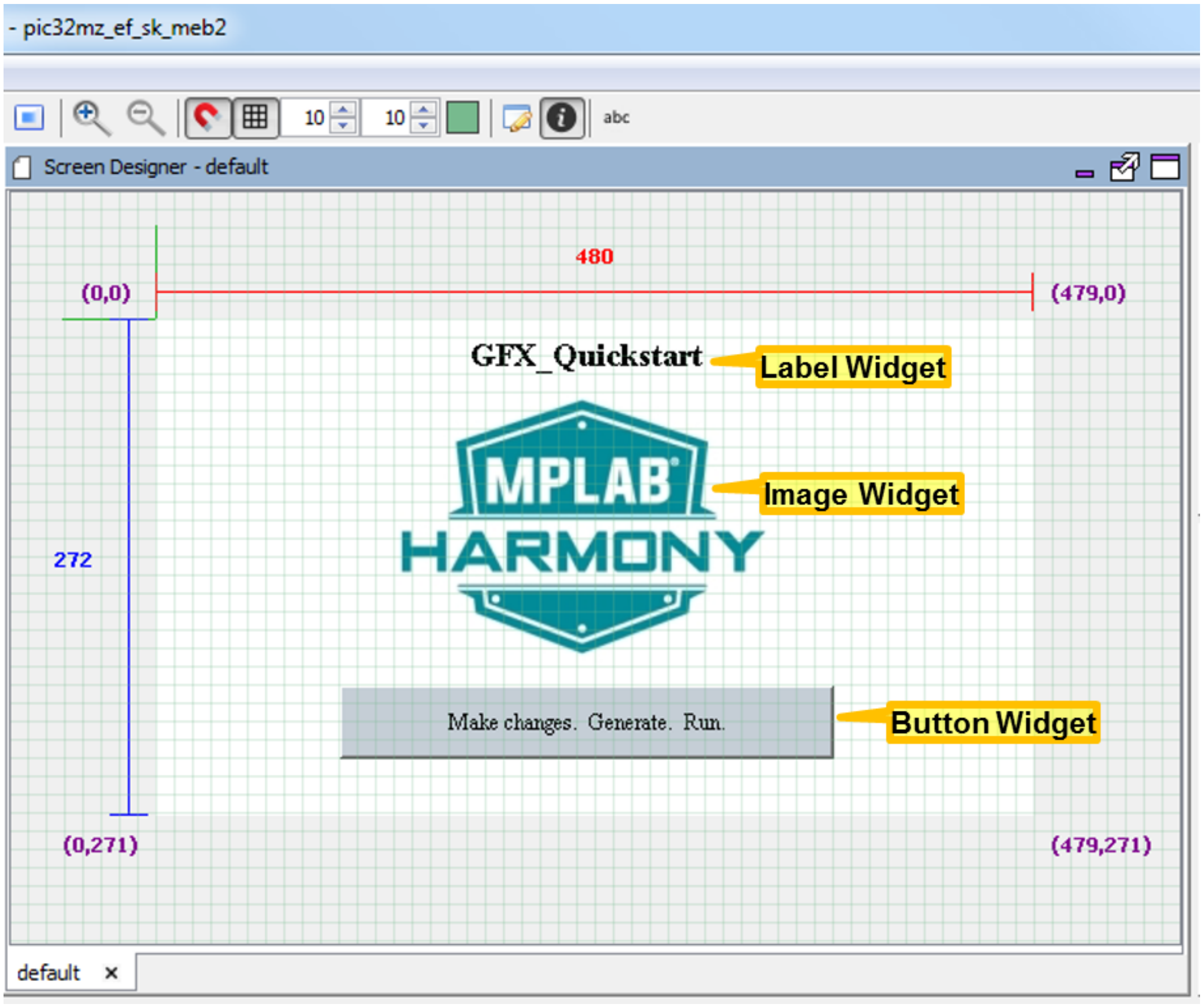
Select Text Preview Language changes the language used on all text strings shown, when the application supports more than one language.

Screen Designer Window

Most of the work of the MPLAB Harmony Graphics Composer is done using the Screen Designer. This section covers the basics of how a graphical user interface is designed using the screen designer.

Description

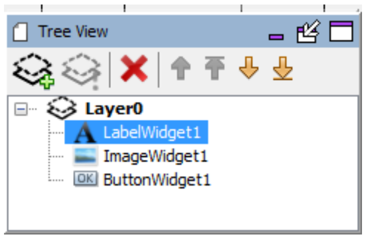
The following figure shows the Screen Designer window for the Aria Quickstart demonstration, with the pic32mz_ef_sk_meb2 configuration selected. (Load whatever configuration belongs to your board and follow along.)



The pixel dimensions of the display (480x272) are determined by the MHC Display Manager. Other configuration in Aria Quickstart can have different size displays (such as: 220x176, 320x24, or 800x480).

This demonstration has three widgets: a label containing the title string at the top, an image of the MPLAB Harmony logo in the middle, and a button containing the text string “Make changes. Generate. Run.” at the bottom. The label widget’s text string was first created using the String Assets window before it was assigned to the label widget. The image assigned to the image widget was first imported using the Image Assets. The string embedded in the button widget was also created using the String Assets window before it was assigned to the button widget.

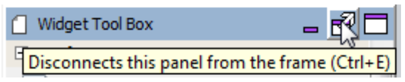
The Tree View panel organizes the display’s widgets into groups using layers. Every display has at least one layer and complex designs can have many more. Within the tree view, the order of layers and the order of widgets within a layer determine the draw order. Draw order goes from top to bottom. Top-most layers and widgets are drawn first and bottom-most are drawn last. Controlling draw order is one of the ways to improve graphics performance by minimizing redrawing.



Since the location of every widget within a layer is relative to the layer, you can move a layer’s worth of widgets by simply moving the layer. Layers also provide inheritance of certain properties from the layer to all the layer’s widgets.

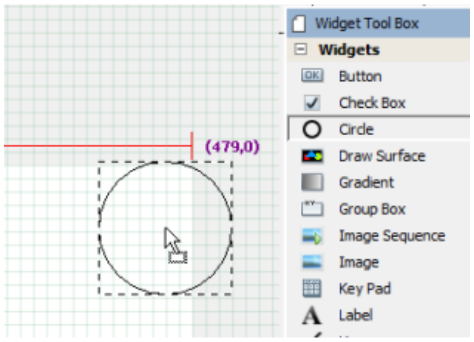
Exploring the Screen Designer Window

We can add another widget to this screen by launching the Widget Tool Box panel into a separate window.

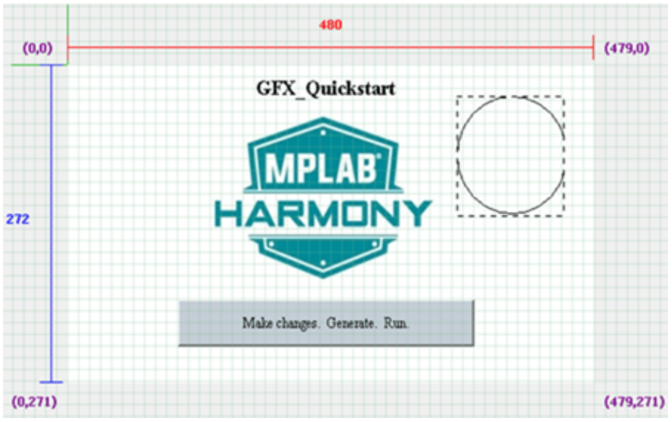


Next, drag a circle from the tool box onto the display. Find a place on the display for this new widget.

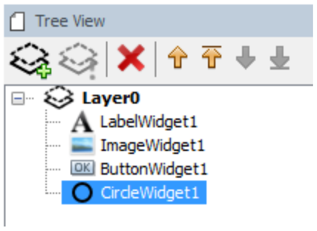
Besides dragging widgets onto the display, you can click on a widget in the Widget Tool Box, converting the cursor into that widget, and then click on the screen to drop the widget in place.



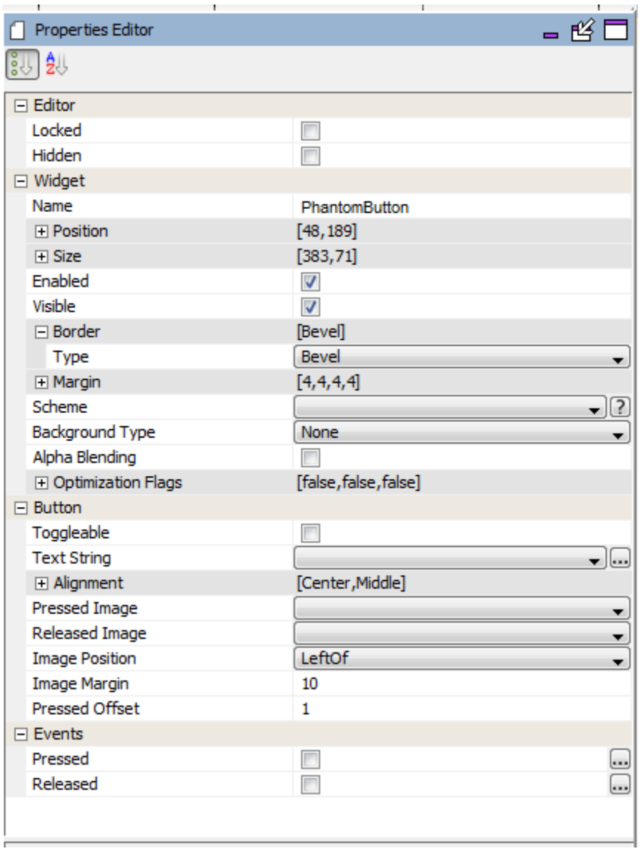
Your display should now look appear like the following figure.



Note how the Tree View panel now shows the widget you just added.

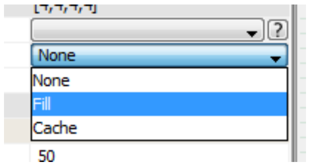


Launch the Properties Editor for the circle.


Properties Editor

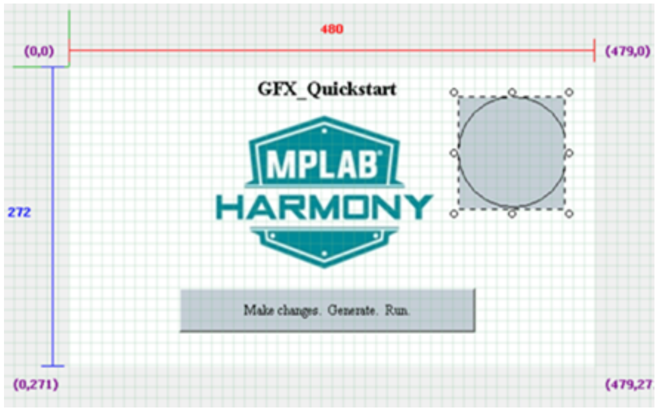
Editor	
Locked	☐
Hidden	☐
Widget	
Name	PhantomButton
Position	[48,189]
Size	[383,71]
Enabled	✓
Visible	✓
Border	[Bevel]
Type	Bevel
Margin	[4,4,4,4]
Scheme	
Background Type	None
Alpha Blending	☐
Optimization Flags	[false,false,false]
Button	
Toggleable	☐
Text String	
Alignment	[Center,Middle]
Pressed Image	
Released Image	
Image Position	LeftOf
Image Margin	10
Pressed Offset	1
Events	
Pressed	☐
Released	☐

Next, change the fill property on the circle from “None” to “Fill”.

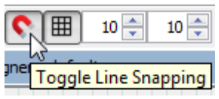


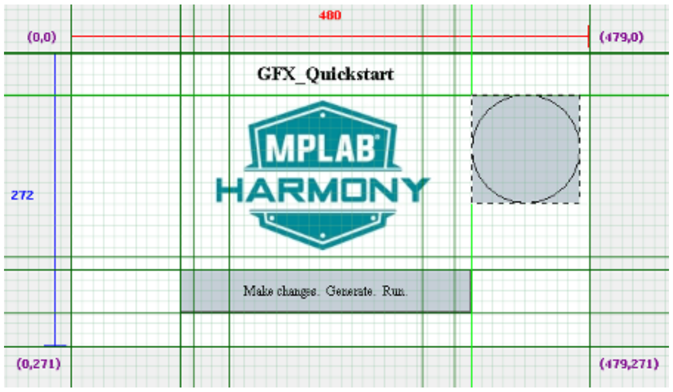
Note: If the properties in the Properties Editor shown are not for CircleWidget1, click on the circle widget to change the focus of the Properties Window.

When done, the screen should now appear, as follows.



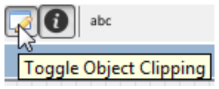
Turn on Line Snapping, which enables drawing guides to assist in aligning widgets on the display.

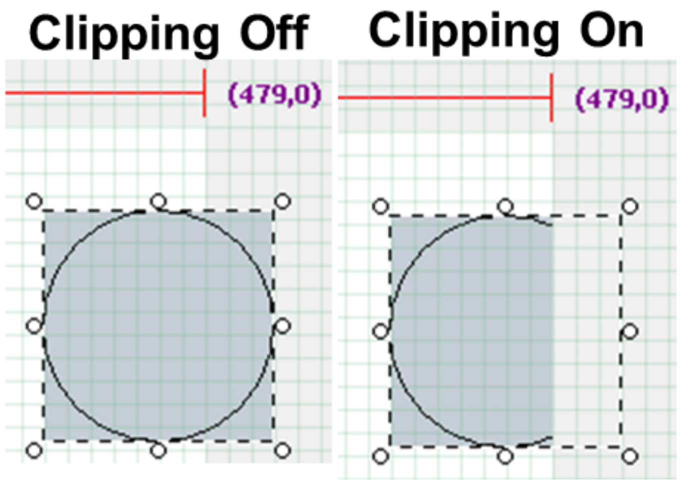




Next, turn on Object Clipping, which allows you to see how widgets are clipped by the boundaries of the layer that contains them.

Note: Clipping applies to layers, which can be smaller than the display.





To delete a widget, select the widget and press Delete on your keyboard or use the delete icon ( ) on the Tree View panel.

For more hands-on exploration of graphics using the Aria Quickstart demonstration, see *Volume 1: Getting Started With MPLAB Harmony > Quick Start Guides > Graphics and Touch Quick Start Guides > Adding an Event to the Aria Quickstart Demonstration*.

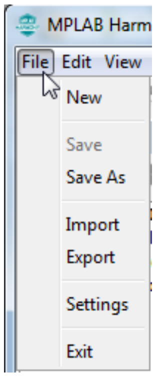
The steps to create a new MPLAB Harmony project with touch input on a PIC32MZ EF Starter Kit with the Multimedia Expansion Board (MEB) II display can be found in *Volume 1: Getting Started With MPLAB Harmony > Quick Start Guides > Graphics and Touch Quick Start Guides > Creating New Graphics Applications*.

Menus

This section provides information on the menus for the MPLAB Harmony Graphics Composer screen.

Description

File Menu



New – Same as the Create New Design tool icon.

Save – Same as the Save Design tool icon.

Save As – Supports exporting the design under a new name. By default, the name is `composer_export.xml`. See Importing and Exporting Graphics Data for more information.

Import - Reads in (imports) a previously exported design or a `./framework/src/system_config/{board_config}/configuration.xml` file that contains the graphics design to be imported. See Importing and Exporting Graphics Data for more information.

Export – Same as Save As. See Importing and Exporting Graphics Data for more information.

Settings – Brings up Project and User Settings dialog, including:

- Project Color Mode - How colors are managed
- Using a Global Palette
- Show Welcome Dialog
- Pre-emption Level – Allows for sharing of the device’s cycles with other parts of the application
- Hardware Acceleration – Is graphics hardware accelerator enabled in software?

Exit – Closes the MHGC window and exits

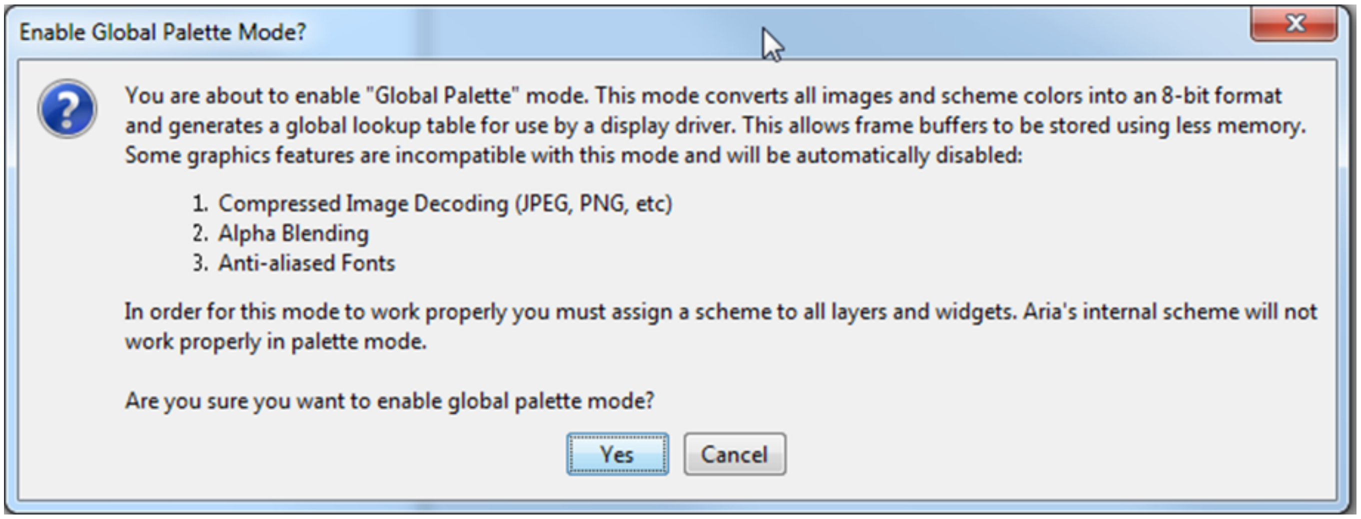
The choices for *Project and User Settings* > *Project Color Mode* are:

- GS_8 - 8-bit gray scale
- RGB_332 - Red/Green/Blue, 3 bits Red/Green, 2 bits Blue
- RGB_565 - Red/Green/Blue, 5 bits Red, 6 bits Green, 5 bits Blue
- RGBA_5551 - Red/Green/Blue/Alpha, 5 bits Red/ Green/Blue, 1 bit for Alpha Blending
- RGB_888 - Red/Green/Blue, 8 bits Red/Green/Blue
- RGBA_8888 - Red/Green/Blue/Alpha, 8 bits Red/Green/Blue/Alpha Blending
- ARGB_8888 - Alpha/Red/Green/Blue, 8 bits Alpha Blending/Red/Green/Blue

Ensure that the Project Color Mode chosen is compatible with the display hardware you are using; otherwise, the colors shown on the display will not match those shown on the Graphics Composer Screen Designer.

Using a Global Palette enables frame buffer compression for applications using the Low-Cost Controllerless (LCC) Graphics Controller or Graphics LCD (GLCD) Controller. If the global palette is enabled, you will have to change the MHC configuration of the Graphics Controller to match. For the LCC controller, enable "Palette Mode". For the GLCD controller, change the *Driver Settings* > *Frame Buffer Color Mode* to "LUT8".

If **Using a Global Palette** is enabled, the following warning appears.



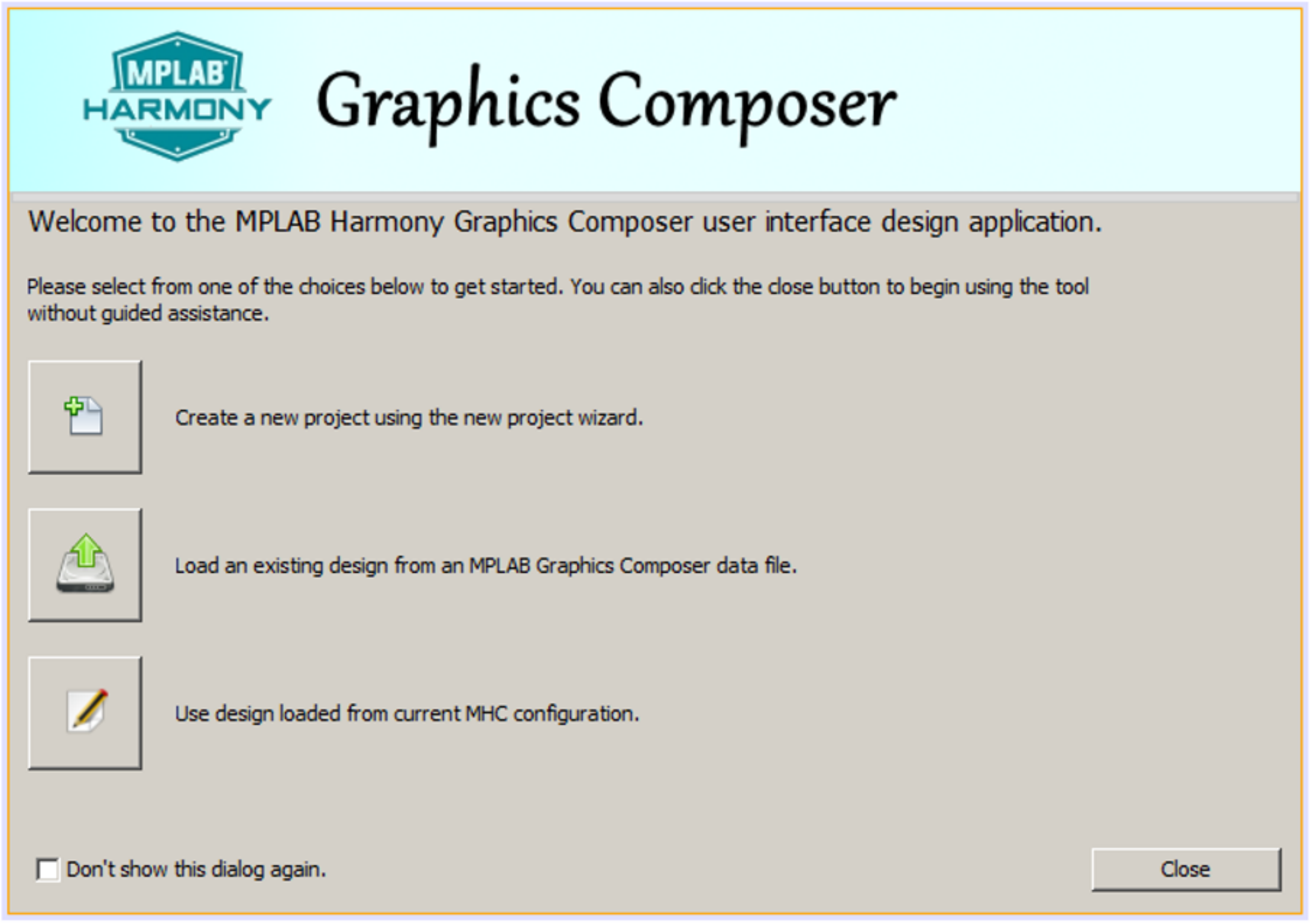
You are about to enable "Global Palette" mode. This mode converts all images and scheme colors into an 8-bit format and generates a global lookup table for use by a display driver. This allows frame buffers to be stored using less memory. Some graphics features are incompatible with this mode and will be automatically disabled:

1. Compressed Image Decoding (JPEG, PNG, etc)
2. Alpha Blending
3. Anti-aliased Fonts

In order for this mode to work properly you must assign a scheme to all layers and widgets. Aria's internal scheme will not work properly in palette mode.

Are you sure you want to enable global palette mode?

If **Show Welcome Dialog** is enabled, the following welcome screen appears when launching MHGC.



Welcome to the MPLAB Harmony Graphics Composer user interface design application.

Please select from one of the choices below to get started. You can also click the close button to begin using the tool without guided assistance.

Create a new project using the new project wizard.

Load an existing design from an MPLAB Graphics Composer data file.

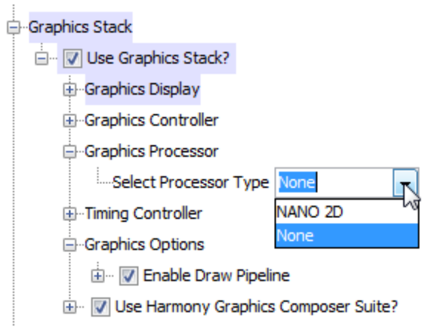
Use design loaded from current MHC configuration.

Note: If you are not creating a new project you can ignore this window.

When the **Preemption Level** is set to zero, all dirty graphics objects are refreshed before the graphics process relinquishes control of the device. (Dirty means needing a redraw.) With the level set to two, graphics provides maximum sharing with the rest of the application, at the cost of slower display refreshes. A level of one provides an intermediate level of sharing.

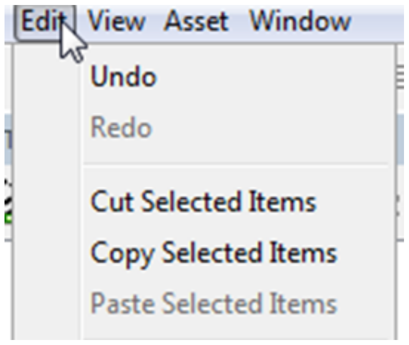
The **Hardware Acceleration** check box determines whether graphics uses the device's built-in graphics hardware accelerator in software.

Note: You must also specify the graphics hardware accelerator in the MPLAB Harmony Framework Configuration within the MHC Options tab. If the host device lacks a graphics processor, you will see a warning message when you try to select a processor that does not exist on your device.



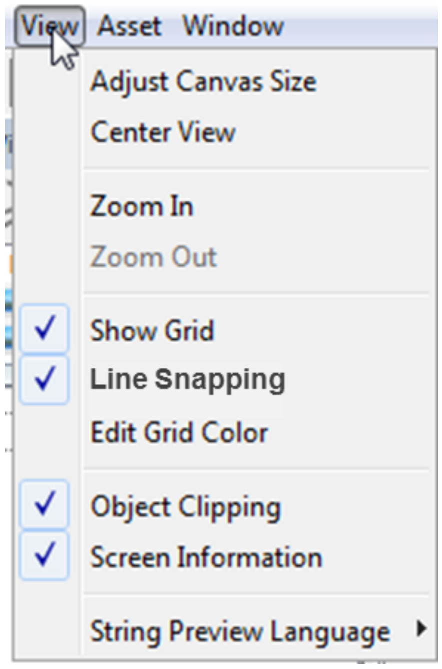
Edit Menu

This menu implements the same functions as the first seven tool icons.



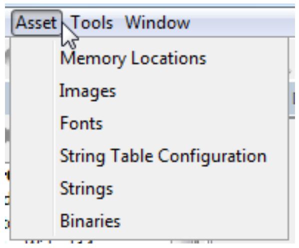
View Menu

This implements the same functions as the remaining tool icons.



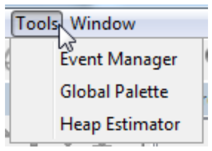
Asset Menu

These menu features are discussed in Graphics Composer Asset Management.



Tools Menu

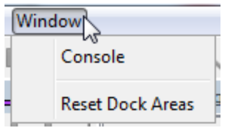
The Event Manager, Global Palette, and Heap Estimator are discussed in MHGC Tools.



Window Menu

Selecting **Console** opens the Output Console for the Graphics Composer. This console panel can be used to debug problems when the Graphics Composer boots up or during its operation.

Selecting **Reset Dock Areas** restores the MHGC panel configuration to the default setup by redocking all of the panels that have been undocked into separate windows.



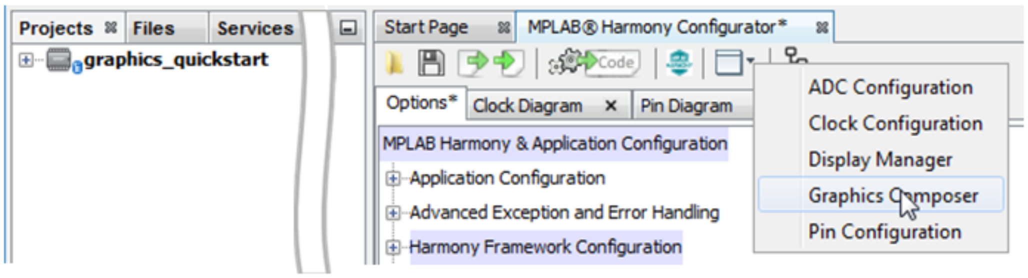
New Project Wizard

The New Project Wizard is launched from the Welcome dialog of the MPLAB Harmony Graphics Composer (MHGC), which supports the creation of a new graphics design, or the importing of an existing graphics design.

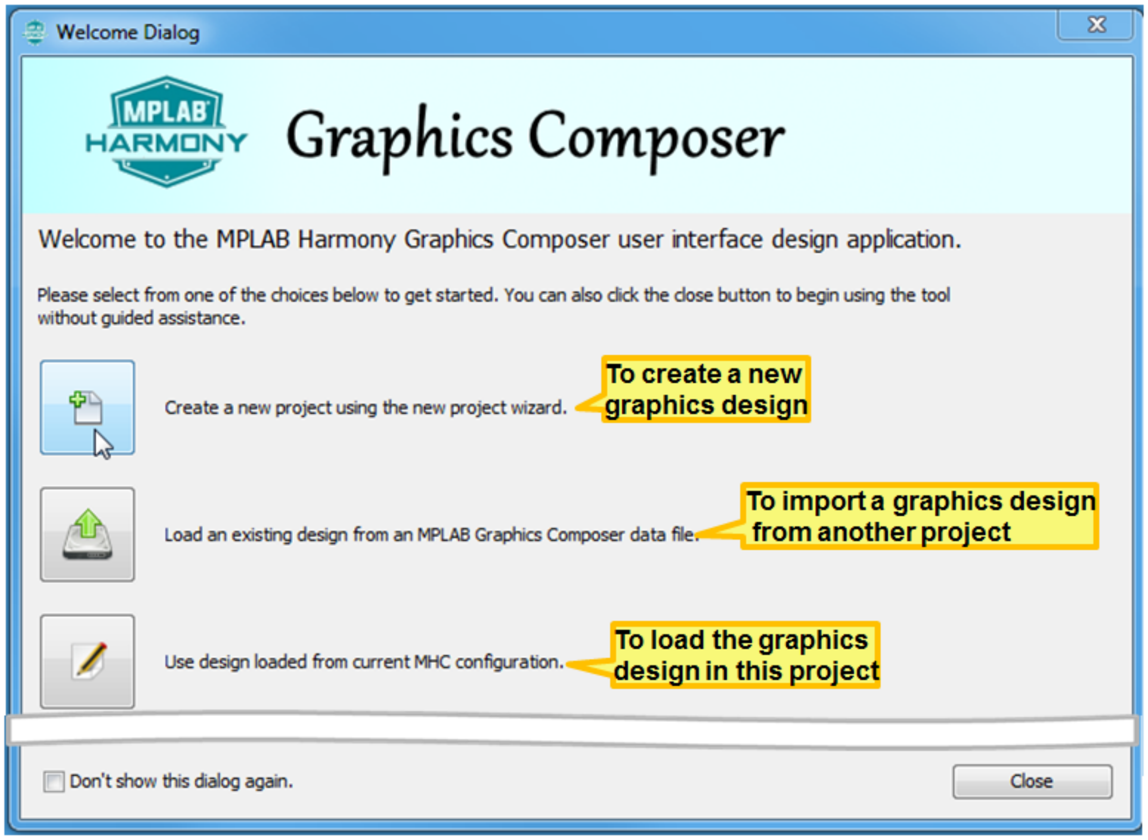
Description

Welcome Dialog window

The Welcome dialog is launched when the Graphics Composer is chosen from the Launch Utility pull-down menu in the MPLAB Harmony Configurator (MHC).



The window has three options:



Welcome to the MPLAB Harmony Graphics Composer user interface design application.

Please select from one of the choices below to get started. You can also click the close button to begin using the tool without guided assistance.

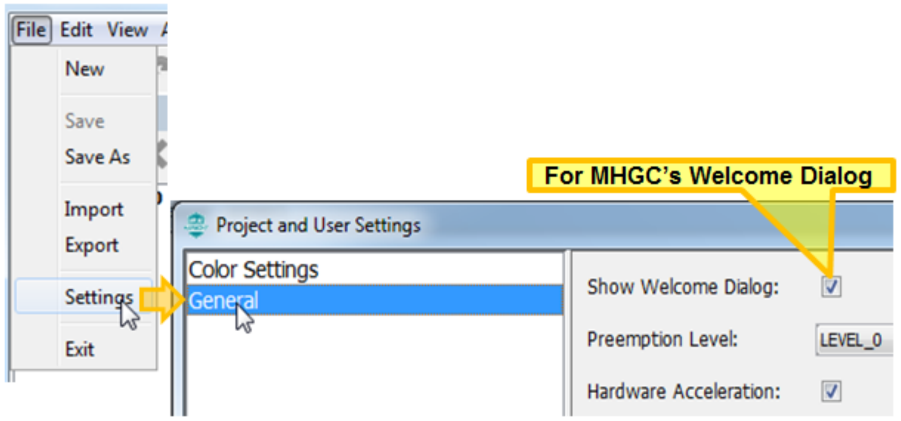
Create a new project using the new project wizard.

Load an existing design from an MPLAB Graphics Composer data file.

Use design loaded from current MHC configuration.

To create a new graphics design
To import a graphics design from another project
To load the graphics design in this project
Don't show this dialog again.
Close

Note: If this window does not appear, it can be re-enabled from MHGC's *File* > *Settings* > *General* menu.



Show Welcome Dialog: ☑

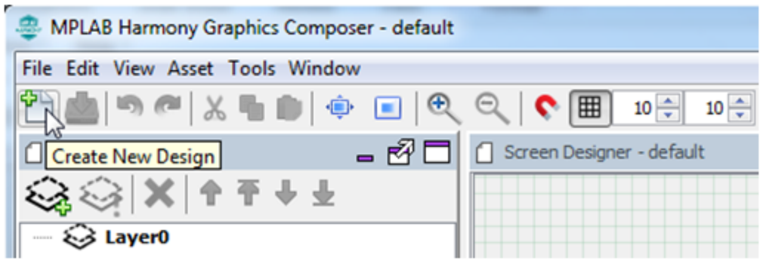
Preemption Level: LEVEL_0

Hardware Acceleration: ☑

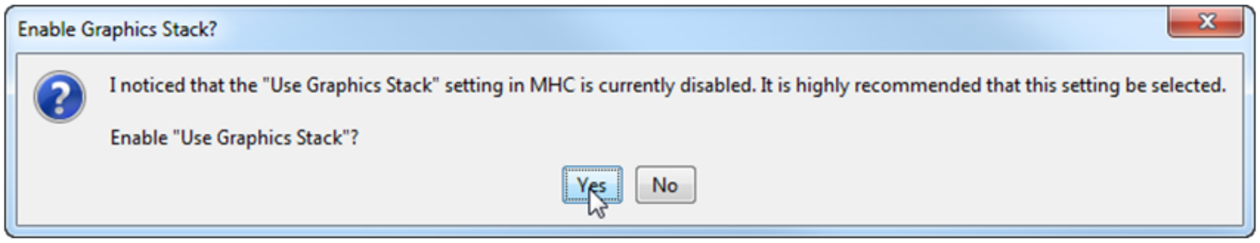
New Project Wizard Windows

Selecting the first icon in the Welcome dialog launches the New Project Wizard. There are four stages in the New Project Wizard: Color Mode, Memory Size, Project Type, and Finish.

The New Project Wizard can also be launched from the first icon (Create New Design) of MHGC's tool bar:


MPLAB Harmony Graphics Composer - default
File Edit View Asset Tools Window
10
10
Create New Design
Screen Designer - default
Layer0

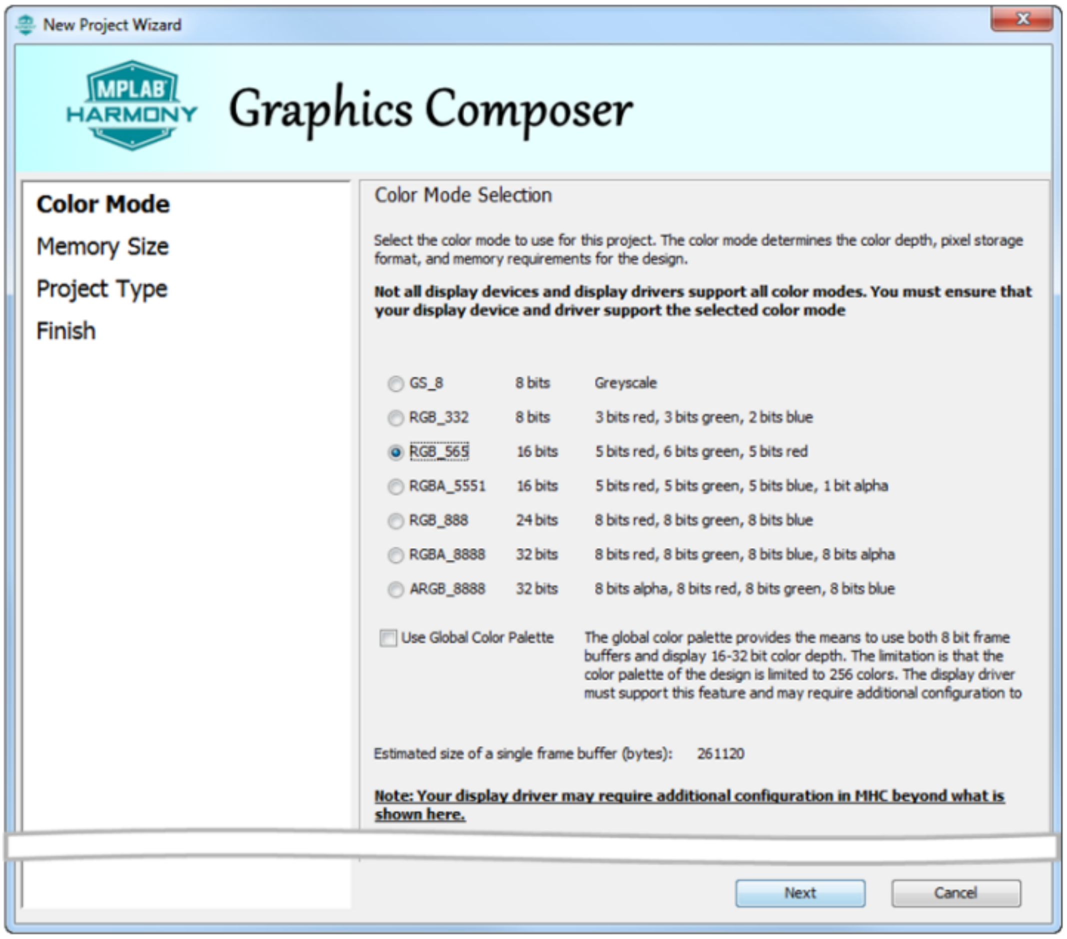
If the Graphics Stack has not been enabled in MHC, an Enable Graphics Stack? dialog will appear to support enabling the Graphics Stack before proceeding:


Enable Graphics Stack?

I noticed that the "Use Graphics Stack" setting in MHC is currently disabled. It is highly recommended that this setting be selected.

Enable "Use Graphics Stack"?

In the Color Mode stage you choose the Display Color Mode for the new graphics design:



Color Mode Selection

Select the color mode to use for this project. The color mode determines the color depth, pixel storage format, and memory requirements for the design.

Not all display devices and display drivers support all color modes. You must ensure that your display device and driver support the selected color mode

GS_8	8 bits	Greyscale
RGB_332	8 bits	3 bits red, 3 bits green, 2 bits blue
● RGB_565	16 bits	5 bits red, 6 bits green, 5 bits red
RGBA_5551	16 bits	5 bits red, 5 bits green, 5 bits blue, 1 bit alpha
RGB_888	24 bits	8 bits red, 8 bits green, 8 bits blue
RGBA_8888	32 bits	8 bits red, 8 bits green, 8 bits blue, 8 bits alpha
ARGB_8888	32 bits	8 bits alpha, 8 bits red, 8 bits green, 8 bits blue

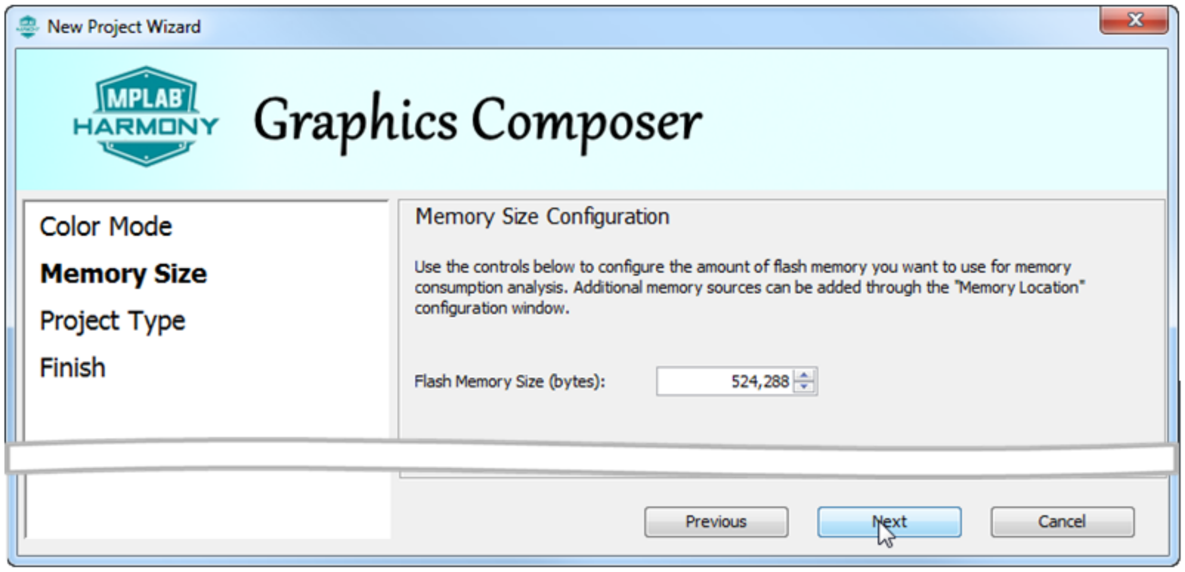
Use Global Color Palette — The global color palette provides the means to use both 8 bit frame buffers and display 16-32 bit color depth. The limitation is that the color palette of the design is limited to 256 colors. The display driver must support this feature and may require additional configuration to

Estimated size of a single frame buffer (bytes): 261120

Note: Your display driver may require additional configuration in MHC beyond what is shown here.

This choice must be supported by the graphics controller defined in the board support package of the project configuration. (If you make a mistake it can be corrected using MHGC's *File > Settings > Project Color Mode* menu.) Click **Next** moves the wizard on to the next stage.

The Memory Size stage configures the Program Flash allocated to memory use. This value is only used by the Graphics Composer's Asset menu Memory Configuration tool. The value used in the Memory Size stage can be updated using the Configuration sub-tab of the Memory Configuration tool window.



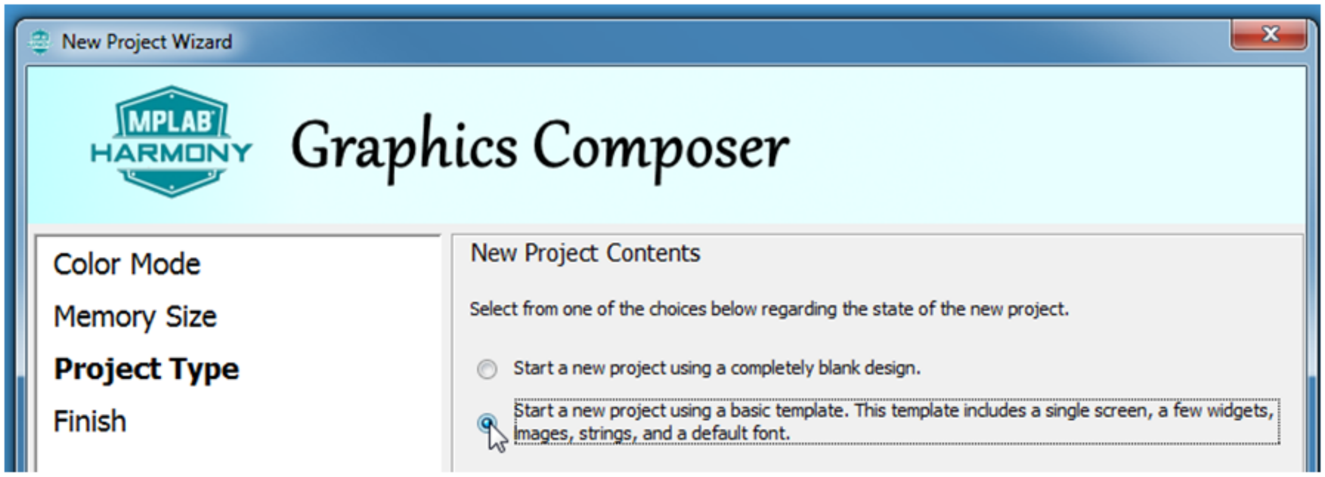
Memory Size Configuration

Use the controls below to configure the amount of flash memory you want to use for memory consumption analysis. Additional memory sources can be added through the "Memory Location" configuration window.

Flash Memory Size (bytes): 524,288

Clicking **Previous** returns to the Color Mode stage and clicking **Next** moves the wizard to the Project Type stage.

There are two choices at the Project Type stage: A completely blank design, and a template design with a few predefined widgets.



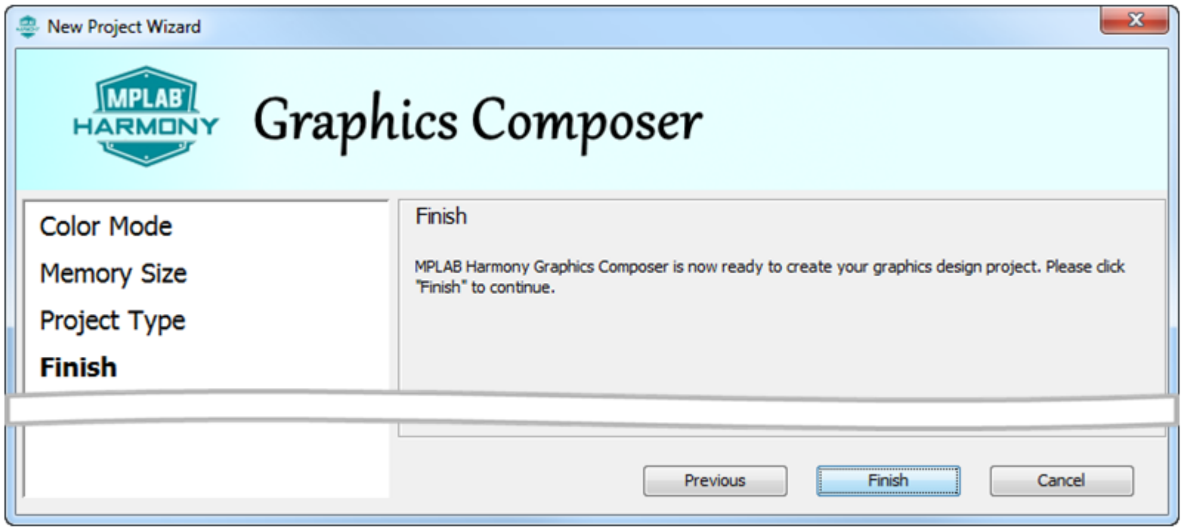
New Project Contents

Select from one of the choices below regarding the state of the new project.

○ Start a new project using a completely blank design.

● Start a new project using a basic template. This template includes a single screen, a few widgets, images, strings, and a default font.

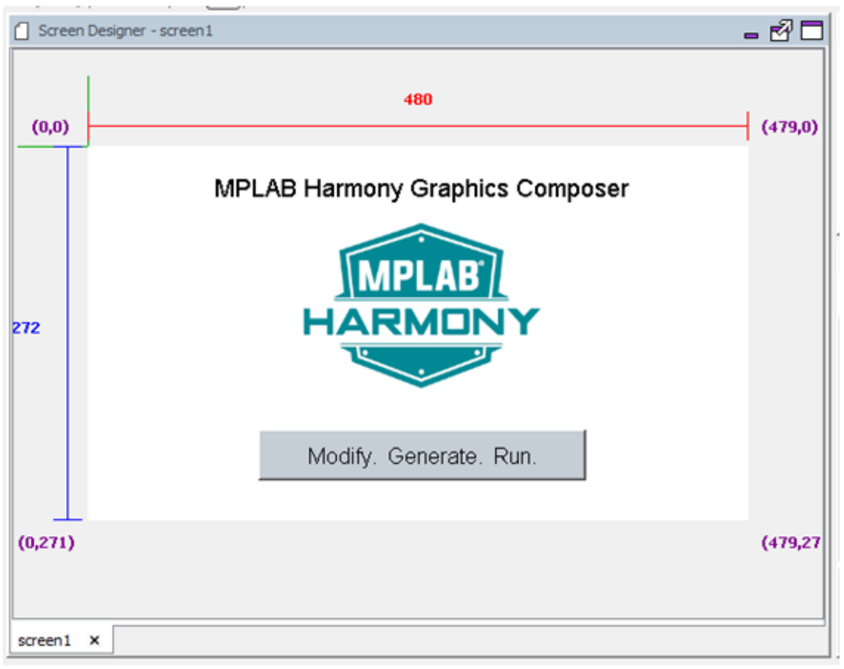
Clicking **Previous** returns to the Memory Size stage, and clicking **Next** moves the wizard to the Finish stage.



Finish

MPLAB Harmony Graphics Composer is now ready to create your graphics design project. Please click "Finish" to continue.

If the “Template” project type was chosen, MHGC’s Screen Designer will show:



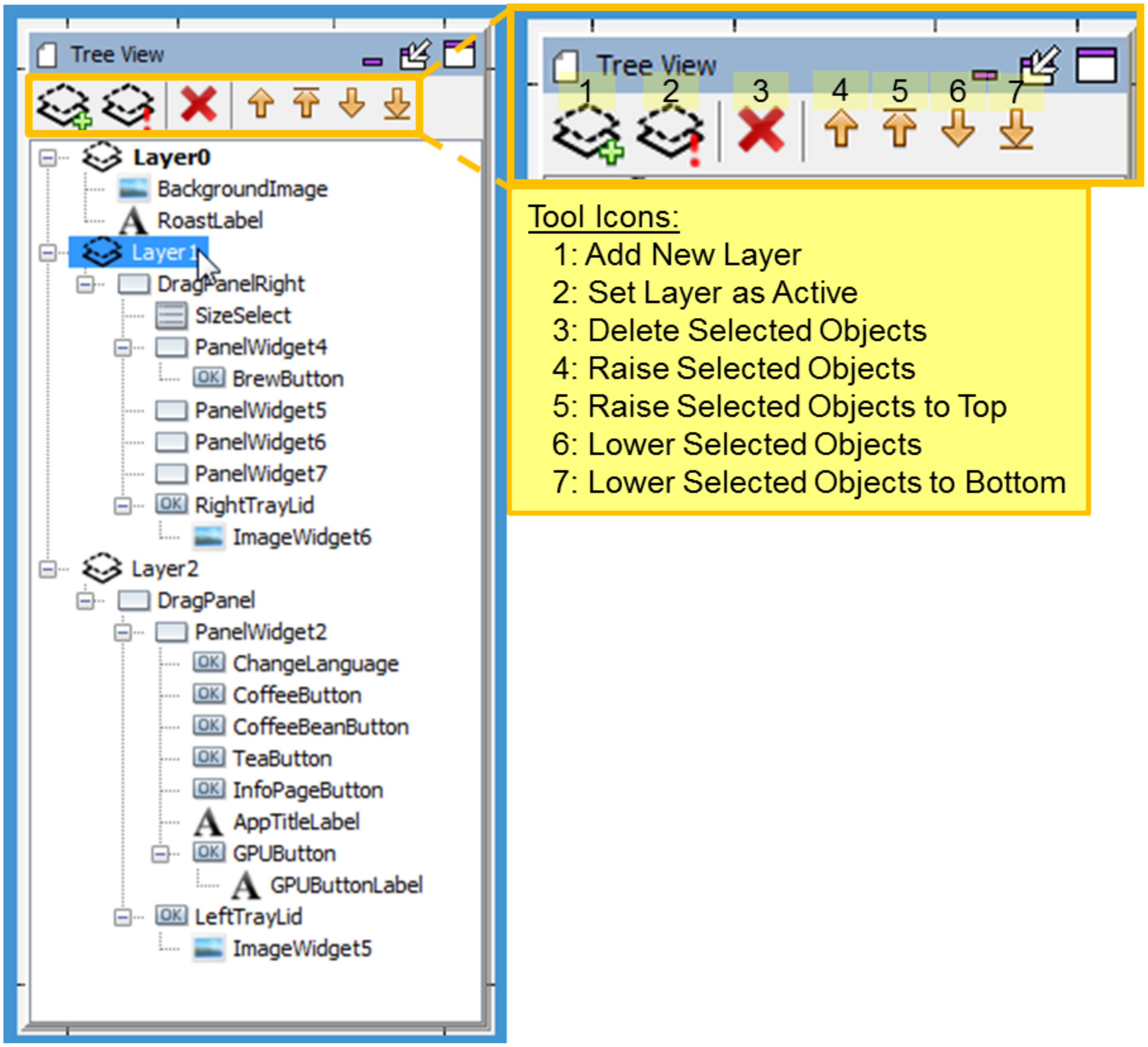
Tree View Panel

The organization of application widgets and layers, including draw order, is managed using this panel.

Description

Example Tree View

The following Tree View (from main screen of the Aria Coffee Maker demonstration shows the tree structure for a screen with three layers.



Tool Icons:

1: Add New Layer
2: Set Layer as Active
3: Delete Selected Objects
4: Raise Selected Objects
5: Raise Selected Objects to Top
6: Lower Selected Objects
7: Lower Selected Objects to Bottom

The tool icons for this panel support layers and managing screen objects (layers/widgets).

Drawing Order and Parent/Child Relationships

The Graphics Composer Tree View panel allows you to organize the widgets per screen in the desired drawing order (z-order). It also allows for the user to organize the widgets into parent – child hierarchies to allow for the paint algorithm to draw the groups together in event of motion or re-draw. Please note that this does not associate or group the widgets by functionality. (Example: a group of radio buttons might not belong to a common parent on the screen.) This parent-child relationship is limited to the widgets location on the screen, motion on the screen and the drawing order on the screen. (Exceptions to this general rule are the Editor > Hidden, Alpha Blending properties, and layer single versus double buffering. These apply to the parent and all the parent's children.)

The tree is traversed depth-first. This means that the z-order goes background (bottom of z-order) to foreground (top of z-order) as we go from top to bottom in the list of widgets, i.e., ImageWidget1, is the widget at the bottom of the z-order and the PanelWidget1 is the topmost widget on the z-order. The tree structure can be arranged and modified by dragging the widgets and releasing it under the desired parent/child. Also, the list can be modified by using the up/down arrows provided at the header of the Composer Widget tree window to traverse the tree.

Editor > Hidden Property for Layers

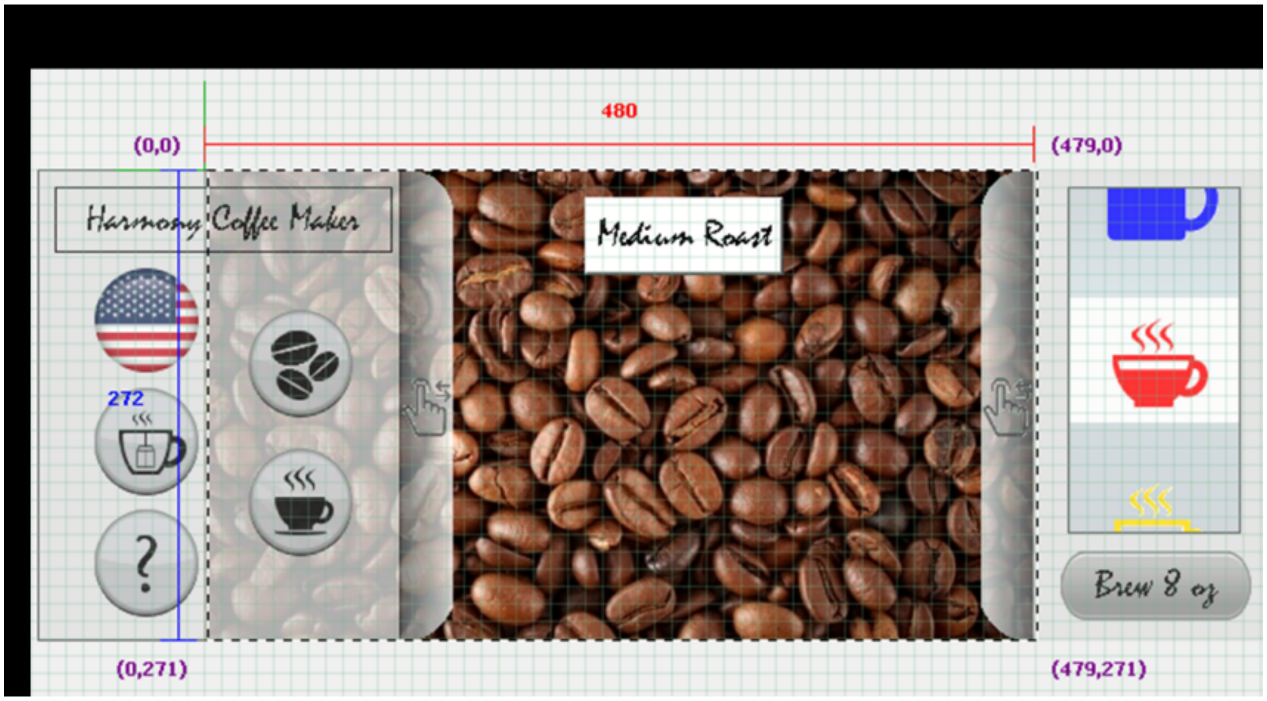
Setting *Editor* > *Hidden* hides the layer and all its children from the Graphics Composer Screen Designer but does not affect how the layer and its children are displayed when the application is running. This can be useful when designing complex screens with overlapping layers.

Alpha Blending Property for Layers

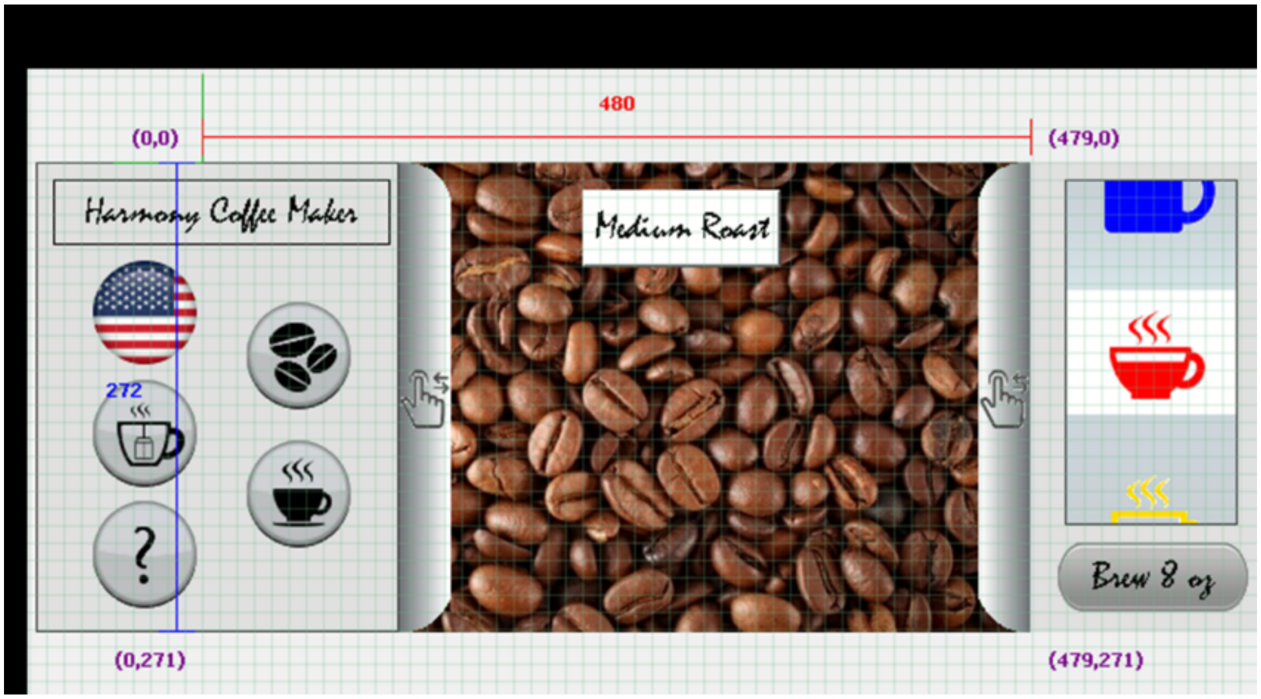
Enabling Alpha Blending allows you to control the transparency of a layer and all its children. You can experiment with Alpha Blending in the Aria Coffee Maker demonstration. Load the project, launch MHC, and then start the Graphics Composer Screen Designer. There are three layers (Layer0, Layer1, Layer2) in this demonstration. Layer1 (the drag panel on the right) and Layer2 (the drag panel on the left) have Alpha Blending enabled with Alpha Amount = 225. Setting the Alpha Amount to 255 is the same as disabling Alpha Blending (255 = no transparency). Setting the

Alpha Amount to 0 makes the layer invisible (0 = full transparency, i.e., invisible).

The following figure shows the main screen with Alpha Blending = 225.

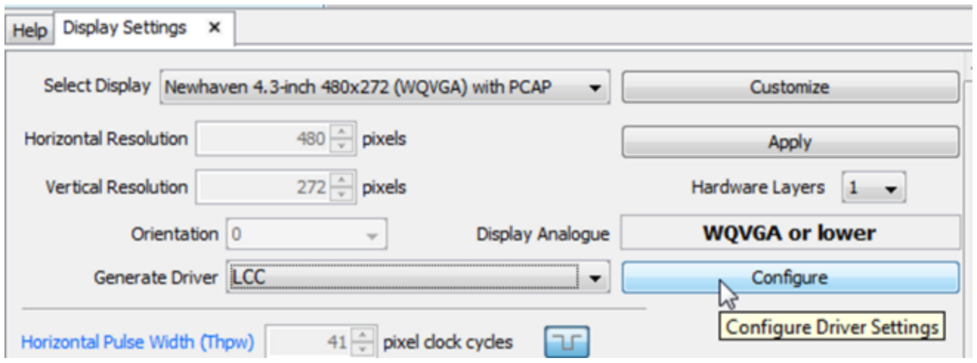


The following figure shows the main screen with Layer 2’s Alpha Blending = 255.



Double Buffering for Layers

Graphics double buffering for the LCC driver is enabled in the Display Manager’s Display Setting screen when the application is changed to use external memory instead of internal. Click **Configure** to bring up the LCC Driver Configuration Settings Window.


Help | Display Settings ×

Select Display: Newhaven 4.3-inch 480x272 (WQVGA) with PCAP

Horizontal Resolution: 480 pixels

Vertical Resolution: 272 pixels

Orientation: 0

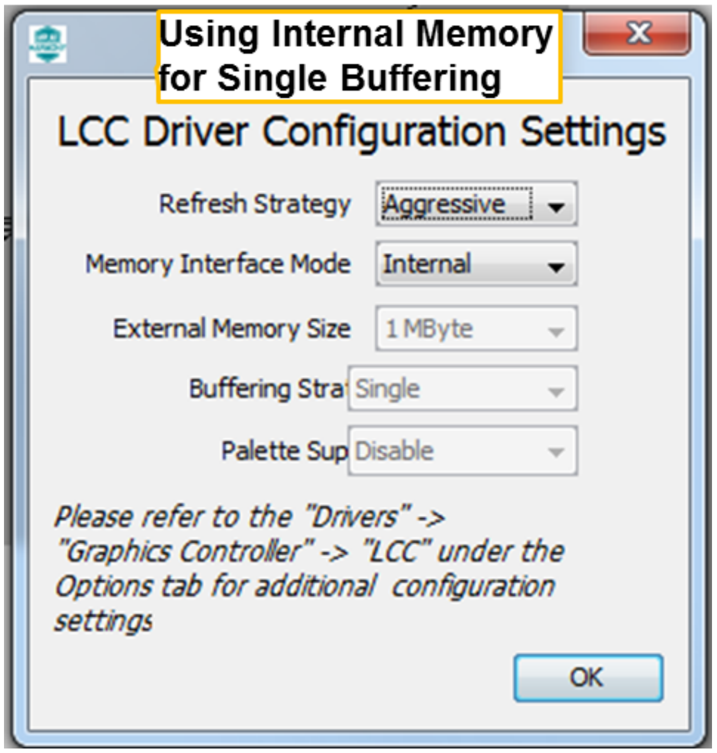
Generate Driver: LCC

Hardware Layers: 1

Display Analogue: WQVGA or lower

Horizontal Pulse Width (Thpw): 41 pixel clock cycles

Configure the memory according to whether double buffering is to be enabled for the display's layer or layers.


Using Internal Memory for Single Buffering

LCC Driver Configuration Settings

Refresh Strategy: Aggressive

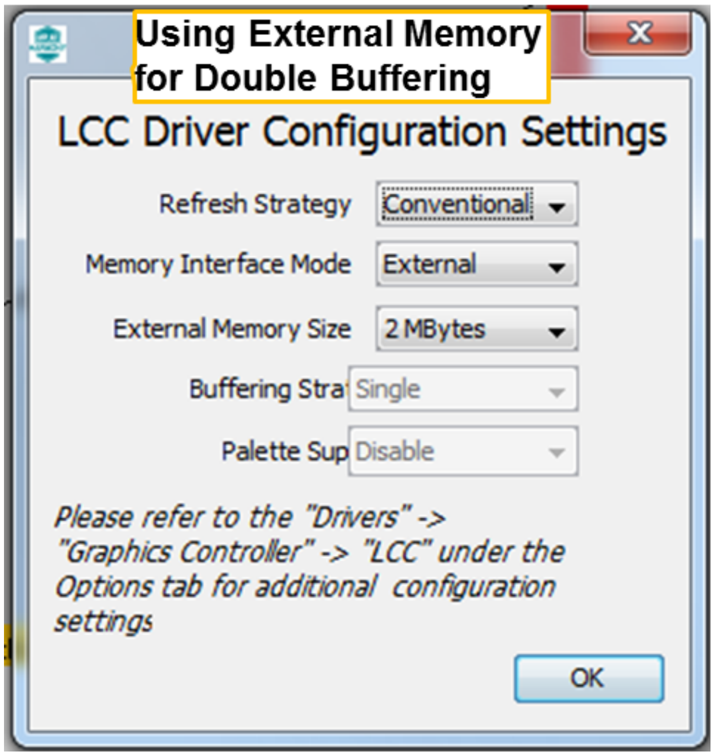
Memory Interface Mode: Internal

External Memory Size: 1 MByte

Buffering Stra: Single

Palette Sup: Disable

Please refer to the "Drivers" -> "Graphics Controller" -> "LCC" under the Options tab for additional configuration settings



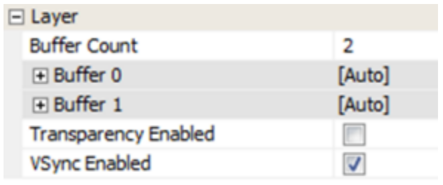
Using External Memory for Double Buffering

LCC Driver Configuration Settings

Refresh Strategy	Conventional
Memory Interface Mode	External
External Memory Size	2 MBytes
Buffering Stra	Single
Palette Sup	Disable

Please refer to the "Drivers" -> "Graphics Controller" -> "LCC" under the Options tab for additional configuration settings

Increasing the Buffer Count of a layer from 1 to 2 enables double buffering for the layer and all its child widgets. To prevent tearing on the display when switching from one buffer to the other, VSync Enabled should also be selected.



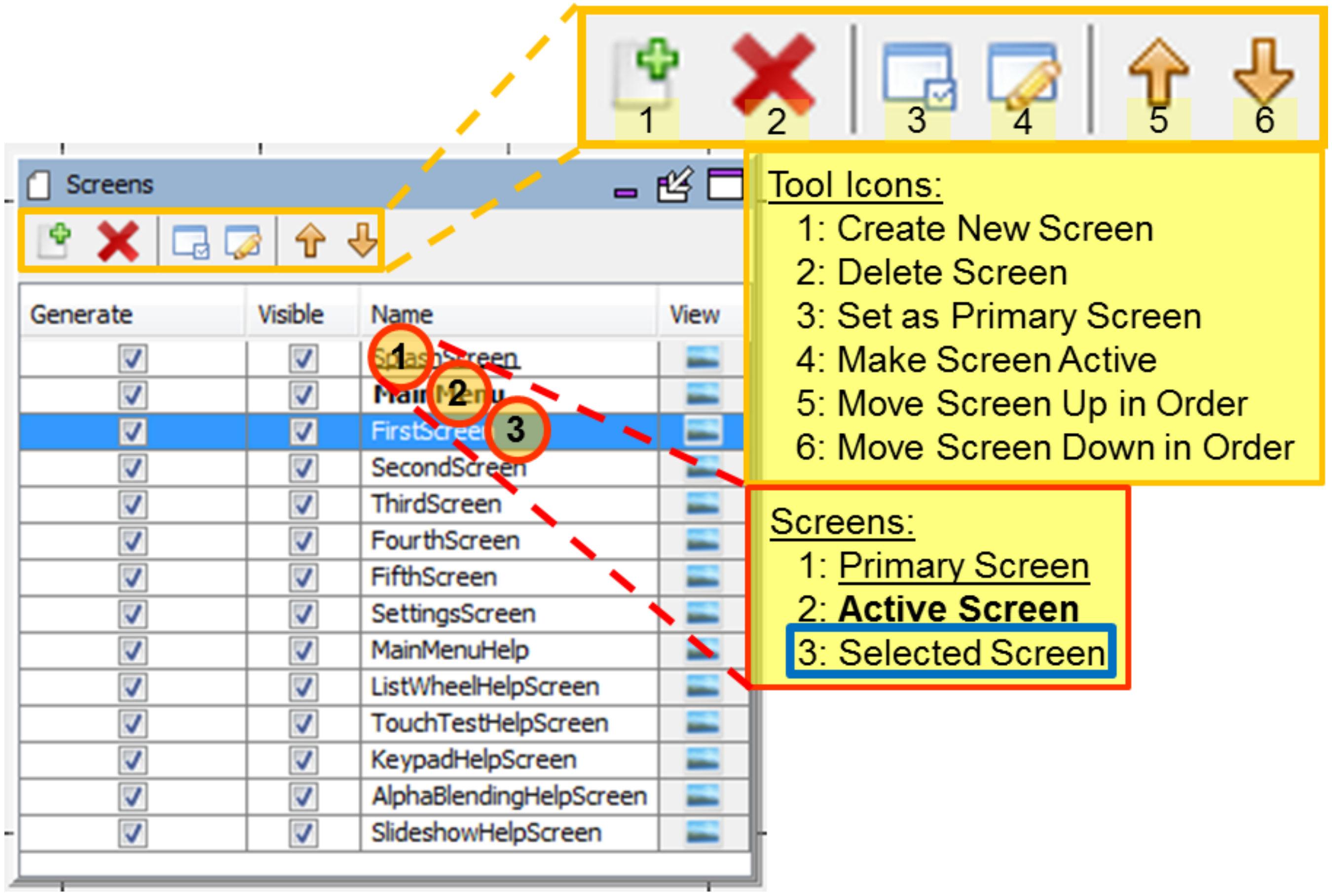
Layer	
Buffer Count	2
Buffer 0	[Auto]
Buffer 1	[Auto]
Transparency Enabled	☐
VSync Enabled	✓

Screens Panel

Application screens are managed using the Screens Panel.

Description

The Screens panel tab manages all the application's screens, as shown in the following figure.



Generate	Visible	Name	View
✓	✓	SplashScreen	
✓	✓	MainMenu	
✓	✓	FirstScreen	
✓	✓	SecondScreen	
✓	✓	ThirdScreen	
✓	✓	FourthScreen	
✓	✓	FifthScreen	
✓	✓	SettingsScreen	
✓	✓	MainMenuHelp	
✓	✓	ListWheelHelpScreen	
✓	✓	TouchTestHelpScreen	
✓	✓	KeypadHelpScreen	
✓	✓	AlphaBlendingHelpScreen	
✓	✓	SlideshowHelpScreen	

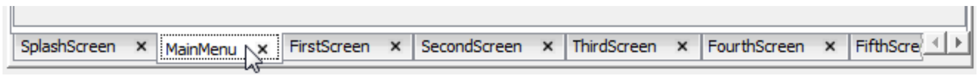
Note: These screens are examples from the Aria Showcase demonstration project

The underlined screen name identifies the primary screen (in this case, SplashScreen.) The **bold** screen name identifies the currently active screen in the Graphics Composer Screen Designer window (in this case MainMenu.) The blue background identifies the selected screen (i.e., the screen that is manipulated by the tool icons), in this case FirstScreen.

Window Toolbar

The window's tools icons support:

1. **Create New Screen** – Create a new screen. You will be prompted for the name of the new screen, which will appear at the bottom of the Screens list.
2. **Delete Screen** – Delete the selected screen. This removes the selected screen from the application.
3. **Set as Primary Screen** – Sets the selected screen as the default screen displayed by the application at boot-up.
4. **Make Screen Active** – This selected screen is displayed in the Screen Designer panel. You can also select the active screen by clicking on the screen's tab at the bottom of the Screen Designer panel.



5. **Move Screen Up in Order** – Moves the selected screen up in the list of screens, which is useful in organizing a large list of screens, but has no other significance.
6. **Move Screen Down in Order** – Moves the selected screen down in the list of screens.

Useful in organizing a large list of screens, but has no other significance.

Window Columns

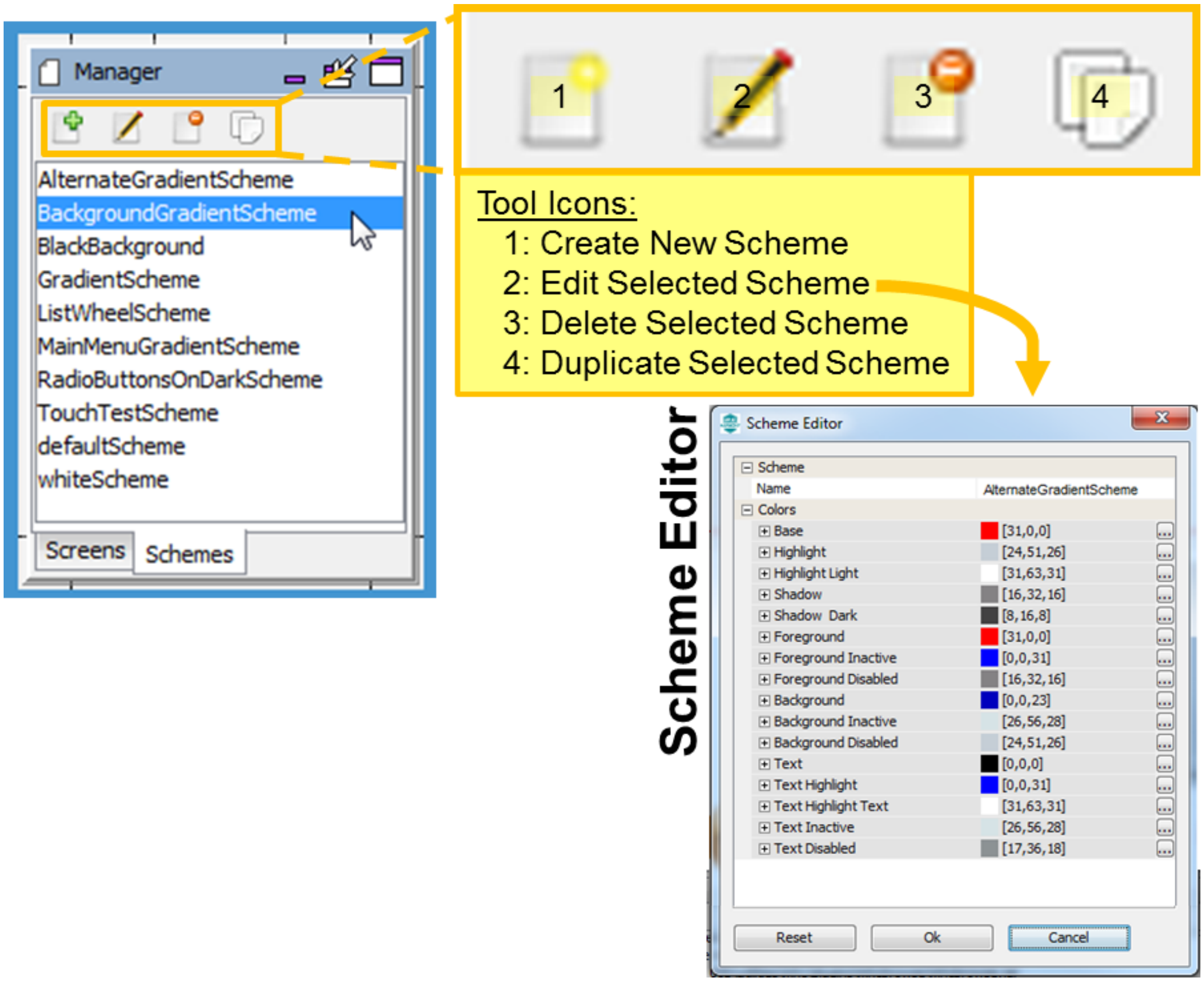
The **Generate** check box is used in selecting those screens that will be included in the application when MPLAB Harmony Configurator (MHC) generates/regenerates the application. (This, along with the Enabled check box for languages, allows customization of the application's build to support different end uses from the same project.) The **Visible** check box can be cleared to hide a screen from the sub-tabs located at the bottom of the Screen Designer. The **View** column provides a mouse-over preview of the screen.

Schemes Panel

Application color schemes are managed using the Schemes Panel.

Description

Color schemes for the application's graphics are managed using the Schemes sub-tab.



AlternateGradientScheme
BackgroundGradientScheme
BlackBackground
GradientScheme
ListWheelScheme
MainMenuGradientScheme
RadioButtonsOnDarkScheme
TouchTestScheme
defaultScheme
whiteScheme

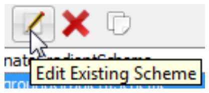
Tool Icons:
1: Create New Scheme
2: Edit Selected Scheme
3: Delete Selected Scheme
4: Duplicate Selected Scheme

Scheme Editor

Scheme	
Name	AlternateGradientScheme
Colors	
Base	🟥 [31,0,0]
Highlight	⬜ [24,51,26]
Highlight Light	⬜ [31,63,31]
Shadow	⬛ [16,32,16]
Shadow Dark	⬛ [8,16,8]
Foreground	🟥 [31,0,0]
Foreground Inactive	🟦 [0,0,31]
Foreground Disabled	⬛ [16,32,16]
Background	🟦 [0,0,23]
Background Inactive	⬜ [26,56,28]
Background Disabled	⬜ [24,51,26]
Text	⬛ [0,0,0]
Text Highlight	🟦 [0,0,31]
Text Highlight Text	⬜ [31,63,31]
Text Inactive	⬜ [26,56,28]
Text Disabled	⬛ [17,36,18]

Editing a Scheme

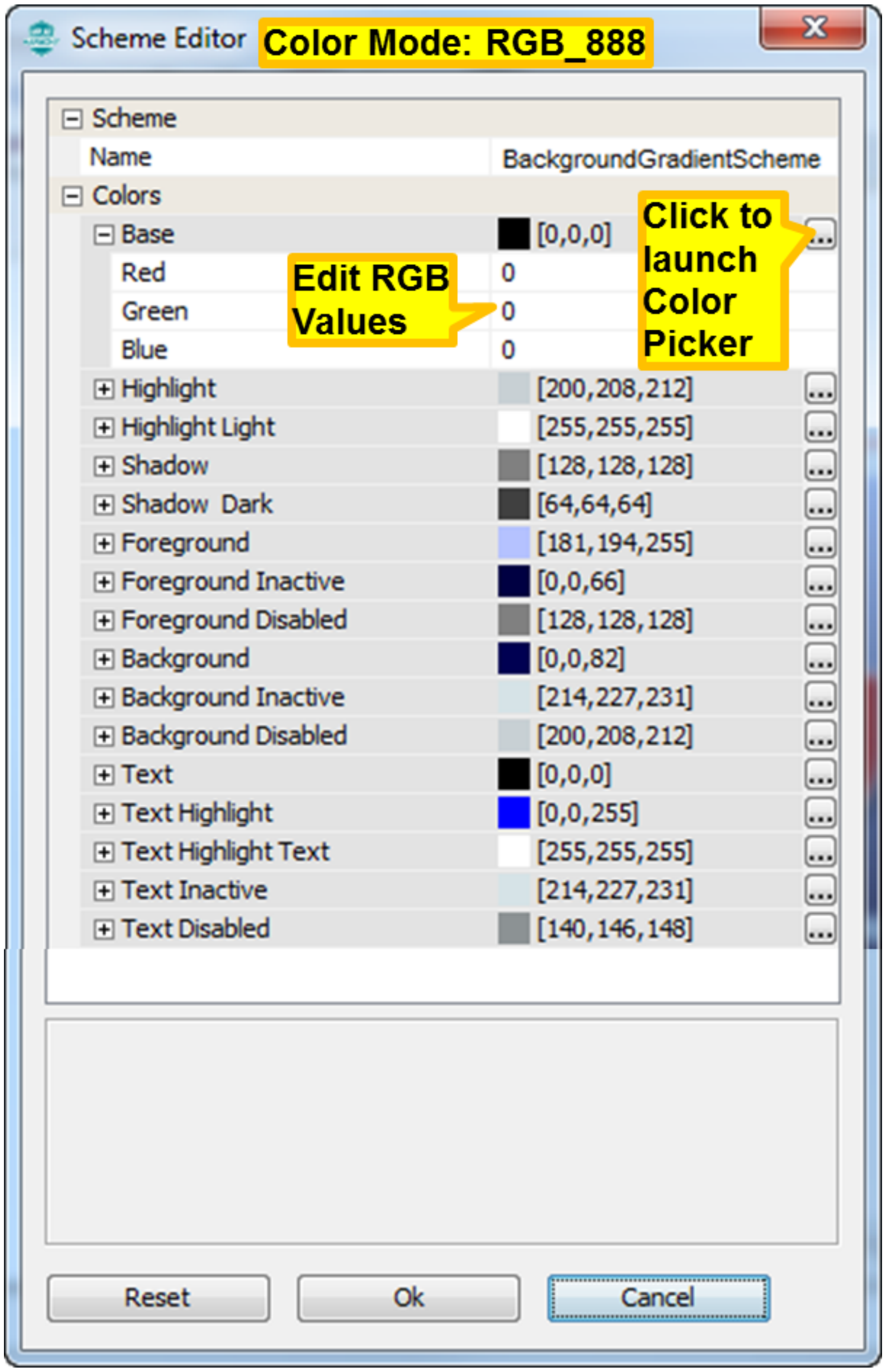
To edit an existing scheme, select the scheme from the list and click **Edit**.



The Scheme Editor dialog appears, which allows you to change the colors associated with this display scheme.

Scheme Editor

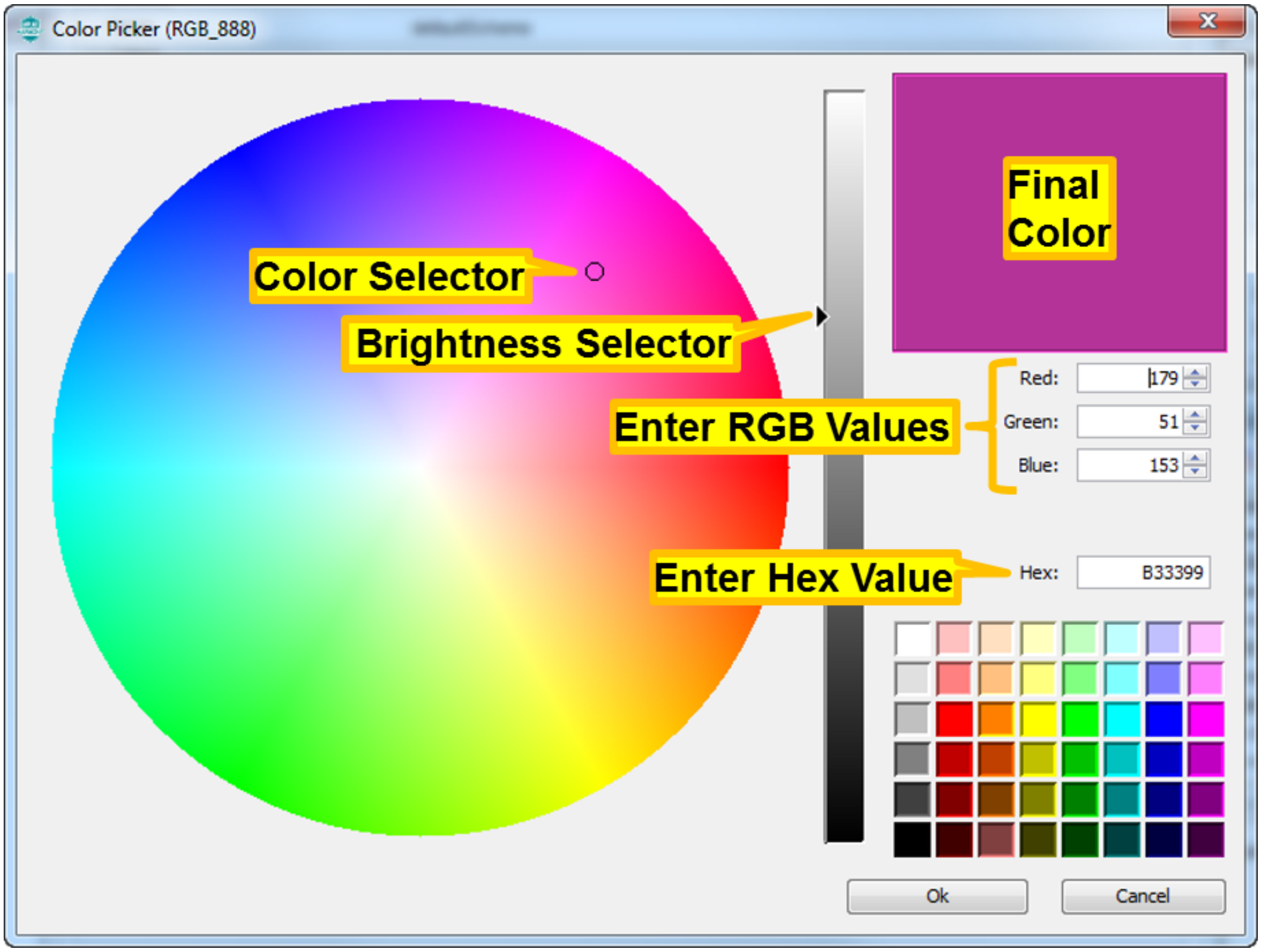
The Scheme Editor window supports editing the individual colors of a color scheme. Clicking the ellipsis (**...**) opens the Color Picker window.



⊟ Scheme	
Name	BackgroundGradientScheme
⊟ Colors	
⊟ Base	■ [0,0,0]
Red	0
Green	0
Blue	0
⊞ Highlight	[200,208,212]
⊞ Highlight Light	[255,255,255]
⊞ Shadow	[128,128,128]
⊞ Shadow Dark	[64,64,64]
⊞ Foreground	[181,194,255]
⊞ Foreground Inactive	[0,0,66]
⊞ Foreground Disabled	[128,128,128]
⊞ Background	[0,0,82]
⊞ Background Inactive	[214,227,231]
⊞ Background Disabled	[200,208,212]
⊞ Text	[0,0,0]
⊞ Text Highlight	[0,0,255]
⊞ Text Highlight Text	[255,255,255]
⊞ Text Inactive	[214,227,231]
⊞ Text Disabled	[140,146,148]

Color Picker

The Color Picker window allows the user to easily select a color by providing a color wheel, brightness gauge, and some common predefined color choices. The user can change the individual color values or input a number in Hexadecimal format. The end result is displayed in the top right corner.



Red: 179
Green: 51
Blue: 153
Hex: B33399

Options

Provides information on the defeatured Options window.

Description

In v2.03b, MPLAB Harmony Graphics Composer user interface provided a third window along with Screens and Schemes, named **Options**. Beginning with v2.04b of MPLAB Harmony, these options are now located within the *File* > *Settings* menu (see Menus for details).

Widget Tool Box Panel

The Widget Tool Box panel is the interface by which users add widgets into the screen representation.

Description

All the available graphics widgets are shown in the Widget Tool Box:

MPLAB Harmony Graphics Composer provides automatic code optimization by keeping track of the widgets that are currently being used. When MHC generates or regenerates the application, only the Graphics Library code necessary for your design is included in the project.

There are two primary methods for creating new widget objects: clicking and dragging. To add a new layer to a screen use the Screens sub-tab.

Click Method

The following actions can be performed by using the Click method:

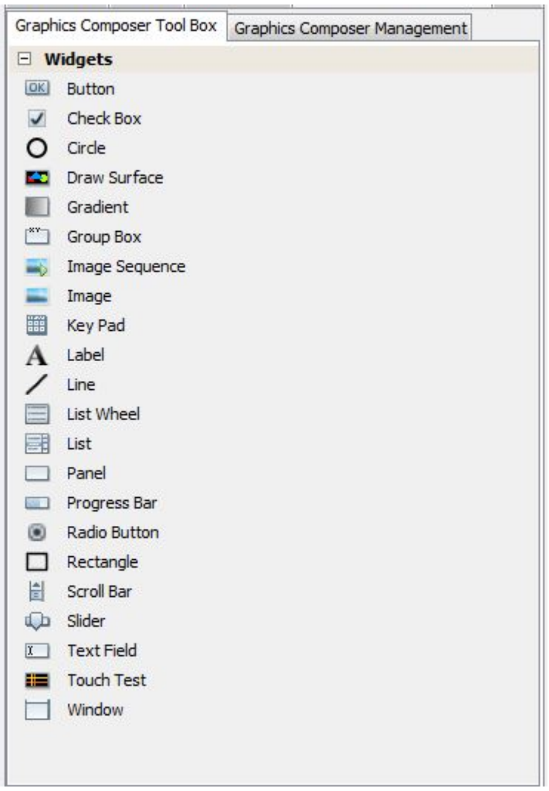
- Clicking an item selects it as active. Users can then move the cursor into the screen window and view a representation of the object about to be added.
- Left-clicking confirms the placement of the new object
- Right-clicking aborts object creation
- Clicking the active item again deactivates it

Drag Method

Dragging and dropping a tool item into the Screen Designer Window creates a new instance of an object. When dragging a tool item, releasing the cursor outside of the Screen Designer Window cancels the drag operation.

Widget List

The Graphics Composer Tool Box is the interface by which users add widgets into the screen representation.



Click Method

The following actions can be performed by using the Click method:

- Clicking an item selects it as active. Users can then move the cursor into the screen window and view a representation of the object about to be added.
- Left-clicking confirms the placement of the new object
- Right-clicking aborts object creation
- Clicking the active item again deactivates it.

Drag Method

Dragging and dropping a tool item into the Screen Designer Window creates a new instance of an object. When dragging a tool item, releasing the cursor outside of the Screen Designer Window cancels the drag operation.

Automatic Code Optimization

MPLAB Harmony Graphics Composer keeps track of the types of widgets that are used and updates the MHC Tree constantly to ensure that only the Graphics Library code necessary for your design is included in the project.

Widgets

Widgets can be configured by using the Properties Editor on the right side of the MHGC interface. Each widget has multiple properties to manage their appearance as well as their functioning. Most properties related to appearance are common between widgets, though some widgets require specific property entries.

Button - A binary On and Off control with events generation for Press and Release state.

Check Box - A selection box with Checked and Unchecked states, and associated events.

Circle - A graphical object in the shape of a circle.

Draw Surface - A container with a callback from its paint loop. a draw surface lets the application have a chance to make draw calls directly to the HAL during LibAria's paint loop.

Gradient - A draw window that can be associated with a gradient color scheme. This allows for color variation on the window.

Group Box - A container with a border and a text title. With respect to functionality, a group box is similar to a window.

Image Sequence - A special widget that allows image display on screen to be scheduled and sequenced. You can select the images to be displayed, the order for display, and the durations.

Image - Allows an image to be displayed on screen. The size and shape of the widget decides the visible part of the image, as scaling is not enabled for images at this time.

Key Pad - A key entry widget that can can be designed for the number of entries divided as specified number of rows and column entries. The widget has a key click event that can be customized.

Label - A text display widget. This does not have any input at runtime capability. A Text Field widget serves that purpose.

Line - A graphical object in the shape of a line.

List Wheel - Allows multiple radial selections that were usually touch-based selections and browsing.

List - Allows making lists of text and image items. The list contents, number of items, and the sequence can be managed through a List Configuration dialog box in the Properties box.

Panel - A container widget that is a simpler alternative to DrawSurface as it does not have the DrawSurface callback feature.

Progress Bar - Displays the progress pointer for an event being monitored through the "Value Changed" event in the Properties Editor.

Radio Button - A set of button widgets that are selected out of the group one at a time. The group is specified by the Group property in the Properties Editor.

Note: The radio buttons in the same group must have the same group number specified in their properties.

Rectangle - A graphical object in the shape of a rectangle.

Scroll Bar - Intended to be used with another relevant widget such as the List Wheel to scroll up and down. It has a callback each time the value is changed. The callback allows users to trigger actions to be handled on the scroll value change event.

Slider - Can change values with an external input such as touch. Event callbacks on value change are also available through the Properties Editor.

Text Field - Text input can be accepted into the text field from an external input or from a widget such as keypad. Event 'Text Changed' in the Properties Editor is used for accepting the input.

Touch Test - Allows tracking of touch inputs. Each new touch input is added to the list of displayed touch coordinates. The input is accepted through the 'Point Added' event callback in the Properties Editor.

Window - A container widget similar to the Panel but has the customizable title bar.

Properties Editor Panel

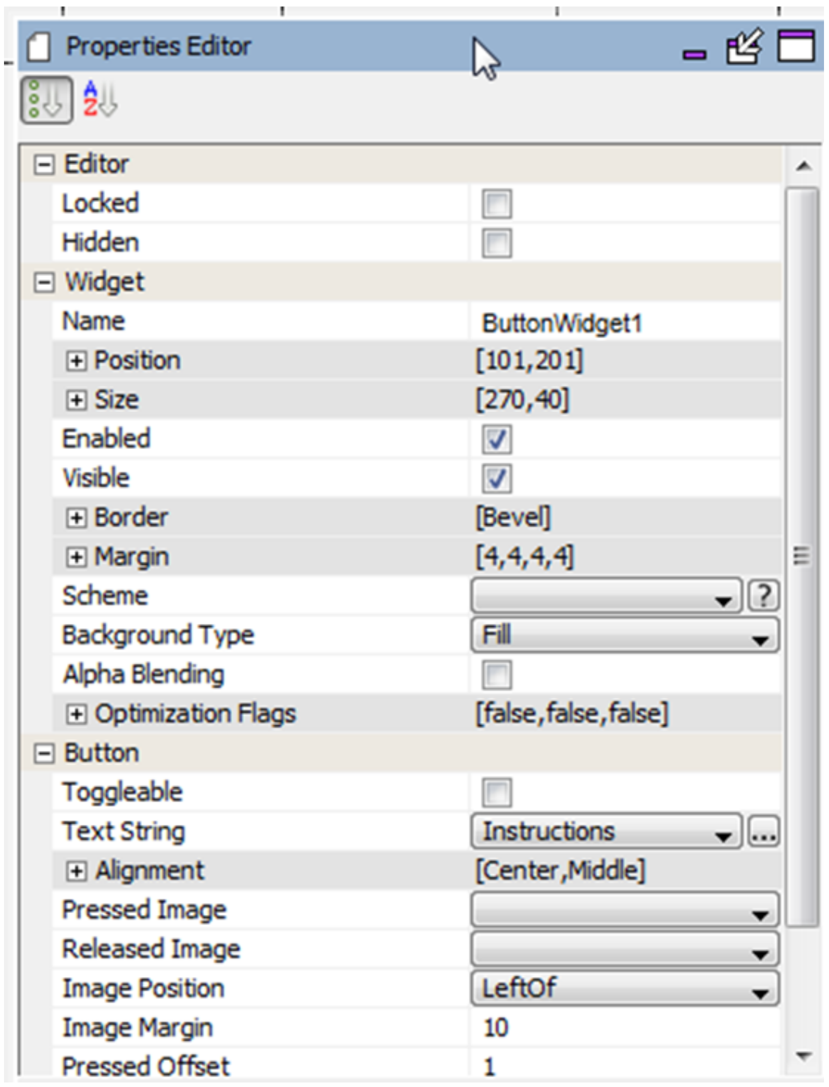
The properties for all layers and widgets are managed using this panel.

Description

The Properties Editor displays options for the currently-selected object (layer or widget), or the options for the active screen if no objects are selected. To edit an option: left-click the value in the right column and then change the value. Some values have an ellipsis that will provide additional options. In the previous case, the ellipsis button will display the Color Picker dialog.

Some properties, like the screen width and height, are locked and cannot be edited. Other properties offer check boxes and combo-type drop-down box choices. Some properties are grouped together like the Position and Size entries. Individual values of the group can be edited by expanding the group using the plus symbol. For example, the following figure shows properties for a Button Widget.

A new support feature is the ? icon to the right of the Scheme pull-down, which brings up an “Scheme Helper” for the widget showing how it is colored when using a Bevel border. For a more complete description of widget coloring, see Widget Colors.


Properties Editor

⊟ Editor	
Locked	☐
Hidden	☐
⊟ Widget	
Name	ButtonWidget1
⊞ Position	[101,201]
⊞ Size	[270,40]
Enabled	✓
Visible	✓
⊞ Border	[Bevel]
⊞ Margin	[4,4,4,4]
Scheme	
Background Type	Fill
Alpha Blending	☐
⊞ Optimization Flags	[false,false,false]
⊟ Button	
Toggleable	☐
Text String	Instructions
⊞ Alignment	[Center,Middle]
Pressed Image	
Released Image	
Image Position	LeftOf
Image Margin	10
Pressed Offset	1

Object Properties

Provides information on widget, layer, and screen properties.

Description

Object Properties and Event Actions

Each widget has a structured tree of properties, visible under the MPLAB Harmony Configurator window on the right of the standard window setup within MPLAB X IDE. Most widget properties have a Related Event action that can be use in an event or macro to change or set a property from the application.

Each widget has 3-4 property sets:

Editor – Controls the behavior of layers and widgets under the MPLAB Harmony Graphics Composer Suite Editor.

Property Name	Type	Description	Related Event Actions
Locked	Boolean	Locks the object (widget), preventing changes by the designer. Only affects the object (widget) in the editor.	N/A
Hidden	Boolean	Hides the widget and its children in the designer window. Only affects the appearance of the widget in the editor.	N/A
Active	Boolean	For layers only. Sets the layer as active. Any objects (widgets) added to the screen will be added to this layer.	N/A
Locked to Screen Size	Boolean	For layers only. Locks the layer size to the size of the display's screen.	N/A

Widget – Controls the behavior of screens, layers, and widgets on the display.

Property Name	Type	Description	Related Event Actions
Name	String	Editable name for each object. By default, widgets are named NameWidget1, …,NameWidgetN. For example: ButtonWidget1, ButtonWidget2, … .	N/A
Position	[X,Y] Pair of Integers	Location on the layer of the upper left corner of the widget or the location on the display of the upper left corner of the layer. Measured in display pixels. X is measured from left-to-right and Y is measured from up-to-down from the upper left corner of the parent object (typically a Layer or Panel).	Adjust Position, Set X Position, Set Y Position
Size	[X,Y] Pair of Integers	X: Width, Y: Height of object, in display pixels.	Adjust Size, Set Size, Set Width, Set Height
Enabled	Boolean	Is the object enabled? Disabled objects are not built into the display's firmware.	Set Enabled
Visible	Boolean	Is the object visible by default? Object visibility can be manipulated in firmware using laWidget_GetVisible and laWidget_SetVisible.	Set Visible
Border	Widget Border	Choices are: { None | Line | Bevel }.	Set Border Type
Margin	Integer	Four integers ([Left,Top,Right,Bottom]) defining the widget's margins on the display, in display pixels.	Set Margins
Scheme		Color scheme assigned to the layer or widget. Blank implies the default color scheme.	Set Scheme
Background Type		Sets the background of the layer or widget. Choices are { None | Fill | Cache }. In MPLAB Harmony v2.03, this type was Boolean. Now, Off = None, On = Fill. With Fill selected, the widget's background is one solid color. With Cache selected, a copy (cache) of the framebuffer is created before the widget is drawn and this cache is used to fill the background of the widget. This supports transparent widgets in front of complex widgets, such as JPEG images. Instead of rerendering the JPEG image, it is just drawn from the cache.	Set Draw Background
Alpha Blending	Boolean	Is alpha blending enabled for this layer or widget and all of its children? If enabled, specify the amount of alpha blending as an 8-bit integer. Zero makes the object invisible, whereas 255 makes the background invisible.	N/A

Widget Advanced – Advanced control of layers and widgets

Optimization Sub-Property Name	Type	Description	Related Event Actions
Draw Once	Boolean	Indicates that the widget should draw once per screen Show Event. All other attempts to invalidate or paint the widget will be rejected.	N/A
Force Opaque	Boolean	Provides a hint to the renderer that the entire area for this widget is opaque. Useful for widgets that may use something like an opaque image to fill the entire widget rectangle despite having fill mode set to None. This can help reduce unnecessary drawing.	N/A
Local Redraw	Boolean	Provides a "hint" to the widget's renderer that the widget is responsible for removing old pixel data. This can avoid unnecessary redrawing.	N/A

Important! Use Local Redraw only if you know what you're doing!

Widget Name (e.g., Button Check Box, Circle, etc.) – Optional properties tied to each widget. See **Dedicated Widget Properties and Event Actions**.

Events – Associates widget events with event call-backs. For example, you can enable and specify a button pressed event and button release event for the Button widget.

For each event you specify:

- Enabled/Disabled Check box – To enable or disable (default) the event.

- Event Callback – Selected from the Event Editor Action List.

There are additional Event actions that do not correspond to any specific property:

- Set Parent – Set the parent of the object, including no parent.

Dedicated Widget Properties and Event Actions

Button

Property Name	Type	Description	Related Event Actions
Toggleable	Boolean	Is button toggle enabled?	Set Toggleable
Pressed	Boolean	If Toggleable is enabled, provide default state of the button. This can be used to see the colors of an asserted button.	Set Press State
Text String		Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical		Text string alignment within the button object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Pressed Image		Select image used for pressed state. Default: no image.	Set Pressed Image
Released Image		Select image used for pressed state. Default: no image.	Set Released Image
Image Position		Position of image relative to button text. Choices are: { LeftOf | Above | RightOf | Below | Bottom }.	Set Image Position
Pressed Offset	Integer	Offset of button contents when pressed. In Pixels. The X and Y position of the button contents is offset by this amount.	Set Pressed Offset

Check Box

Property Name	Type	Description	Related Event Actions
Text String		Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical		Text string alignment within the button object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Checked	Boolean	Default state of the check box.	Set Check State
Unchecked Image		Select image used for widget's unchecked state. Default: no image.	Set Unchecked Image
Checked Image		Select image used for the widget's checked state. Default: no image.	Set Checked Image
Image Position		Position of image relative to check box text. Choices are: : { LeftOf | Above | RightOf | Below | Bottom }.	Set Image Position
Image Margin	Integer	Space between image and text. In Pixels.	Set Image Margin

Circle

Property Name	Type	Description	Related Event Actions
X	Integer	X offset of circle's center, from widget's upper left hand corner, in pixels.	N/A
Y	Integer	Y offset of circle's center, from widget's upper left hand corner, in pixels.	N/A
Radius	Integer	Circle's radius, in pixels.	Set Radius

Draw Surface – No additional properties.

Gradient

Property Name	Type	Description	Related Event Actions
Direction		Gradient draw direction. Choices are: { Right | Down | Left | Up }.	Set Direction

Group Box

Property Name	Type	Description	Related Event Actions
Text String		Select widget's text string from the Select String Dialog.	Set Text
Alignment		Text string alignment within the widget. Choices are: { Left|Center|Right }.	Set Alignment

Image Sequence

Property Name	Type	Description	Related Event Actions
Sequence Configuration Dialog		Specify image sequence by using the Image Sequence Configuration Dialog window.	Set Entry Image, Set Entry Horizontal Alignment, Set Entry Vertical Alignment, Set Entry Duration, Set Image Count
Starting Image	Integer	Selects the first image to be shown.	Set Active Image
Play By Default	Boolean	Will image sequence play automatically?	N/A
Repeat	Boolean	Should the image sequence repeat?	Set Repeat Additional related event actions: , Show Next, Start Playing, Stop Playing.

Image Widget

Property Name	Type	Description	Related Event Actions
Image		Select image used.	Set Image
Alignment: • Horizontal • Vertical		Image alignment within the image object. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment

Key Pad

Property Name	Type	Description	Related Event Actions
Row Count	Integer	Number of key pad rows.	None.
Column Count	Integer	Number of key pad columns.	None.
Key Pad Configuration Dialog	(see Description)	The Key Pad dialog window has the following: • Width – Integer. Width of each key, in pixels. • Height – Integer. Height of each key, in pixels. • Rows – Integer. Number of key rows. A duplicate of Row Count. • Columns – Integer. Number of key columns. A duplicate of Column Count.	 None. None. None. None.

-	-	Selecting one of the keys on the key pad diagram displays the Cell Properties for that key: • Enabled – Boolean. Disabled cells (keys) are made invisible. • Text String – Select key's text string from the Select String Dialog. • Pressed Image – Select image used for pressed state. Default: no image. • Released Image – Select image used for released state. Default: no image. • Image Position – Position of image relative to key text. Choices are: { LeftOf | Above | RightOf | Below | Behind }. • Image Margin – Integer. Space between image and text. In Pixels. • Draw Background – Boolean. Controls whether the key should fill its background rectangle. • Editor Action – Select the generic editor action that fires when the key is clicked. Choices are: { None | Accept | Append | • Editor Value String Other Key Event Actions:	Set Key Enabled Set Key Text Set Key Pressed Image Set Key Released Image Set Key Image position Set Key Image Margin None. Set Key Action Set Key Value Set Key Background Type

Label

Property Name	Type	Description	Related Event Actions
Text String		Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical		Text string alignment within the widget. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment

Line

Property Name	Type	Description	Related Event Actions
Start X	Integer	X start of line, in pixels, from upper left hand corner of the widget.	Set Start Point Position
Start Y	Integer	Y start of line, in pixels, from upper left hand corner of the widget.	Set Start Point Position
End X	Integer	X end of line, in pixels, from upper left hand corner of the widget.	Set End Point Position.
End Y	Integer	Y end of line, in pixels, from upper left hand corner of the widget.	Set End Point Position.

List

Property Name	Type	Description	Related Event Actions
Selection Mode		Select list selection mode. Choices are: {Single|Multiple|Contiguous}.	Set Selection Mode
Allow Empty Selection	Boolean	Is a list selection allowed to be empty?	Set Allow Empty Selection
Alignment		Horizontal text alignment. Choices are: { Left | Center | Right }.	Set Item Alignment
Icon Position		Position of list icons relative to list text. Choices are: { LeftOf | RightOf }.	Set Icon Position
Icon Margin		Space between icon and text, in pixels.	Set Icon Margin
List Configuration Dialog		Defines the string and icon image for each entry in the list.	Set Item Icon, Set Item Icon (actually sets item text). Additional Related Event Actions: Deselect All Items, Insert Item, Remove All Items, Remove Item, Select All Items, Set Item Selected, Toggle Item Select(ed).

List Wheel

Property Name	Type	Description	Related Event Actions
Alignment		Sets horizontal text alignment. Choices are: { Left | Center | Right }.	Set Item Alignment
Icon Position		Position of icons relative to text. Choices are: { LeftOf | RightOf }.	Set Icon Position
Icon Margin	Integer	Sets the space between icon and text. In pixels.	Set Icon Margin
Selected Index	Integer	Selects the default list item.	Set Selected Index
List Configuration Dialog		Defines the image/text for each entry in the list.	Set Item Icon, Set Item Icon (actually sets item text) Additional Related Event Actions: Append Item, Insert Item, Remove All Items, Remove Item, Select Next Item, Select Previous Item.

Panel – No additional properties.

Progress Bar

Property Name	Type	Description	Related Event Actions
Direction		Direction of progress bar. Choices are: { Right | Down | Left | Up }.	Set Direction
Value		Default value of the progress bar. The primitives `laProgressBarWidget_GetValue` and `laProgressBarWidget_GetValue` can be used to manipulate the widget's value during run time.	Set Value

Radio Button

Property Name	Type	Description	Related Event Actions
Text String		Select widget's text string from the Select String Dialog.	Set Text
Alignment: • Horizontal • Vertical		Text string alignment within the widget. Horizontal alignment. Choices are: { Left | Center | Right }. Vertical alignment. Choices are: { Top | Middle | Bottom }.	 Set Horizontal Alignment Set Vertical Alignment
Group	Integer	Radio Button Group Number. Default is -1, indicating no group. Only one radio button in a group can have a default selected value of On. All others in the group are Off	N/A
Selected	Boolean	If selected, the button has a default value of On. All other buttons in the group have a Selected value of Off.	Select
Selected Image		Select image used for selected state. Default: no image.	Set Selected Image
Unselected Image		Select image used for unselected state. Default: no image.	Set Unselected Image
Image Position		Position of image relative to widget text. Choices are: { LeftOf | Above | RightOf | Below | Behind }.	Set Image Position
Image Margin		Space between radio button image and text, in pixels.	Set Image Margin

Rectangle

Property Name	Type	Description	Related Event Actions
Thickness	Integer	Line thickness in pixels.	Set Thickness

Scroll Bar

Property Name	Type	Description	Related Event Actions
Orientation		Scroll bar orientation. Choices are: { Vertical | Horizontal }.	Set Orientation
Maximum	Integer	Maximum scroll value (minimum = 0.)	Set Maximum Value

Extent	Integer	Length of scroll bar slider, re scroll bar maximum value. Indicates the number of lines or size of window visible at each scroll setting.	Set Extent
Value	Integer	Initial scroll bar value.	Set Value, Set Value Percentage
Step Size	Integer	Step size value of scroll bar arrow buttons. (Min = 1, Max = 9999).	Set Step Size Additional Related Event Actions: Step Backward, Step Forward

Slider

Property Name	Type	Description	Related Event Actions
Orientation		Orientation of the slider. Choices are: { Vertical | Horizontal }.	Set Orientation
Minimum		Minimum slider value.	Set Minimum Value
Maximum		Maximum slider value.	Set Maximum Value
Value		Initial slider value.	Set Value, Set Value Percentage
Grip Size		Grip size of slider, from 10 to 9999, in pixels.	Set Grip Size Additional Related Event Actions: Step

Text Field

Property Name	Type	Description	Related Event Actions
Text String		Select widget's text string from the Select String Dialog.	Clear Text followed by Append Text
Alignment		Horizontal alignment. Choices are: { Left | Center | Right }.	Set Alignment
Cursor Enable		Boolean. Show blinking cursor while editing.	Set Cursor Enabled
Cursor Delay		Cursor delay in milliseconds. From 1 to 999,999.	Set Cursor Delay Additional Related Event Actions: Accept Text, Append Text, Backspace, Clear Text, Start Editing.

Touch Test – No dedicated properties.

Window

Property Name	Type	Description	Related Event Actions
Title String		Select widget's title string from the Select String Dialog.	Set Title
Icon Image		Select image used. Default: no image.	Set Icon
Image Margin	Integer	Space between icon and title, in pixels.	N/A

Layer Properties and Event Actions

The property list for a graphic layer is close in look and feel to that of a widget. Each Layer has three property sets: Editor (see above), Widget (see above), and Layer (see below).

Layer Properties

Property Name	Type	Description	Related Event Actions
Transparency Enabled	Boolean	Automatically mask out pixels of with a specified color. If enabled Specify:	N/A
Mask Color	Integer	Red/Green/Blue or Red/Green/Blue/Alpha color value	N/A
All Input Passthrough	Boolean	Allow input events to pass through this layer to layers behind it.	N/A
VSync Enabled	Boolean	Layers should swap only during vertical syncs.	N/A

Buffer Count	Integer	Integer number of frame buffers associated with this layer, either 1 or 2.	N/A
Buffer N		For each buffer (N= 1 or 2) you specify:	
Allocation Method		Buffer allocation method. Choices are: { Auto | Address | Variable Name } • Auto – Automatically allocate frame buffer space • Address – Specify a memory address • Variable Name – Use variable name as buffer location	N/A
Memory Address		If Address is the allocation method, specify the raw (physical) memory address as a hexadecimal number.	N/A
Variable Name	String	If Variable name is the allocation method, specify the variable name as a string value.	N/A

Screen Properties and Events

The property list for a screen shares the Name and Size properties with Layers and Widgets but has these unique properties.

Screen Properties

Property Name	**Type**	**Description**	**Related Event Actions**
Orientation		Display orientation: 0, 90, 180, 270 Degrees. This can also be set using the Display Manager.	N/A
Mirrored	Boolean	Enables screen mirroring.	N/A
Layer Swap Sync	Boolean	Enables that all layer buffer swapping happen at the same time, delaying lower layers until higher layers are finished drawing as well. For example, assume you make changes to layer 0 and layer 1 and you want to see those changes show up on the screen at the same time. Without this option you'd see layer 0's changes as soon as it finishes when layer 1 has not yet started drawing. This option will hold layer 0's swap operation until layer 1 finishes as well. **Note:** Currently, this property is only supported by the CLCD Graphics Controller Driver and is ignored by all other drivers.	N/A
Persistent	Boolean	Indicates that the screen should not free its widgets and memory when it is hidden. This results in faster load times and persistent data, but at the cost of higher memory consumption.	N/A
Export	Boolean	Includes this screen the application build. This can also be set using the Screens panel.	N/A
Primary	Boolean	Sets this screen as the primary screen. The primary screen is the first screen displayed when the application starts. This can also be done using the Screens Panel Generate check box.	N/A

Graphics Composer Asset Management

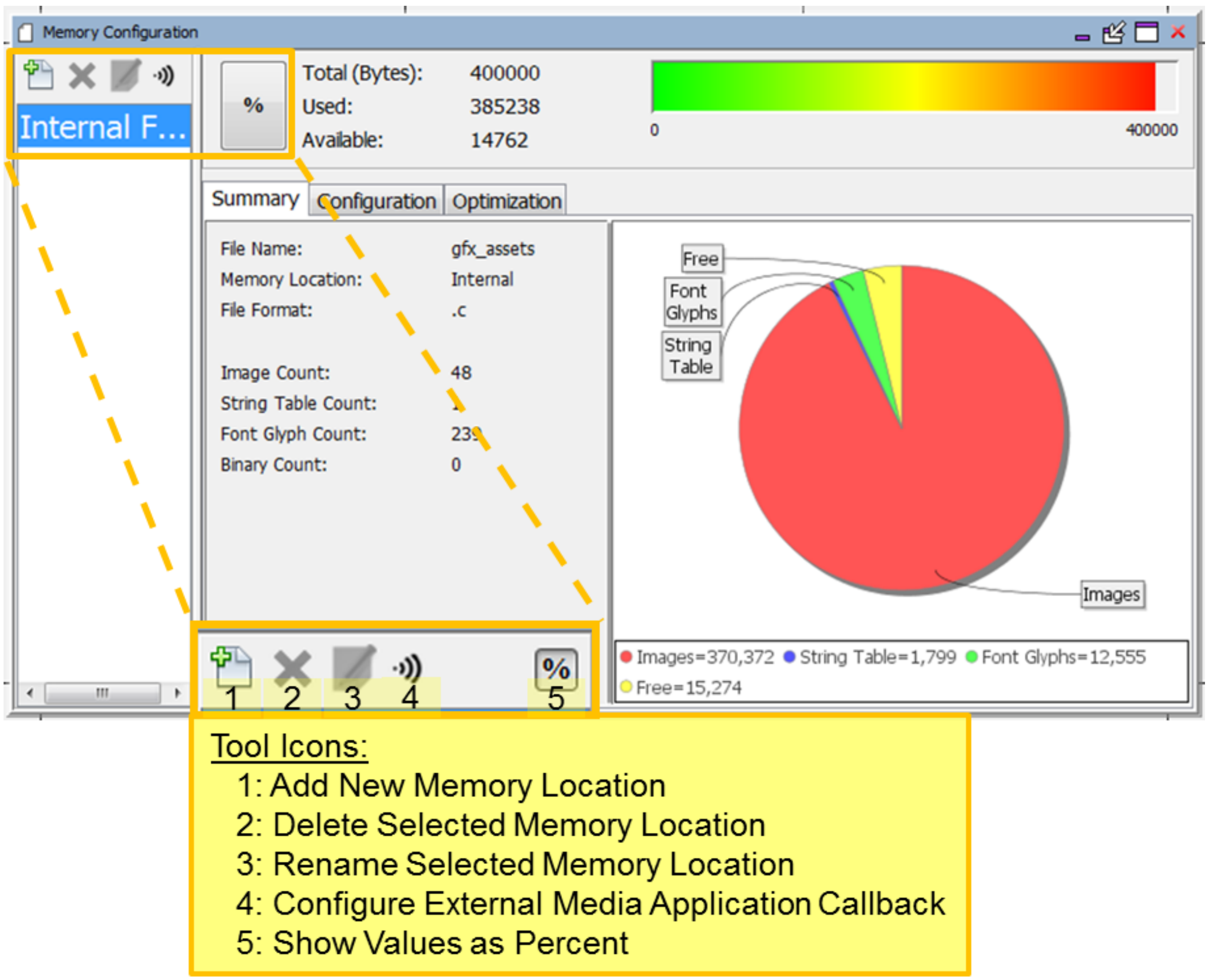
The Asset menu supports managing all graphical assets (memory, images, languages, fonts, strings, and binary data).

Memory Configuration

Provides information on configuring memory locations.

Description

The Memory Locations window is launched from the Graphics Composer's Asset menu. Selecting Memory Locations this brings up a window with three sub-tabs (in this example, the Aria Showcase demonstration is referenced):



Total (Bytes): 400000
Used: 385238
Available: 14762

File Name: gfx_assets
Memory Location: Internal
File Format: .c

Image Count: 48
String Table Count: 1
Font Glyph Count: 239
Binary Count: 0

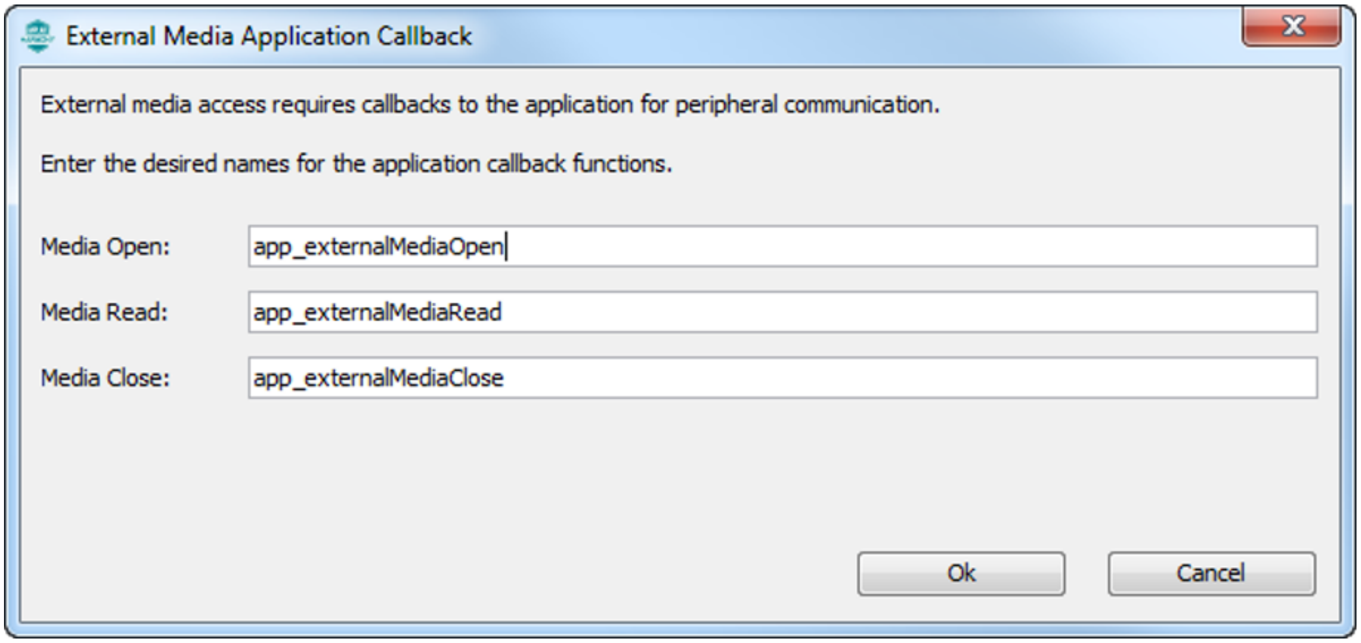
● Images=370,372 ● String Table=1,799 ● Font Glyphs=12,555
● Free=15,274

Tool Icons:
1: Add New Memory Location
2: Delete Selected Memory Location
3: Rename Selected Memory Location
4: Configure External Media Application Callback
5: Show Values as Percent

Window Toolbar

The window's tools icons support:

1. **Add New Memory Location** – This supports multiple external memory resources.
2. **Delete Selected Memory Location** – Removes a previously defined memory location.
3. **Rename Selected Memory Location** – Renames a previously defined memory location.
4. **Configure External Media Application Callback** – This allow definition of media callbacks, which must be provided in the project.



External media access requires callbacks to the application for peripheral communication.

Enter the desired names for the application callback functions.

Media Open: app_externalMediaOpen

Media Read: app_externalMediaRead

Media Close: app_externalMediaClose

5. **Show Values as Percent** – Memory utilization on the bar graph can be in bytes or as a percent of the total internal flash memory assigned to support asset storage. (That memory allocation is set using the Configuration sub-tab.)

The APIs for the external media callback functions are as follows:

```
GFX_Result app_externalMediaOpen(GFXU_AssetHeader* asset);
GFX_Result app_externalMediaRead(GFXU_ExternalAssetReader* reader,
                                 GFXU_AssetHeader* asset,
                                 void* address,
                                 uint32_t readSize,
                                 uint8_t* destBuffer,
                                 GFXU_MediaReadRequestCallback_FnPtr cb);
void app_externalMediaClose(GFXU_AssetHeader* asset);
```

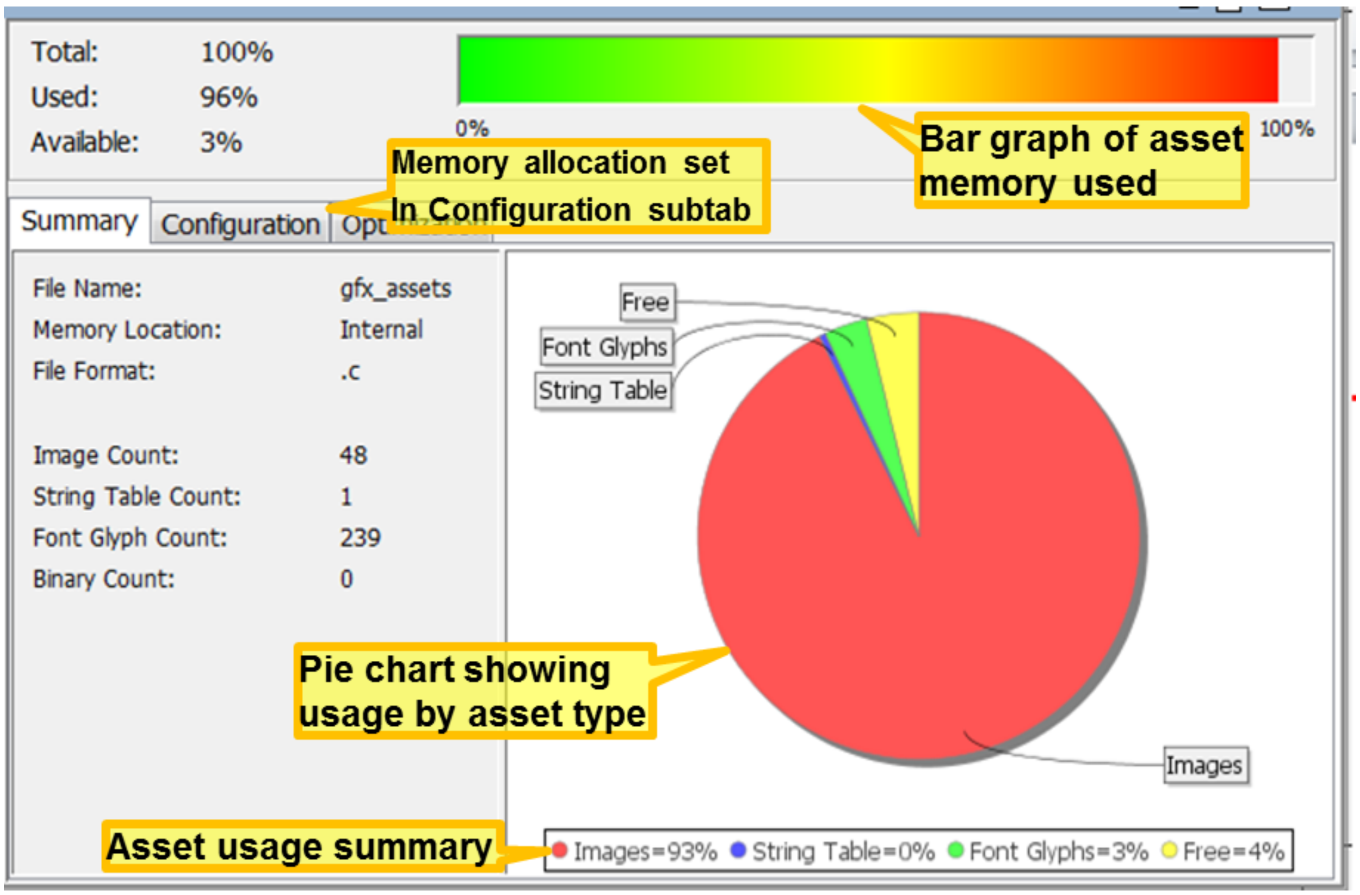
The graphics demonstration project, aria_external_resources, provides an example of how to write these callbacks. This demonstration supports three types of external memory: SQI External Memory, USB Binary, and USB with File System. Examples of these callbacks are found in the project's `app.c` file. The Aria demonstration projects Aria External Resources and Aria Flash provide more details on how to use external memory to store graphics assets.

Sub-tabs

There are three sub-tabs to this window.

Summary Sub-tab

This sub-tab summarizes program flash allocations for images, strings, and fonts.


Total: 100%
Used: 96%
Available: 3%

File Name: gfx_assets
Memory Location: Internal
File Format: .c

Image Count: 48
String Table Count: 1
Font Glyph Count: 239
Binary Count: 0

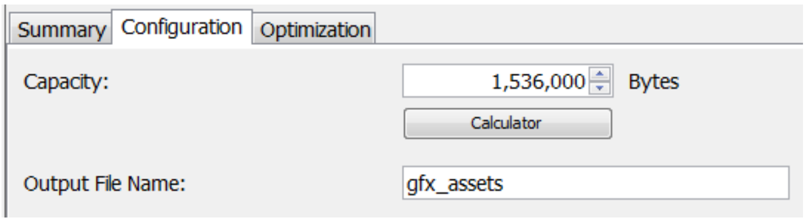
Images=93% String Table=0% Font Glyphs=3% Free=4%
0%
100%
Memory allocation set In Configuration subtab
Bar graph of asset memory used
Summary
Configuration
Optimization
Free
Font Glyphs
String Table
Images
Pie chart showing usage by asset type
Asset usage summary

The memory allocation shown for “Font Glyphs” measure the space that holds all the font glyphs used by the application, either by static strings or by glyph ranges defined in support of dynamic strings. Strings are defined by arrays of pointers to glyphs, so string memory usage measures the size of these arrays, not the actual font glyphs used. (“Glyph” is defined here.)

Note: The word “glyph” comes from the Greek for “carving”, as seen in the word hieroglyph – Greek for “sacred writing”. In modern usage, a glyph is an elemental or atomic symbol representing a readable character for purposes of communicating through writing.

Configuration Sub-tab

This sub-tab specifies the intended allocation of internal (program) flash memory to graphics assets (Total Size). (The default value is 1024 bytes.) It also names the graphics assets file name (here it will be `gfx_assets.c`). The allocation of flash is only used to scale the Total/Used/Available bar graph at the top of the display. Under sizing or oversizing this amount does not affect how the application is built.


Capacity: 1,536,000 Bytes

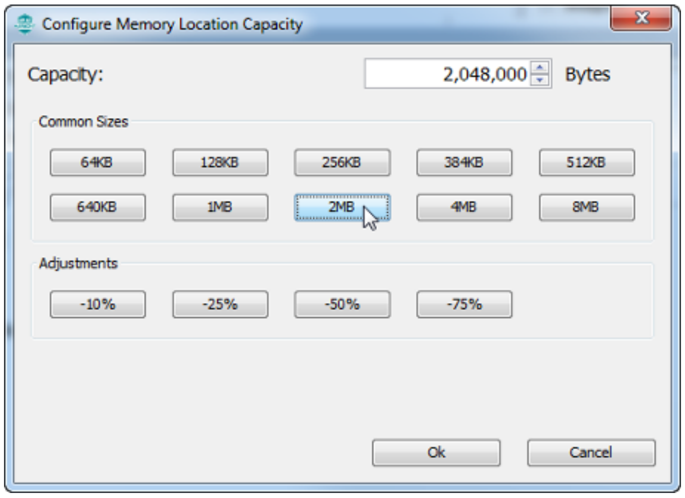
Output File Name: gfx_assets

If your device has 1024 Kbytes (1048576 bytes) of flash, you can assign 40% to asset storage and 60% to code. In that case the “Total Size” in the above sub-tab would be set to 419430 (= 40% of 1048576).

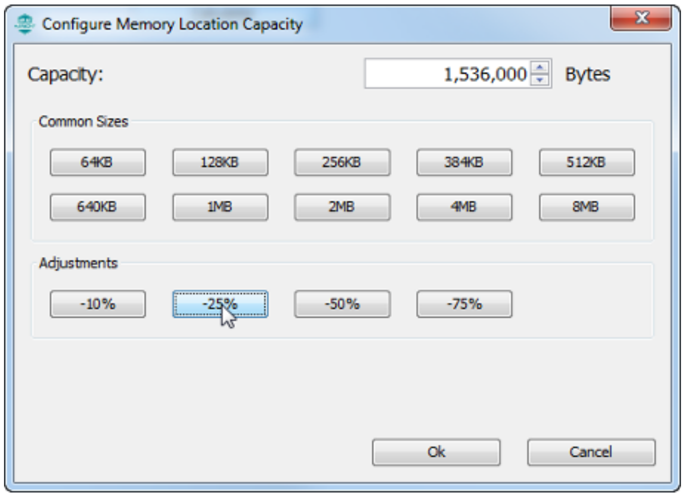
The Calculator button can assist you in allocating internal flash. Click on it and then set the device flash capacity. Then you can apply an adjustment to that value to assign that memory to asset storage.

Example:

If the device has 2 Mbytes of internal Flash, click **2MB**.



Then, to assign 75% of the 2 Mbytes to asset storage, click **-25%** to reduce the 2 MB by 25%, leaving 75%, and then click **OK** to finish. This will then assign 1,536,000 bytes to asset storage.

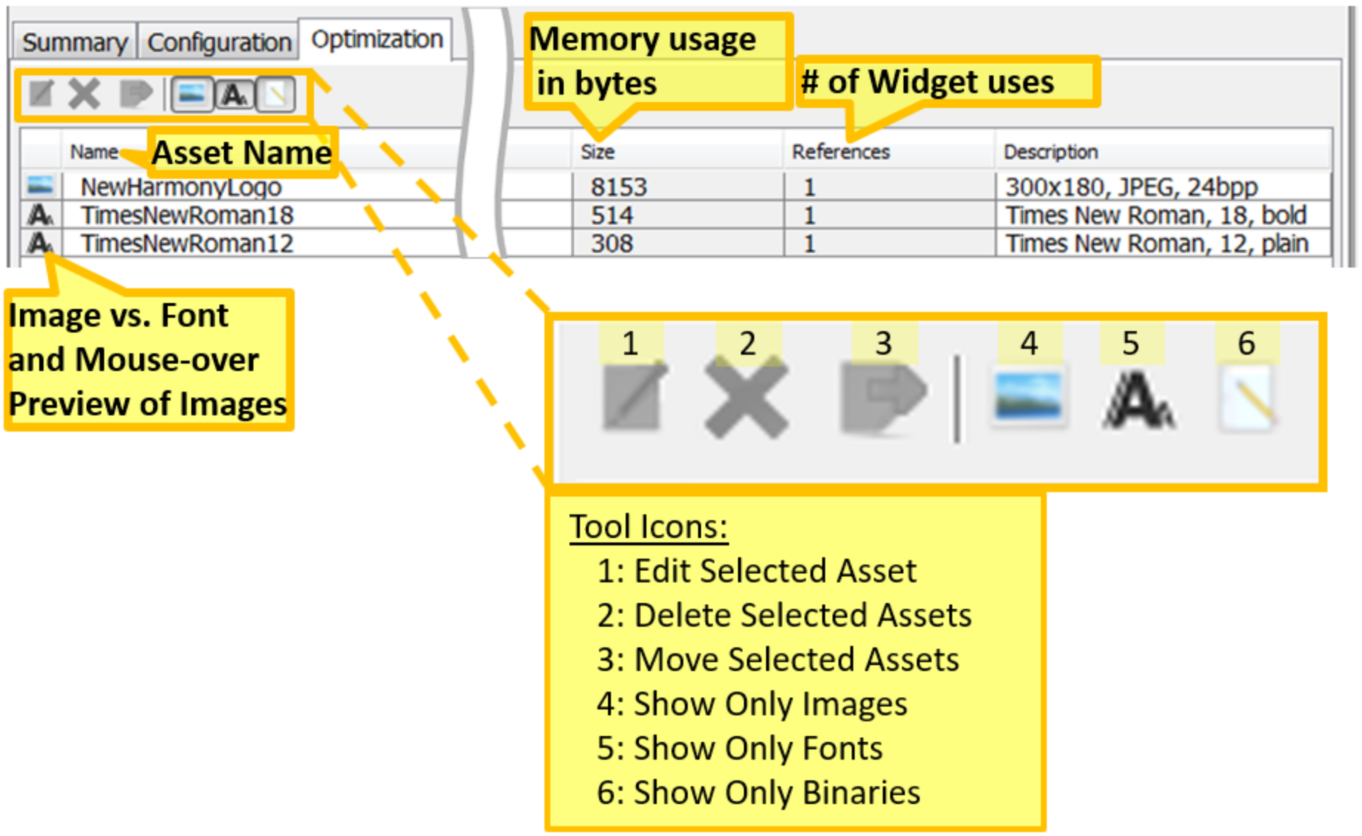


Internal (program) Flash is shared between the application’s code and asset storage. If the application code and graphics assets (fonts, strings, images) won’t fit into the available flash memory then the linker will be unable to build the application and an error will be generated in MPLAB X IDE.

The **Output File Name** must be compatible with the operating system hosting MPLAB X IDE. In most cases the default name (`gfx_asset.c`) will suffice, but this is provided for additional flexibility in building the application.

Optimization Sub-tab

The Optimization sub-tab for the Aria Quickstart demonstration is shown in the following figure.



Name	Size	References	Description
NewHarmonyLogo	8153	1	300x180, JPEG, 24bpp
TimesNewRoman18	514	1	Times New Roman, 18, bold
TimesNewRoman12	308	1	Times New Roman, 12, plain

The **Size** column shows the bytes allocated for storage in internal flash for the images, fonts, and binaries of the application.

The **References** column shows the number of known references for these assets by the application's widgets. A references count of zero suggests that the asset is not used by the application, but it could also mean that the asset is only used in real-time when it is dynamically assigned to a widget by the application. Clicking the title of a column (Name, Size, or References) sorts the lists of graphics assets by that column. Clicking the same column again reverses the sort order.

The window's tools icons support:

1. **Edit Selected Asset** – This brings up the edit dialog for the image, font, or binary chosen
2. **Delete Selected Assets** – Removes the selected assets
3. **Move Selected Assets** – Move assets from one location to another. This is useful for moving assets to/from internal memory from/to external memory.
4. **Show Only Images** – Show image assets toggle on/off
5. **Show Only Fonts** – Show font assets toggle on/off
6. **Show Only Binaries** – Show binary assets toggle on/off

Image Assets

Provides information on the Image Assets features.

Description

The Image Assets window is launched from the Graphics Composer's Asset menu.

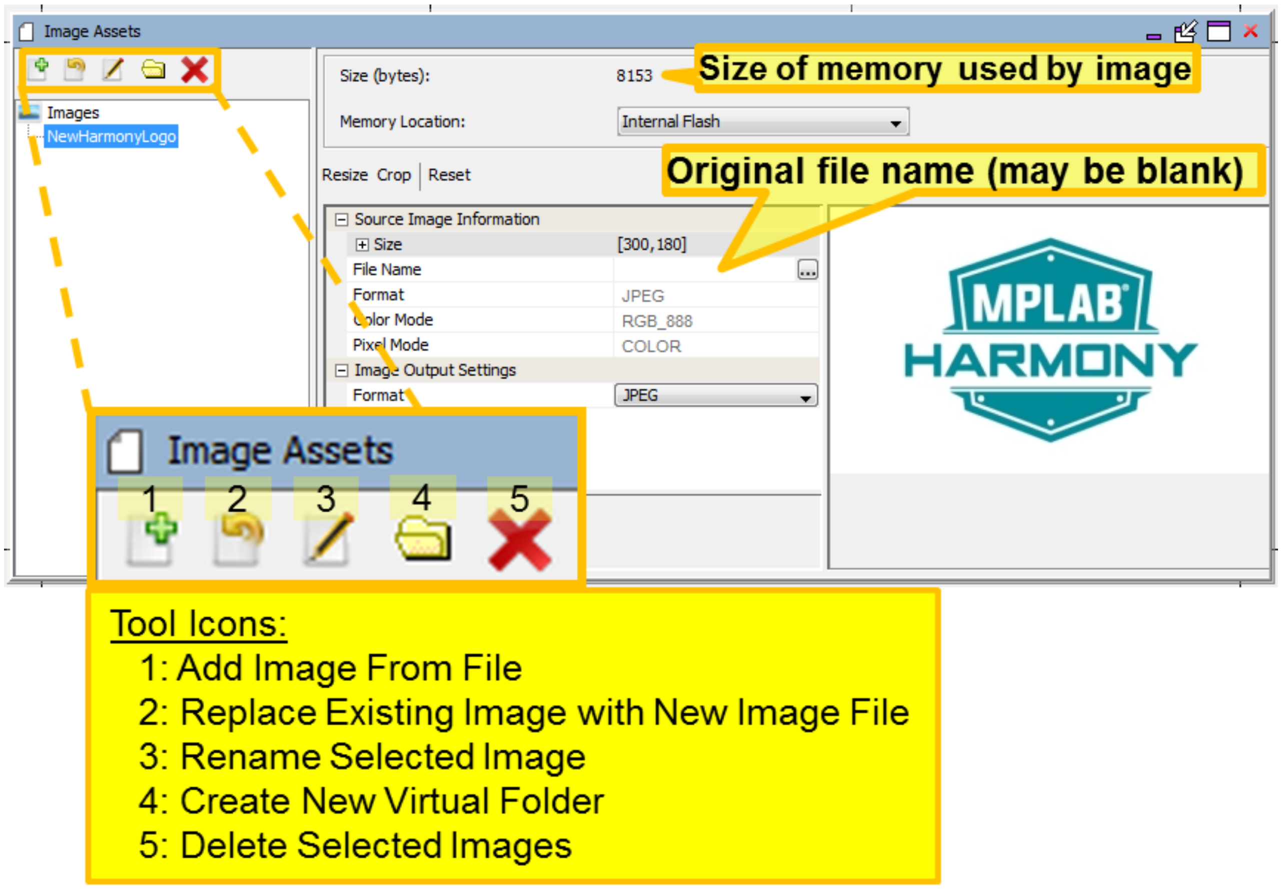
The Image Assets window lets you import images, select different image formats/color modes for each image, select compression methods (for example, RLE) for each image, and displays the memory footprint of each. Images can be imported as a BMP, GIF, JPEG, and PNG (but not TIFF). Images can be stored as Raw (BMP, GIF), JPEG, and PNG.

Note: MHGC does not support image motion that can be found in GIF (`.gif`) files. GIF images are stored in the raw image format, meaning that there is no image header information stored with the image.

When an image is imported into MGHC, the Graphics Asset Converter (GAC) stores the input format and color mode along with any relevant header data. The image's pixel data is then promoted from its native format into a Java Image using 32 bits/pixel (8 bits for each color, RGB, and 8 bits for Alpha Blending). If the image contains Alpha Blending then this information is stored in the "A" of RGBA, otherwise the "A" is set to maximum opacity. When the application is built each image is stored in the image format and color mode selected. Images displayed in the Screen Designer are converted from Java Image format into the format/color mode selected so that the Screen Designer accurately represents what the application will show when running.

The images are decoded on the fly by the graphics library and rendered on the screen. This provides the designer with considerable flexibility to import using one format and store resources using another format, thus exploring and maximizing the best memory utilization for their application and hardware. This supports trading a smaller memory footprint at the cost of additional processing (for static or drawn-once) or reducing processing at the cost of a larger memory footprint (dynamic or drawn many times).

The following figure shows the Image Assets window for the Aria Quickstart demonstration.

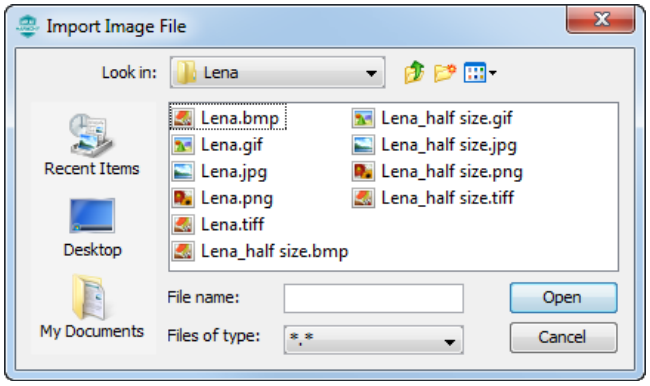


Window Toolbar

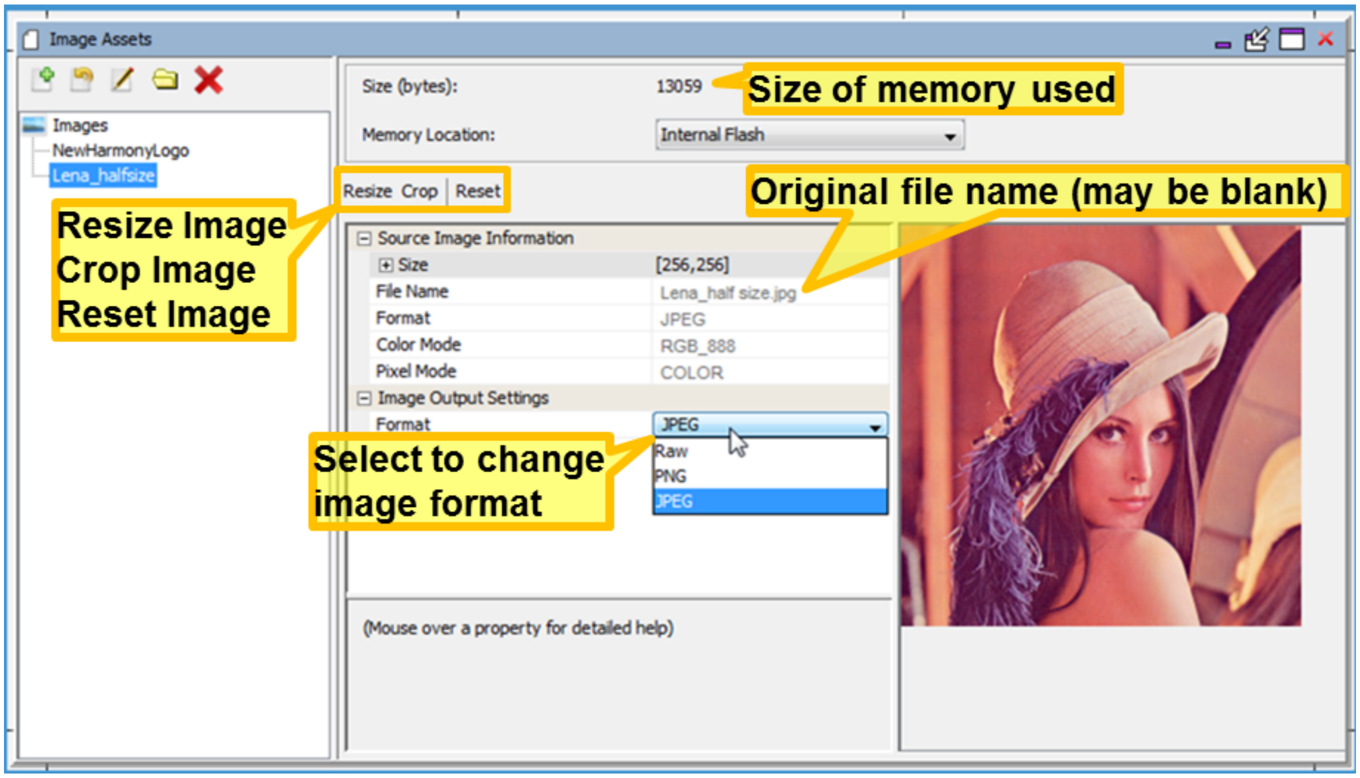
There are five icons on the toolbar below the Images tab:

1. **Add Image Asset** – Brings up "Import Image File" dialog window to select image file to add to the graphics application.
2. **Replace Existing Image with New Image File** – Brings up the same "Import Image File" dialog but instead of creating a new image, the file's content replaces the currently selected image.
3. **Rename Selected Image** – Renames the selected image.
4. **Create New Virtual Folder** – Creates a new virtual folder, allowing you to organize images in a hierarchy.
5. **Delete Selected Images** – removes the selected images from the application.

Selecting the Add Image Asset or Replace Existing Image icon opens the Import Image File dialog that can be used to select and import an image.



After selecting the file and clicking **Open**, the Image Assets window opens.



The size of the memory used for this image based on its color mode, format, compression, and global palette usage is shown by Size (bytes). See Image Format Options below for more details.

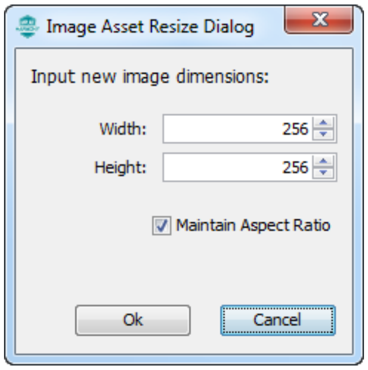
The File Name of the original source file is also shown, but may be blank if the image was imported under MPLAB Harmony v2.03b or earlier. The format and color mode of the stored image can be changed to reduce the image's memory footprint. (If using an LCC controller, you can also turn on the Global Palette, replacing each pixel in the image with just an 8 bit LUT index.)

The three internal image formats are:

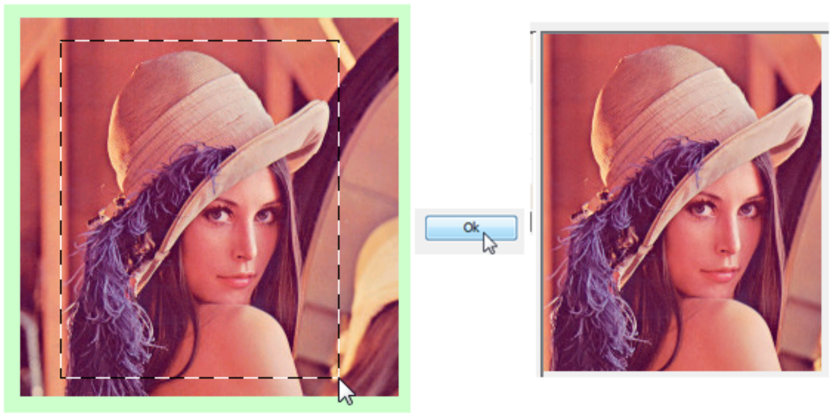
- Raw – binary bit map with no associated header information. GIF and BMP images are imported into this format.
- PNG – lossless image format with compression, 24 bits/pixel (RBG_888) or 32bits/pixel (RGBA_8888). A good choice for line drawings, text, and icons.
- JPEG (JPG) – loss compressed format, uses much less storage than the equivalent bit map (raw). Good for photos and realistic images.

The Image Assets window supports resizing, cropping, or resetting an image:

- **Resize** – Brings up a dialog window to change the pixel dimensions of the image. The image is interpolated from the original pixel array into the new pixel array.



- **Crop** – Places a cropping rectangle on the image. Click and drag a rectangle across the image to select the new image. Then click **Ok** to crop the image.



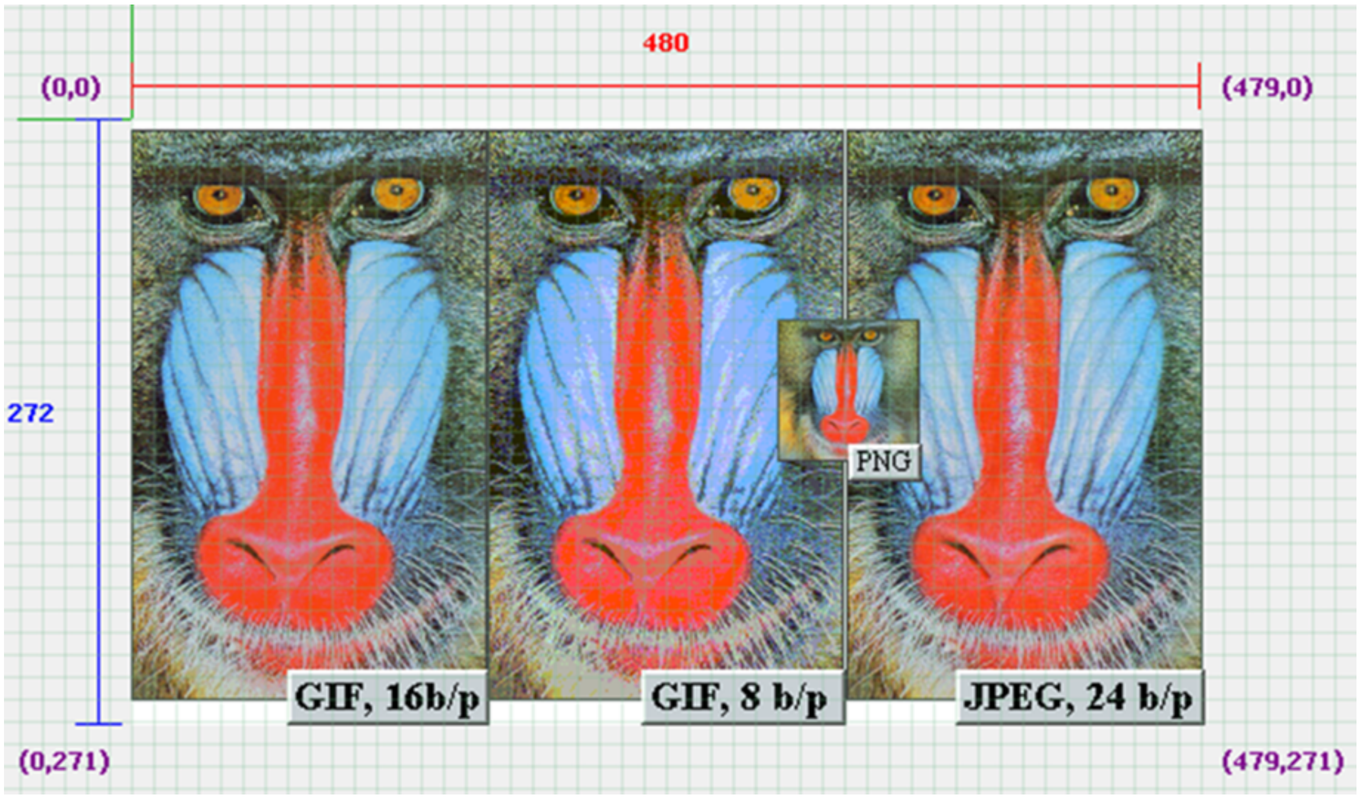
- **Reset** – Allows undoing of a resize or crop. The original image is always stored in the project, so a Reset is always available to return the image to its original state.

Original images are retained by MHGC by the superset Java Image format. So an image crop will change how the image is stored in the application but not how it is stored in MHGC. Reset will always restore the image back to the original pixels. (Reset is not an “undo”.)

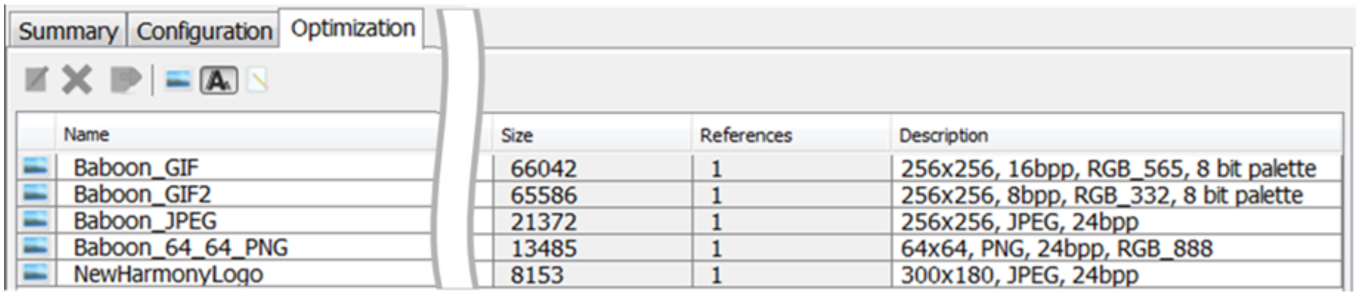
Example Images

Example images are available from many sites on the internet. One of the best sites is found at the USC-SIPI Image Database (http://sipi.usc.edu/database/). There are many canonical test images, such as Lena, The Mandrill (Baboon), and other favorites, all in the TIFF format. The TIFF format is not supported by the Graphics Composer, but you can easily convert from TIFF to BMP, GIF, JPEG, or PNG using the export feature found in the GNU Image Manipulation Program (GIMP), which is available for free download at: https://www.gimp.org. GIMP also allows you to change the pixel size of these images, usually 512x512, to something that will fit on the MEB II display (either 256x256 or smaller).

The following figure shows the Graphics Composer Screen Designer for the pic32mz_da_sk_meb2 configuration of the Aria Quick Start project after adding three images.

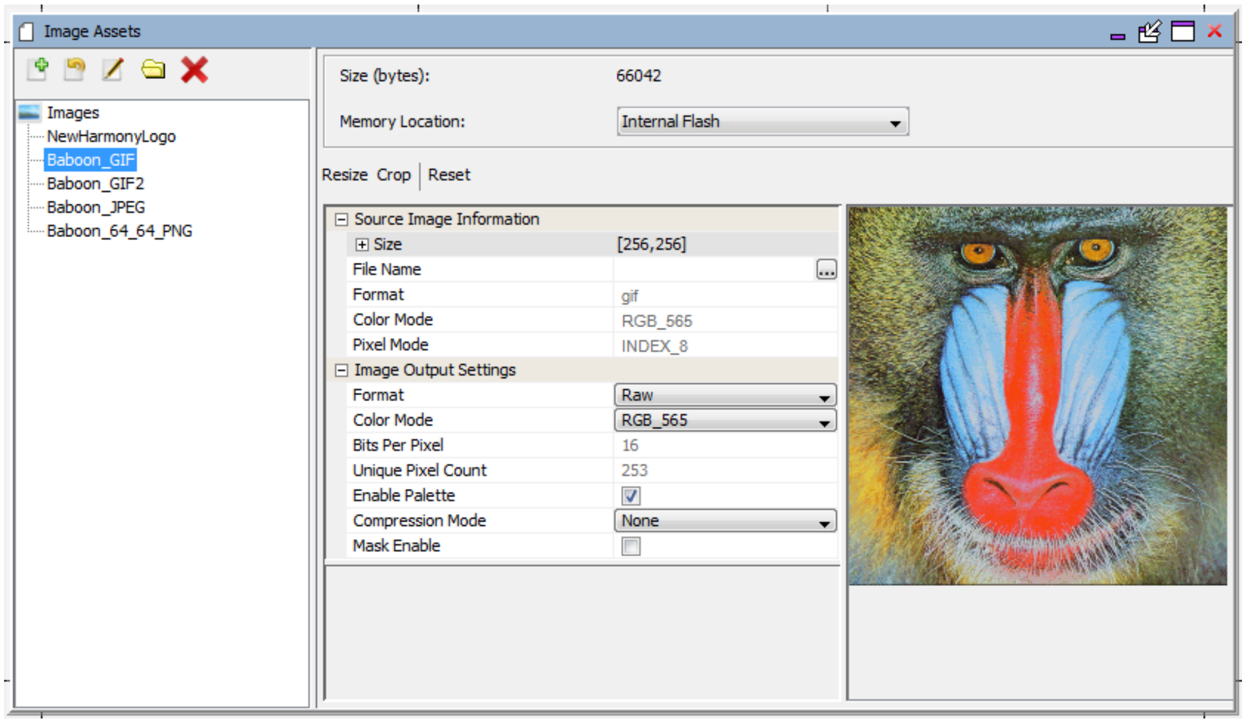


The following figure shows the Optimization Tab after adding these images.



Name	Size	References	Description
Baboon_GIF	66042	1	256x256, 16bpp, RGB_565, 8 bit palette
Baboon_GIF2	65586	1	256x256, 8bpp, RGB_332, 8 bit palette
Baboon_JPEG	21372	1	256x256, JPEG, 24bpp
Baboon_64_64_PNG	13485	1	64x64, PNG, 24bpp, RGB_888
NewHarmonyLogo	8153	1	300x180, JPEG, 24bpp

Selecting the Baboon_GIF image and the Edit Selected Asset icon ( ) opens an Image Assets window, as shown in the following figure.



Size (bytes): 66042

Memory Location: Internal Flash

Source Image Information	
Size	[256,256]
File Name	
Format	gif
Color Mode	RGB_565
Pixel Mode	INDEX_8
Image Output Settings	
Format	Raw
Color Mode	RGB_565
Bits Per Pixel	16
Unique Pixel Count	253
Enable Palette	✓
Compression Mode	None
Mask Enable	

Because this image had only 253 unique pixel colors (Unique Pixel Count = 253) the Enable Palette option was automatically enabled. This feature, which works on an image by image basis, is separate from enabling a Global Palette. The image is stored using 8 bits of indexing into an image-specific lookup table (LUT). If the image has more than 256 unique colors then the Enable Palette option is not available and is not shown.

Image Format Options

Raw Format Images

Raw format images have the following options:

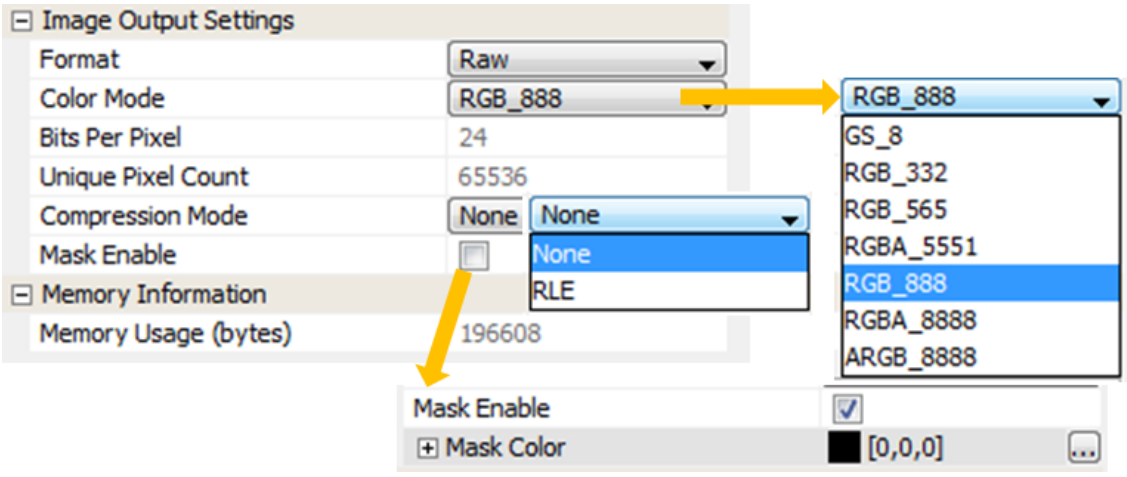


Image Output Settings	
Format	Raw
Color Mode	RGB_888
Bits Per Pixel	24
Unique Pixel Count	65536
Compression Mode	None
Mask Enable	
Memory Information	
Memory Usage (bytes)	196608

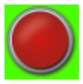
Mask Enable	✓
Mask Color	[0,0,0]

Regardless of the Color Mode of the imported the image, the stored image can be stored in a different color mode. For example, a JPEG image could be in 24 bits/pixel RGB format but stored in the application using RGB_565 or even RBG_332 to save space. The Project Color Mode (set through the *File* > *Settings* menu) is different from the Color Mode of images. This is determined by the capabilities of the projects graphics controller. The graphics library converts images from the stored color mode to the project's color mode before output.

If the image has 256 or less unique pixel colors an option to Enable Palette is set by default. If the image has more than 256 unique colors this option is not displayed. This replaces the palette pixels with 8-bit indices into the image's palette look up table (LUT). NOTE: Enabling the Global Palette disables this for all images and all image pixels are replaced by 8-bit indices into the global palette LUT.

The Compression Mode for a raw format image is either None (no compression) or RLE for run-length encoding.

Image masking is a form of cheap blending. For example, given the following image, you may want to show the image without having to match the lime green background. With image masking you can specify that the lime green color as the "mask color", causing it to be ignored when drawing this image. The rasterizer will simply match a pixel to be drawn with the mask. If they match, the pixel is not rendered.



PNG Format Images

For PNG format images you can change the image format and the image color mode:

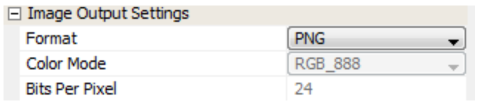


Image Output Settings	
Format	PNG
Color Mode	RGB_888
Bits Per Pixel	24

JPEG Format Images

For JPEG format images you can change from JPEG format to Raw or PNG:

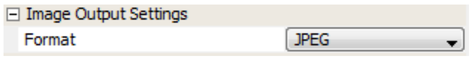


Image Output Settings	
Format	JPEG

Once changed from JPEG into another format, the new format will have other options.

Managing Complex Designs

The Image assets tool lists the images in the order of their creation. In a future version of MPLAB Harmony this will be sortable by image name. For now, it is recommended that you use the Memory Locations asset tool, and use the Optimization sub-tab instead to manage a complex set of images. The Optimization sub-tab allows you to sort graphics assets (fonts, images, binaries) by Name, Size, and number of widget References. This makes it much easier to find and edit an image by its name rather than order of creation.

Font Assets

Provides information on the Font Assets features.

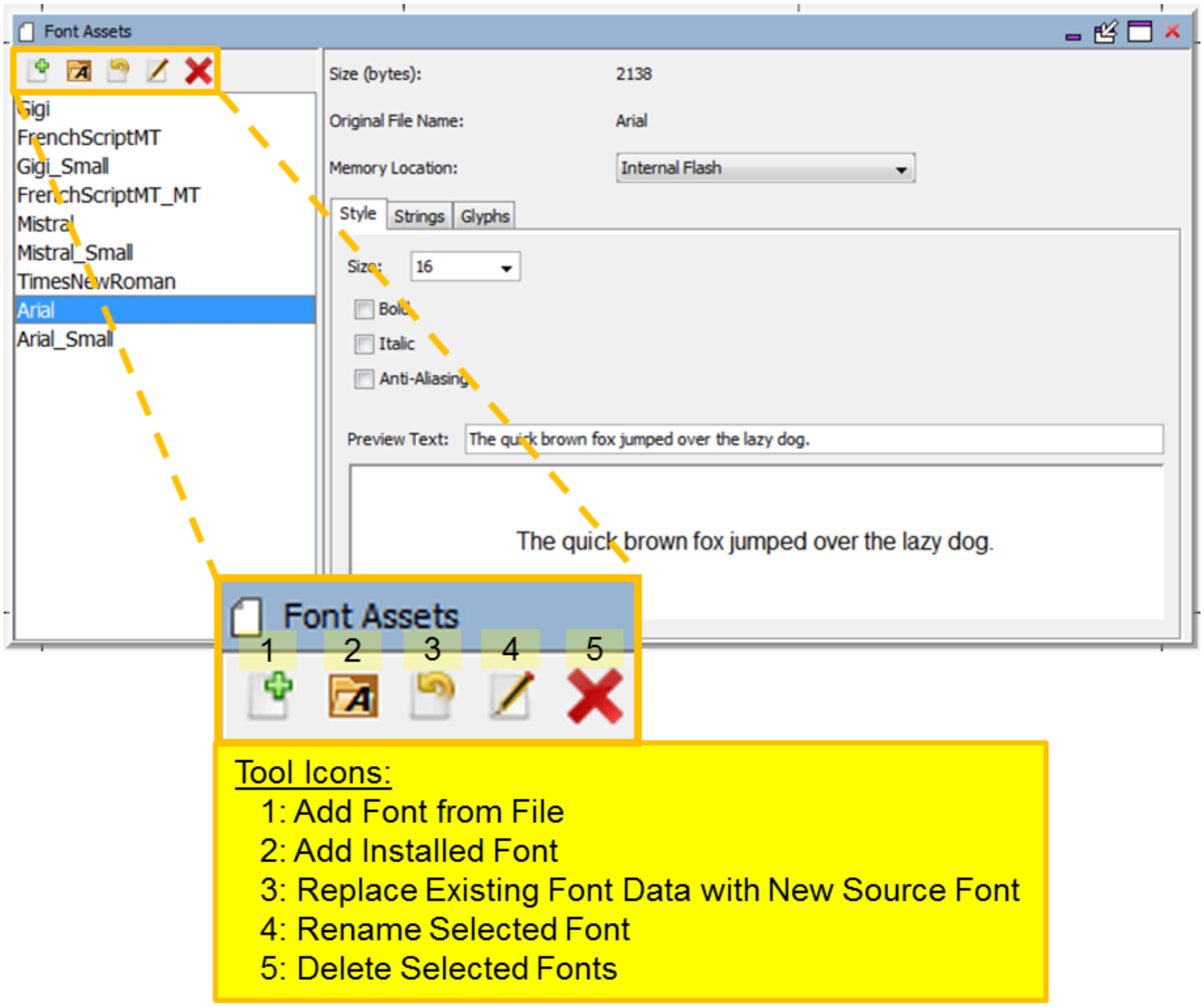
Description

The Font Assets window is launched from the Graphics Composer's Asset menu.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language "ID" strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string's text and the font is chosen for that text.

- Languages are managed within the String Table Configuration window
- Fonts are managed within the Font Assets window (this topic)
- Strings are managed within the String Assets window

The following figure shows the Font Assets window from the Aria Coffee Maker demonstration.



Size (bytes): 2138

Original File Name: Arial

Memory Location: Internal Flash

Size: 16

Preview Text: The quick brown fox jumped over the lazy dog.

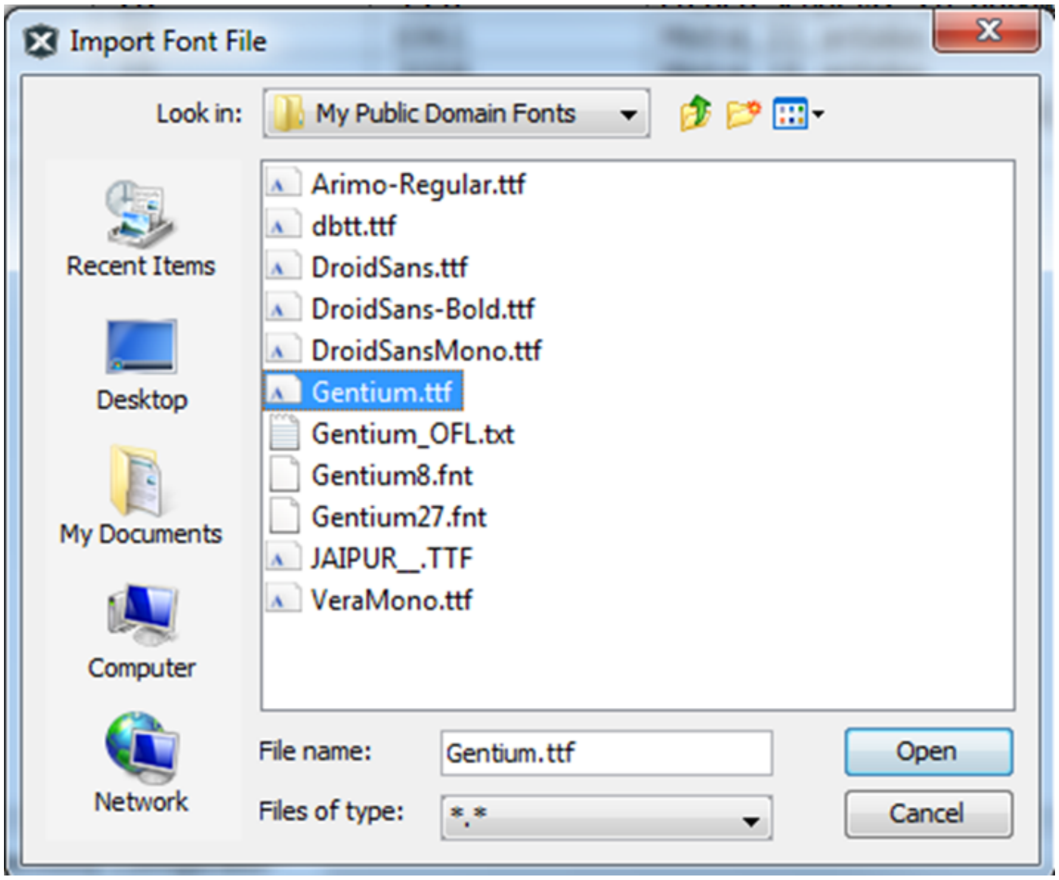
Tool Icons:
1: Add Font from File
2: Add Installed Font
3: Replace Existing Font Data with New Source Font
4: Rename Selected Font
5: Delete Selected Fonts

The Size (bytes): for a Font asset shows how much memory is needed to store all the glyphs used by the application from this font. For static strings MHGC determines which glyphs are used by the application's pre-defined strings and builds these glyphs into the application. For dynamic strings (i.e. strings created during run time) ranges of glyphs are selected by the designer and these ranges are also included in the application by MHGC. The memory needed to store all these glyphs is shown by Size (bytes): .

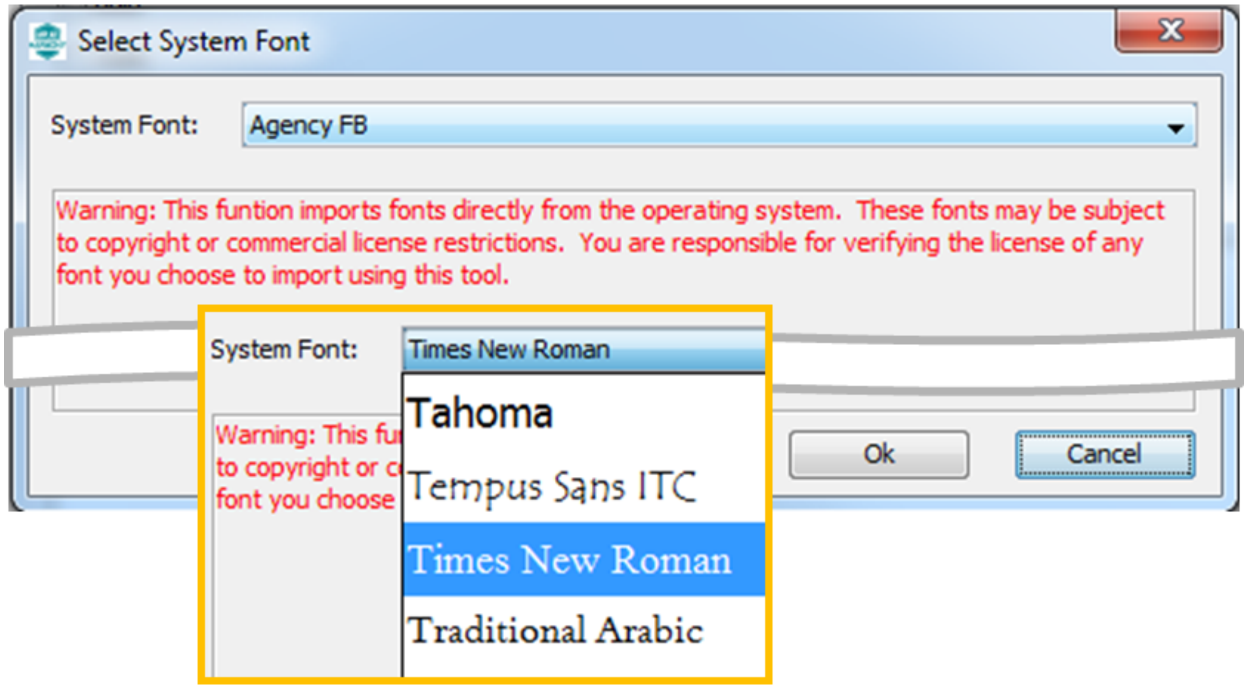
Window Toolbar

There are five icons on the toolbar below the Images tab:

1. **Add Font From File** – Adds a font asset from a file.



2. **Add Installed Font** – Add a font installed on your computer.



Warning: This funtion imports fonts directly from the operating system. These fonts may be subject to copyright or commercial license restrictions. You are responsible for verifying the license of any font you choose to import using this tool.

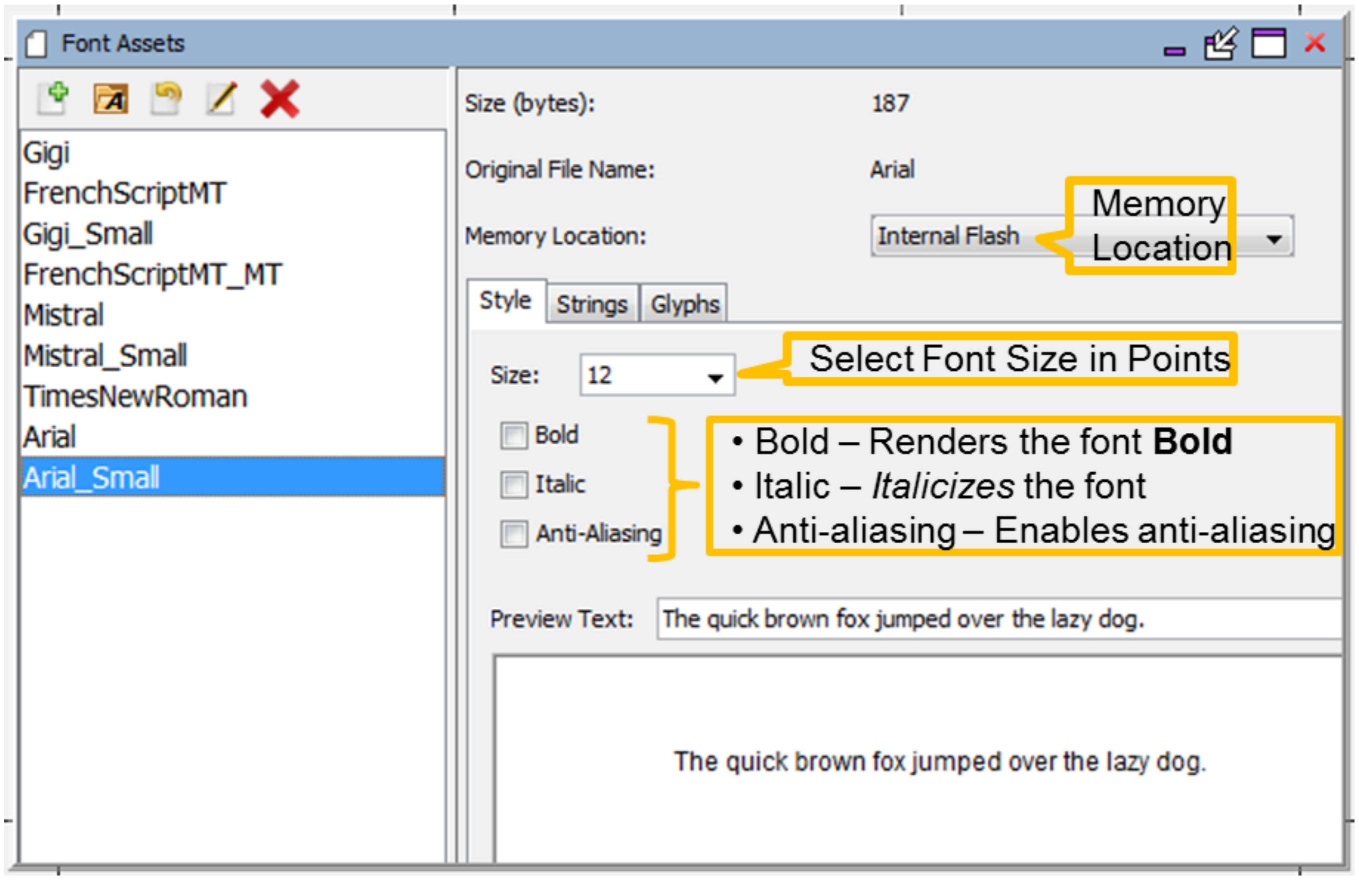
3. **Replace Existing Font Data with New Source Font** – Both Add Font From File and Add Installed Font create a new font asset. This icon allows you to update an existing font asset, importing from a file or using a font installed on your computer.
4. **Rename Selected Font** – Renames an existing font asset. In the example above, the Arial font was installed twice, first as a 16 point font and second as a 12 point font. If added to the fonts assets in this order, the 12 point font will have the name Arial_1. This font asset was renamed to Arial_Small using this tool.

5. **Delete Selected Fonts** – Removes selected font assets from the application.

Sub-tabs

There are three sub-tabs to this window.

Style Sub-tab



Size (bytes): 187

Original File Name: Arial

Memory Location: Internal Flash

Size: 12

Bold

Italic

Anti-Aliasing

• Bold – Renders the font **Bold**
• Italic – *Italicizes* the font
• Anti-aliasing – Enables anti-aliasing

Preview Text: The quick brown fox jumped over the lazy dog.

The quick brown fox jumped over the lazy dog.

The Size (bytes): shown represents the memory needed to store all the font's glyphs. The application only stores the glyphs that are used by static (build-time) strings and by predefined glyph ranges to support dynamic (run-time) strings.

The choices for Memory Location must be defined before the font can be assigned. Go to the Memory Configuration window to add a new location before using it in this sub-tab.

Each font asset consists of a font, size, and some combination of the { Bold, Italic, Anti-Aliasing } options, including selecting none of these options. If you need bold for one set of strings and italic for another, then you will need two font assets, one with Bold checked and a second with Italic checked. The same applies for font sizes. Each font size requires its own font asset. Thus if you need two sizes of Arial, with plain, bold, and italic for each size, you will need 6 separate assets (6 = 2 Sizes x 3).

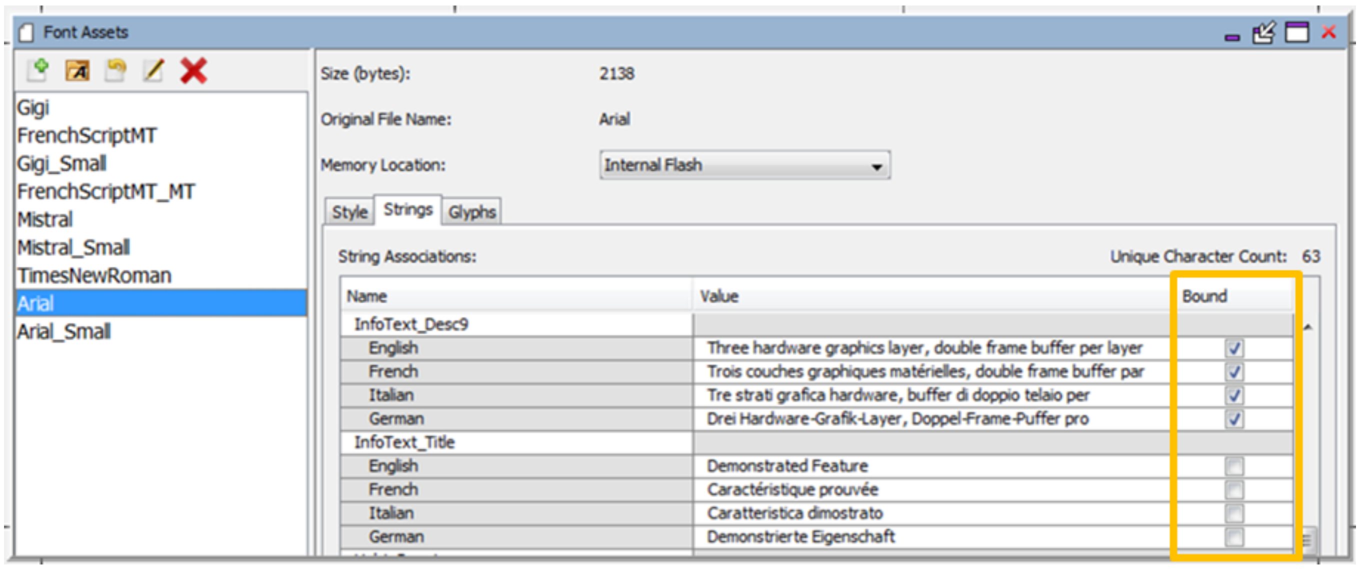
Glyphs are normally (Anti-Aliasing off) stored as a pixel bit array, with each pixel represented by only one bit. Turning on Anti-Aliasing replaces each pixel bit with an 8-bit gray scale, thereby increasing font storage by a factor of 8!

What if a font is chosen that does not support the character types of the text used for a particular language in the application? How can you test and debug this? There a basically two ways:

- Use an external font viewer to examine if the needed glyphs exist
- Configure, build, and run the application and verify the strings are correctly rendered

If the glyphs are not available they will be rendered as rectangles (☐).

Strings Sub-tab



Gigi
FrenchScriptMT
Gigi_Small
FrenchScriptMT_MT
Mistral
Mistral_Small
TimesNewRoman
Arial
Arial_Small

Size (bytes): 2138
Original File Name: Arial
Memory Location: Internal Flash

String Associations: Unique Character Count: 63

Name	Value	Bound
InfoText_Desc9		
English	Three hardware graphics layer, double frame buffer per layer	✓
French	Trois couches graphiques matérielles, double frame buffer par	✓
Italian	Tre strati grafica hardware, buffer di doppio telaio per	✓
German	Drei Hardware-Grafik-Layer, Doppel-Frame-Puffer pro	✓
InfoText_Title		
English	Demonstrated Feature	
French	Caractéristique prouvée	
Italian	Caratteristica dimostrato	
German	Demonstrierte Eigenschaft	

The Bound check box accomplishes the same thing as assigning a font to a text string in the Strings Assets window (Window:Strings menu). Assigning a string to a font means that the font will generate glyphs for that string. This is just another way to accomplish the binding of the string text to font.

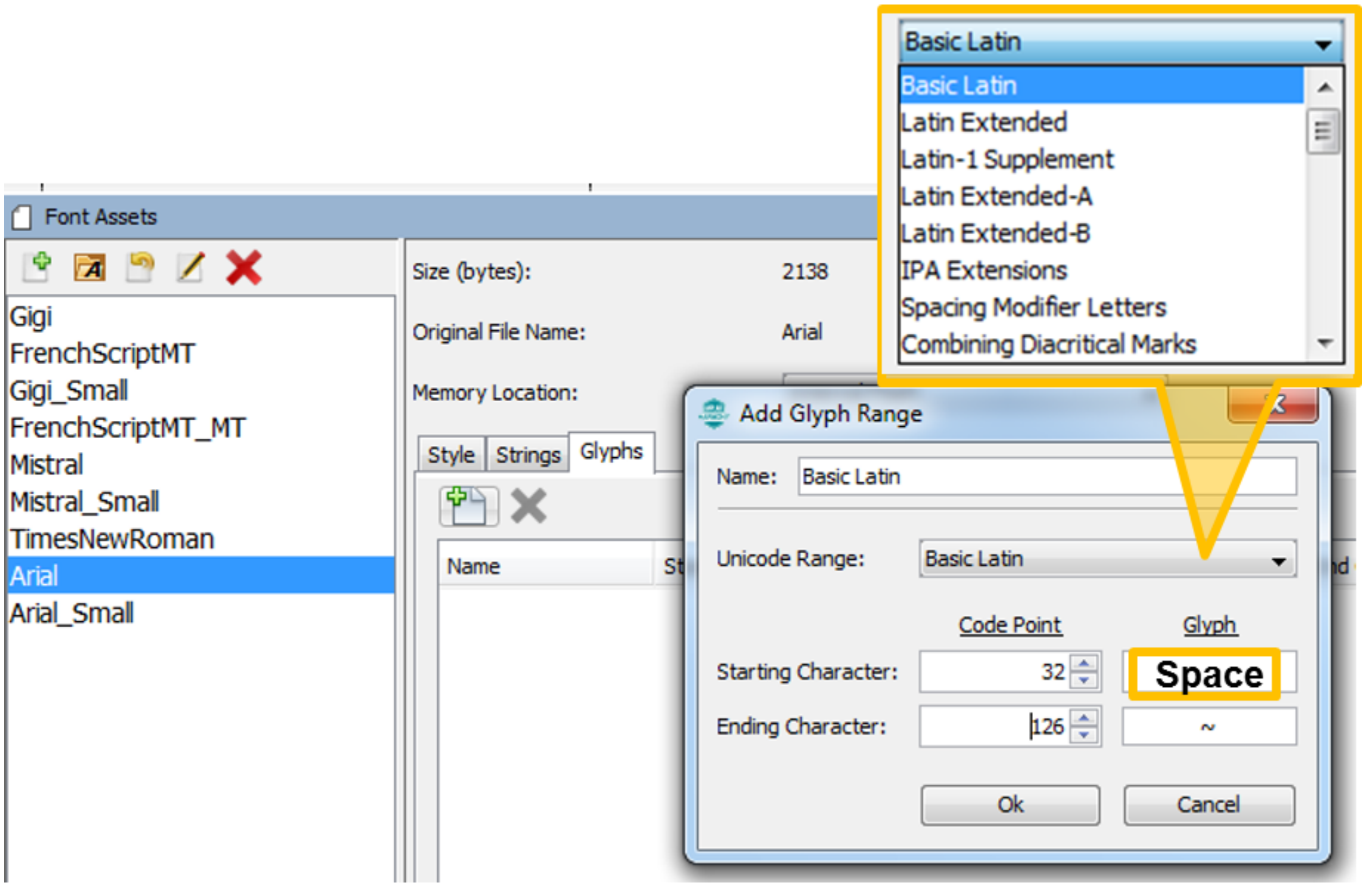
This sub-tab is also useful in a complicated graphics design to see how many strings use a particular font. Lightly-used or unused fonts can be eliminated to free up internal Flash memory.

Glyphs Sub-tab

Note: The word “glyph” comes from the Greek for “carving”, as seen in the word hieroglyph – Greek for “sacred writing”. In modern usage a glyph is an elemental or atomic symbol representing a readable character for purposes of communicating via writing.

The Glyph sub-tab is only used when your application supports dynamic strings. For static (build-time) strings MHGC automatically determines which font glyphs are used based on the characters present in all the strings used by the application’s graphics widgets. Only these glyphs are included as part of the application’s font assets. With dynamic (i.e. run-time) strings this is not possible. This sub-tab allows you to specify which range of glyphs will be used by run-time strings. Once glyph ranges are defined, these glyphs are added to the font glyphs used by static strings.

The Create New Custom Import Range icon ( ) allows you to input a new glyph range for the font. Selecting this icon opens the Font Assets window.



String Table Configuration

Provides information on the String Assets features.

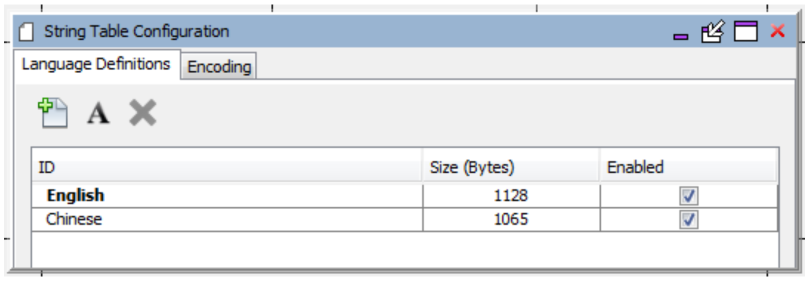
Description

The String Table Configuration window is launched from the Graphics Composer's Asset menu.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language "ID" strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string's text and the font is chosen for that text.

- Languages are managed within the String Table Configuration window (this topic)
- Fonts are managed within the Font Assets window
- Strings are managed within the String Assets window

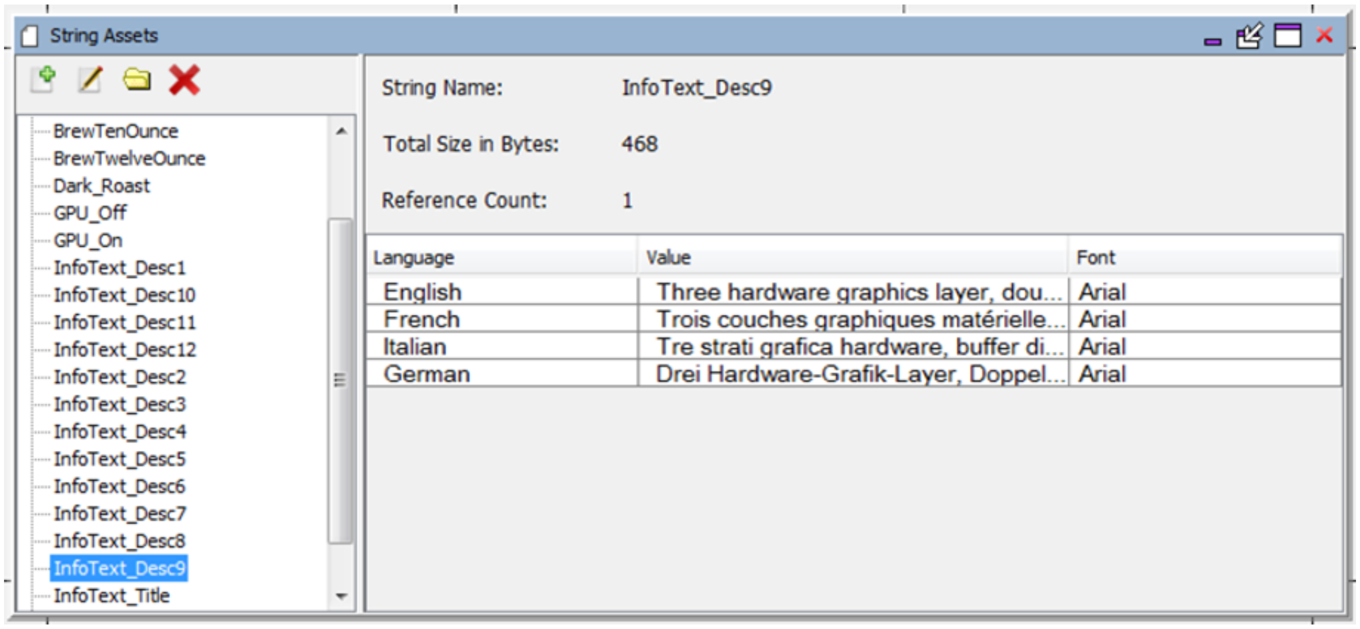
Within this window, the Languages supported by the application are defined and the encoding for all application glyphs selected.



ID	Size (Bytes)	Enabled
English	1128	✓
Chinese	1065	✓

The "ID" string used for each language is merely for ease of use in building the texts to be used. "English", "American", or any other string can be used to identity that language, as long as it is understood by the application's creator when selecting the text to be used for that particular language. Then the application can switch to supporting one of its languages using "ID" strings defined.

Here is an example string asset definition, taken from the Aria Coffee Maker demonstration. This application supports English, French, Italian, and German. The text string "InfoText_Desc9" uses the Arial font, and text for each language is specified within the String Assets window.



String Name: InfoText_Desc9

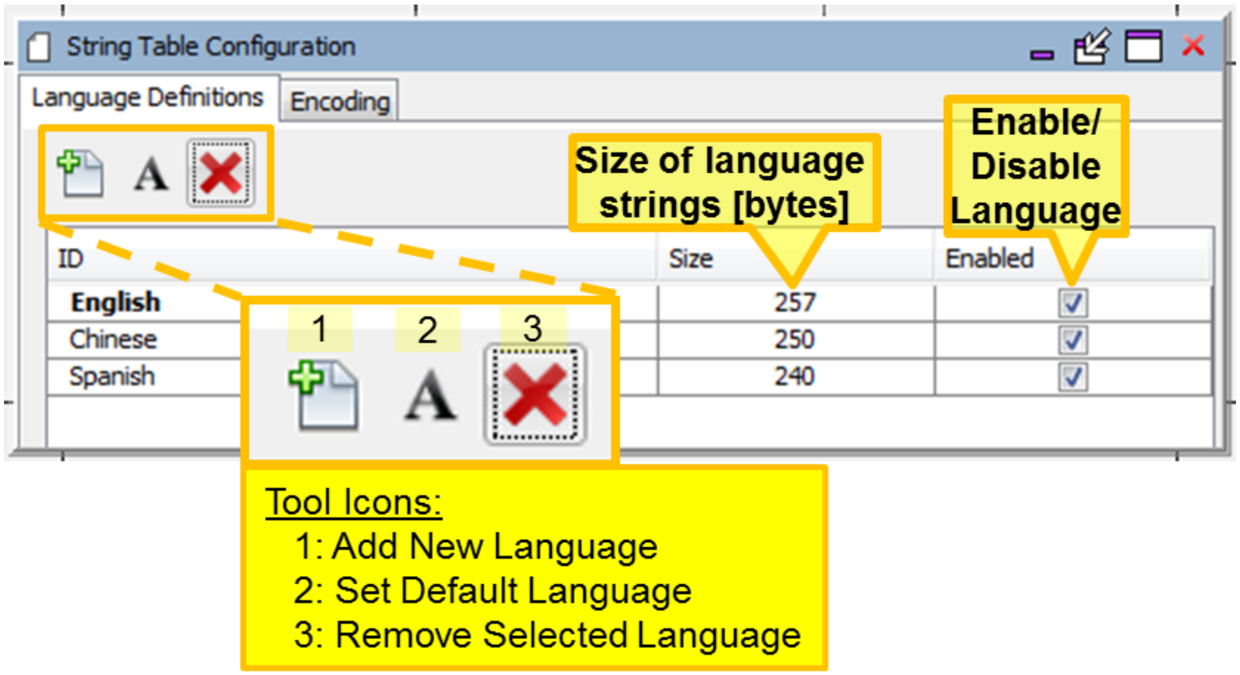
Total Size in Bytes: 468

Reference Count: 1

Language	Value	Font
English	Three hardware graphics layer, dou...	Arial
French	Trois couches graphiques matérielle...	Arial
Italian	Tre strati grafica hardware, buffer di...	Arial
German	Drei Hardware-Grafik-Layer, Doppel...	Arial

Any number of languages can be defined as long as there is memory to store the strings needed.

The following figure shows the String Table Configuration for an application that uses English, Spanish, and Chinese.



ID	Size	Enabled
English	257	✓
Chinese	250	✓
Spanish	240	✓

Tool Icons:
1: Add New Language
2: Set Default Language
3: Remove Selected Language

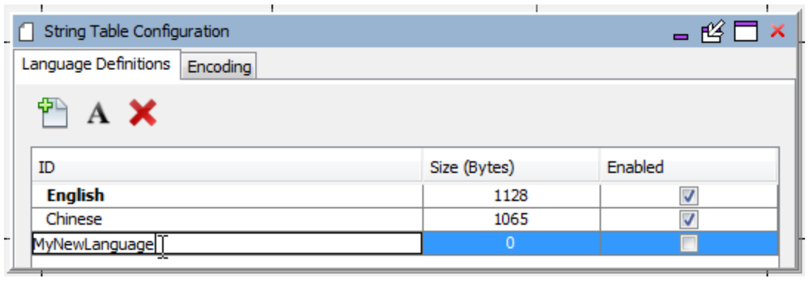
The size of all the strings for each language is shown in the Size column. String size represents the memory allocated for glyph indices for all the strings supporting that language. A language can be enabled/disabled via the check box in the Enabled column. Disabling a language removes it from the application build but keeps it in the project.

Window Toolbar

There are three icons on the toolbar:

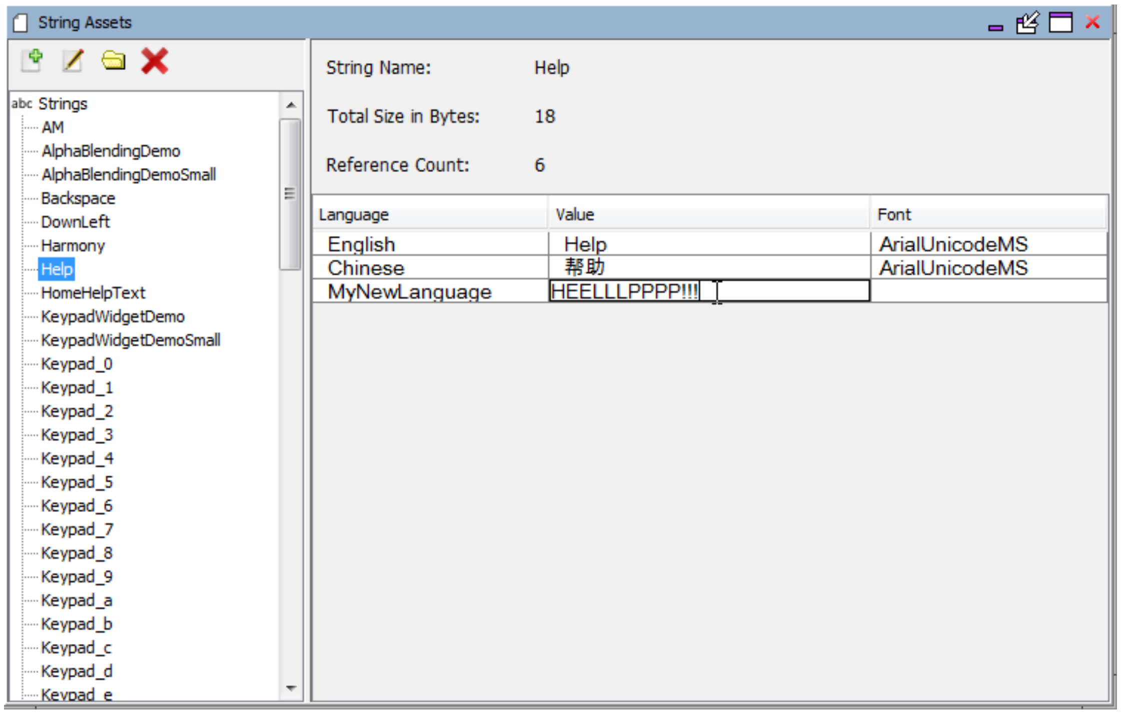
1. **Add New Language** – Adds a new Language.
2. **Set Default Language** – Sets the application's default language. Note, this is different than the abc tool on the Graphics Composer Window toolbar. The abc icon sets the preview language for the Screen Designer panel only. This icon sets the language used by the application after boot-up.
3. **Remove Selected Language** – Removes language from the application.

Clicking **Add New Language** opens a new line, allowing you to select and edit the new language's "ID" string.



ID	Size (Bytes)	Enabled
English	1128	✓
Chinese	1065	✓
MyNewLanguage	0	

Then, for every string defined in the application there will be a line to define the needed text, and to specify the font to be used.

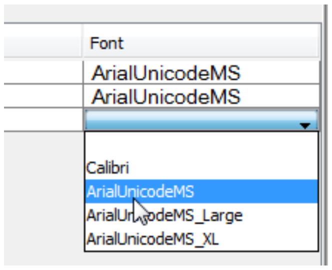


String Name: Help

Total Size in Bytes: 18

Reference Count: 6

Language	Value	Font
English	Help	ArialUnicodeMS
Chinese	帮助	ArialUnicodeMS
MyNewLanguage	HEELLLPPPP!!!	



If you don't provide a value for the new language the string will be output as a null (empty string). If you don't provide a Font selection then the string will be output as a series of blocks (?).

The Aria User Interface Library primitive, `LIB_EXPORT void laContext_SetStringLanguage(uint32_t id)`, allows the application to switch between languages using the Language ID `#defines` are specified in the application's `gfx_assets.h` file.

Sub-tabs

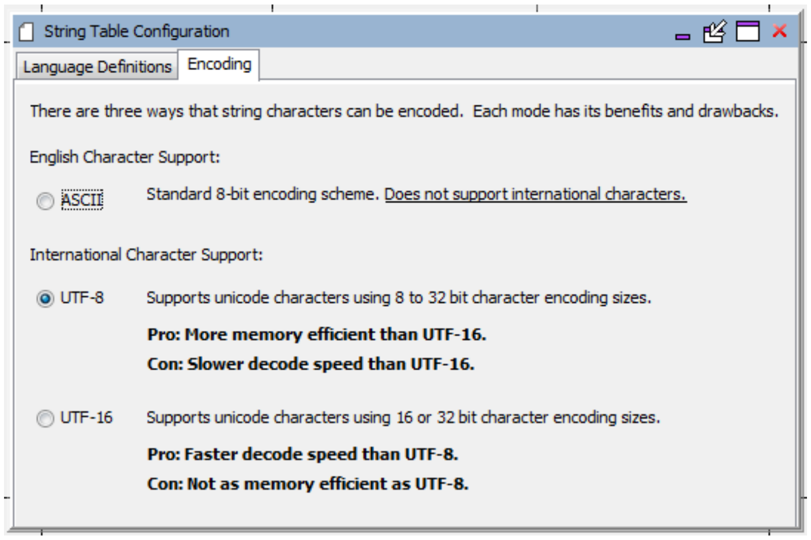
There are two sub-tabs to this window.

Language Definitions Sub-tab

This sub-tab shows the languages defined for the application. A Language can be enabled/disabled to include or exclude it from the application's generation/regeneration under MPLAB Harmony Configurator (MHC). New languages can be added by specifying a text string for the language. With a new language, go to the String Assets window to specify the text and fonts for all defined strings.

Encoding Sub-tab

Selecting the Character Encoding Format Selection Dialog icon gives you three choices for how the characters in all strings in the graphics application are encoded:



There are three ways that string characters can be encoded. Each mode has its benefits and drawbacks.

English Character Support:

ASCII Standard 8-bit encoding scheme. Does not support international characters.

International Character Support:

[✓ UTF-8] Supports unicode characters using 8 to 32 bit character encoding sizes.

Pro: More memory efficient than UTF-16.

Con: Slower decode speed than UTF-16.

UTF-16 Supports unicode characters using 16 or 32 bit character encoding sizes.

Pro: Faster decode speed than UTF-8.

Con: Not as memory efficient as UTF-8.

The default is ASCII. It is typically the most efficient in terms of memory and processing, but it does not support as many glyphs. Chinese text should be encoded in UTF-8 or UTF-16, but Western language text can be encoded in ASCII to save memory. The trade-off between ASCII, UTF-8, and UTF-16 depends on the application. Changing from UTF-8 to UTF-16 will double the size of all strings in the application. This is because the sizes of all glyph indices double in size. (String sizes are the sizes of glyph reference indices, not the size of the particular font glyphs used to write out the string.)

The memory utilization resulting from an encoding choice can be seen in the Summary sub-tab of the Memory Configuration window.

String Assets

Provides information on the String Assets features.

Description

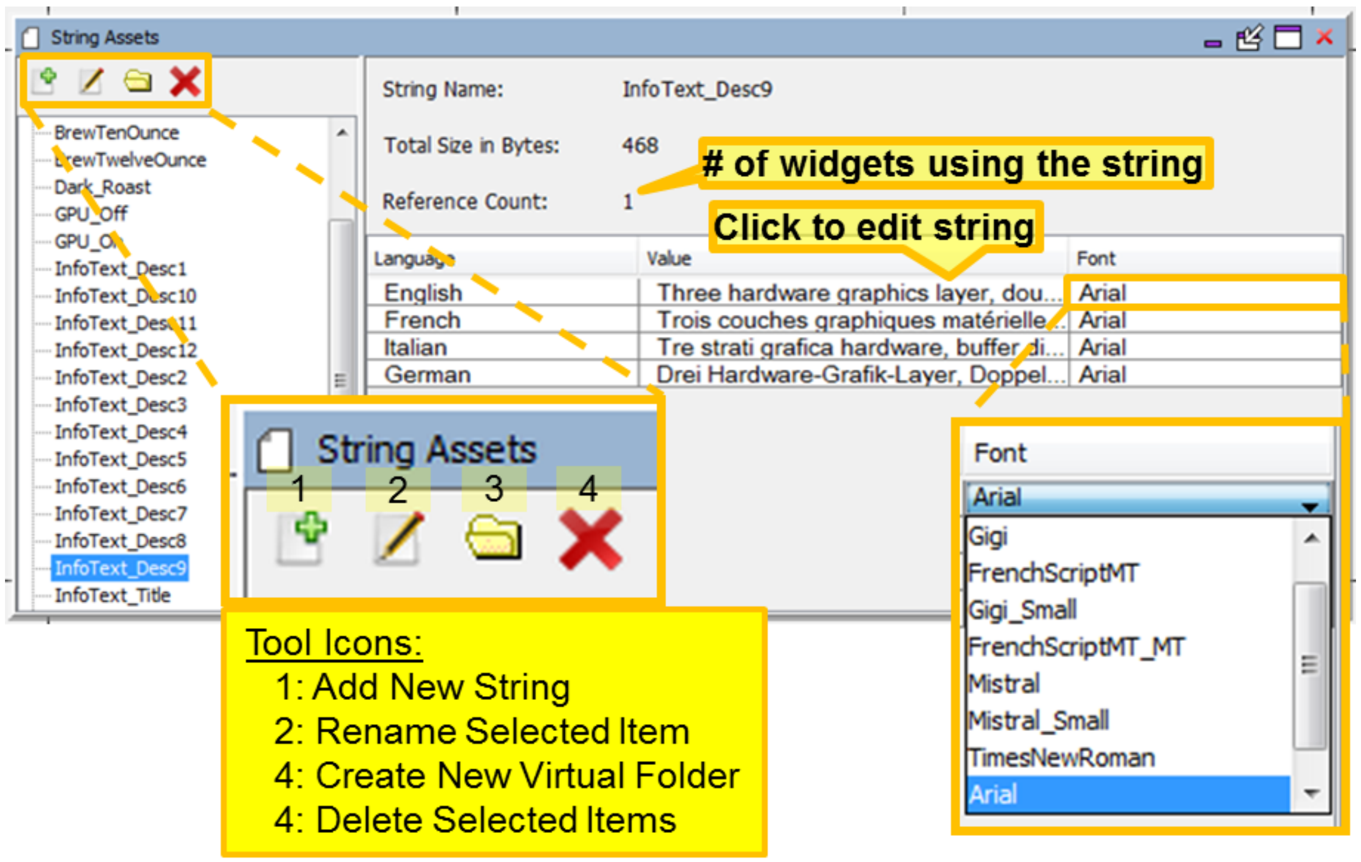
The String Assets window is launched from the Graphics Composer’s Asset menu.

The String Assets window supports managing the strings in the application. Strings are referenced by graphic widgets using an application-wide unique name. This unique name is built into an enumeration that the application’s C code uses. For each language supported text is defined and a font asset selected.

Note: There are three dimensions to text support: Languages, Fonts, and Strings. Language “ID” strings are identified when an application supports more than one language. (In the case of single language support, the language default is provided.) Fonts are imported and organized using the Font Assets window. Strings are defined by a string name, and this name is used by widgets to reference the string. For each string and each language supported the glyphs are defined to spell out the string’s text and the font is chosen for that text.

- Languages are managed within the String Table Configuration window
- Fonts are managed within the Font Assets window
- Strings are managed within the String Assets window (this topic)

The following figure shows an example taken from the Aria Coffee Maker demonstration. The string name, InfoText_Desc9, defines a string asset that is used by the application.



Language	Value	Font
English	Three hardware graphics layer, dou...	Arial
French	Trois couches graphiques matérielle...	Arial
Italian	Tre strati grafica hardware, buffer di...	Arial
German	Drei Hardware-Grafik-Layer, Doppel...	Arial

Tool Icons:
1: Add New String
2: Rename Selected Item
4: Create New Virtual Folder
4: Delete Selected Items

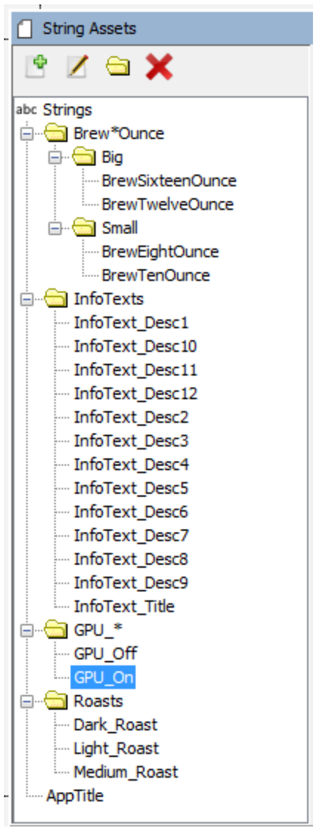
The Total Size in Byte: for a string asset represents the memory needed to store the glyph indices for all the text defined for that string asset. Adding more text will increase the number of glyph indices needed thus increasing the size of the string's memory. Adding another language will do the same, since the number of glyph indices also increases. Changing the font does not increase the size of the string's memory, but may increase the size of the font chosen if it is a "bigger" font and adds more glyphs to the new font. (By "bigger" we mean a font with more pixels, for example because it is bigger in size, or perhaps because it is anti-aliased and the original font was not.)

Note: The Reference Count shown reflects the number of build-time references to the string. Dynamic uses of a string, such as through macros or events, is not reflected in this number.

Window Toolbar

There are four icons on the toolbar:

1. **Add New String** – Adds a new string.
2. **Rename Selected Item** – Allows renaming the string.
3. **Create New Virtual Folder** – Creates a new virtual folder, allowing you to organize strings in a hierarchy. Here's an example reorganization of the existing strings. Note the order of virtual folders or items in the list is strictly alphabetical. Virtual folders and string asset organization is merely for the convenience of the developer. Neither has an effect on how the application is built.

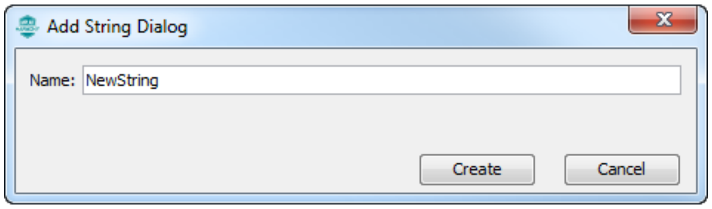


4. **Delete Selected Items** – Deletes selected strings from the application.

Creating New Strings

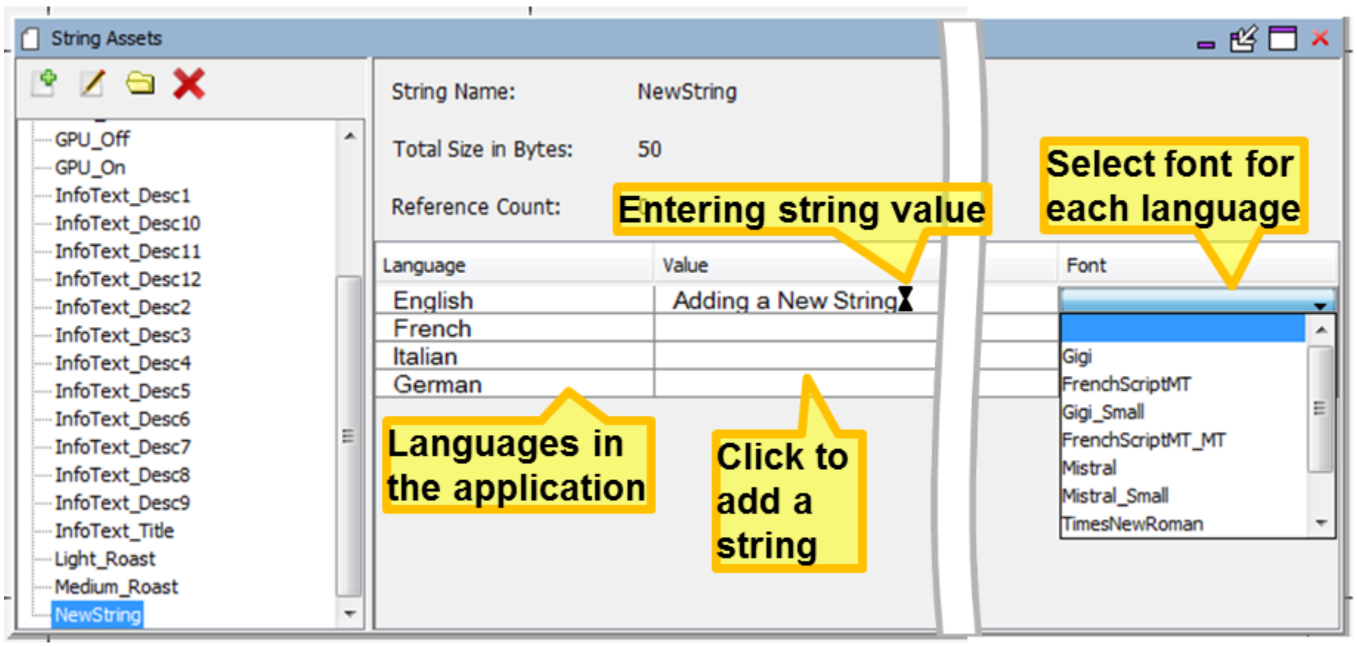
To create a new string, click Add New String ( ).

Selecting this icon opens the Add String dialog to name the string. The text chosen for the string name should be acceptable as a C variable.


Add String Dialog

Name: NewString

After entering the new string's name and click Create, the following String Assets window appears.



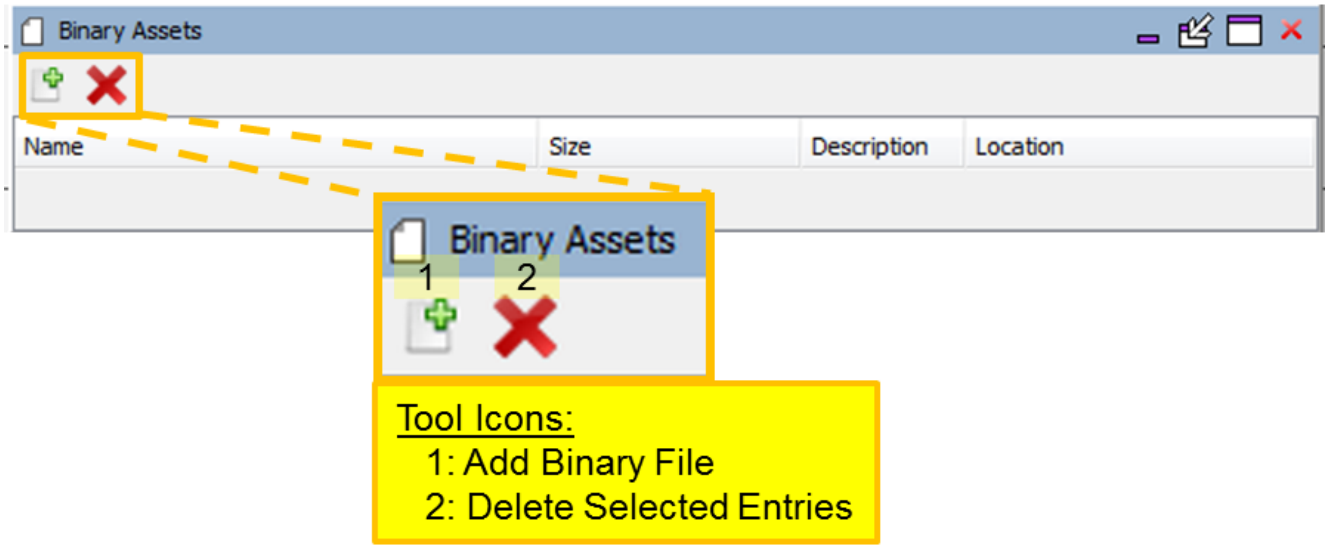
In the String Assets window, there will be a line for each of the languages defined for the application. Provide the string text and font for each of the languages. If you don't provide the text an empty string will be used instead. Not providing a font causes the string to be rendered as a string of boxes (□).

Binary Assets

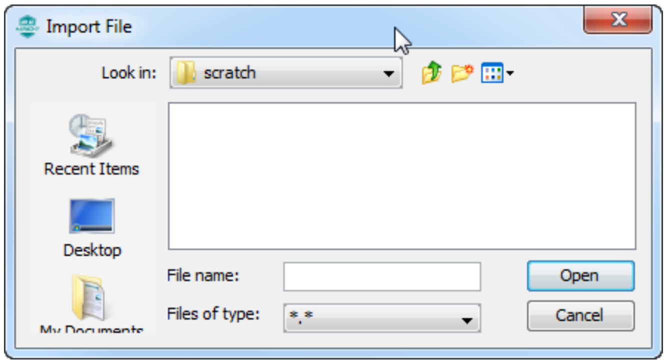
Provides information on the Binary Assets features.

Description

The Binary Assets window is launched from the Graphics Composer's Asset menu.



Selecting the Add Binary File icon (opens the Import File dialog.



This supports any formatted binary file. Developers can then add a custom-coded decoder to support the format implied by the imported file. (A future version of the GFX library will include a bin2code utility in support of this feature.)

MHGC Tools

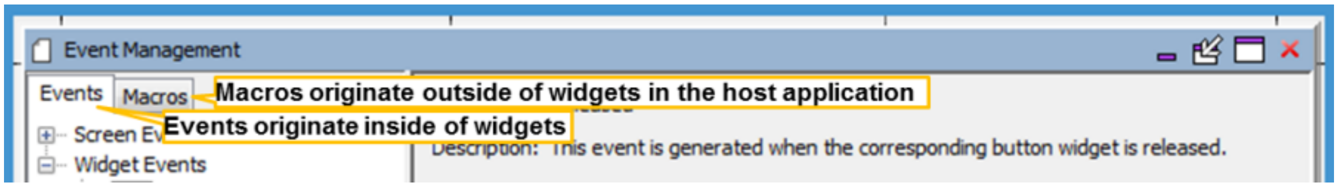
The Tools menu supports managing all graphics events, using a global palette, and estimating heap memory usage.

Event Manager

This section provide information on the Event Manager.

Description

The Graphics Composer Event Manager provides a GUI interface to manage all of the events associated with a graphics application. In a general sense, an event is an action or occurrence that is processed by software using an “event handler”. Button pushes or keystrokes are widely recognized and handled events. Events related to a touch screen are commonly called “gestures”. This GUI allows the assignment of actions to events associated with graphics widgets and to events outside of the graphics library. Under the Graphics Composer Event Manager tab there are two sub-tabs, one for “Events” and a second for “Macros”.


Event Management
Events
Macros
Macros originate outside of widgets in the host application
Events originate inside of widgets
Screen Ev
Widget Events
Description: This event is generated when the corresponding button widget is released.

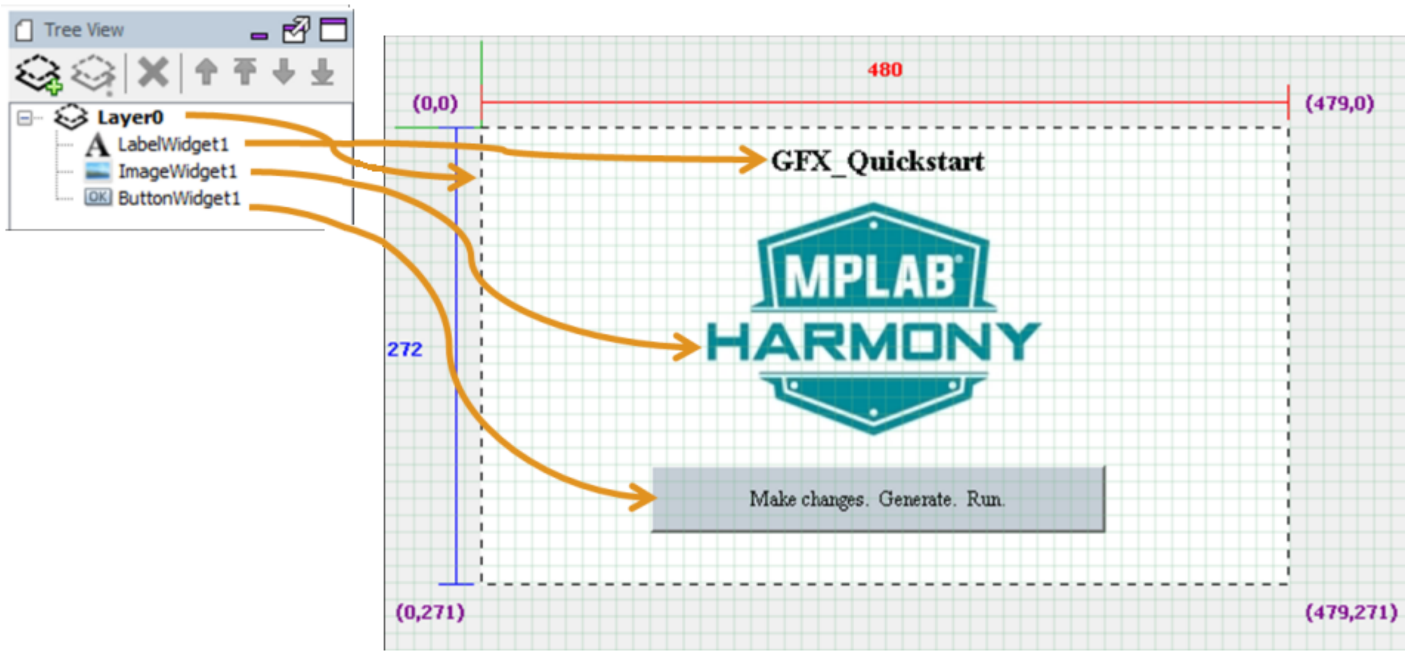
“Events” under the first tab are generated from within graphics widgets and can manipulate the properties of screen widgets or set semaphores that engage with the rest of the application. “Macros” are executed outside of graphics widgets by other parts of the application. “Macros” allow the application to change widget properties or behavior.

Both “Events” and “Macros” event handlers can be built using collections of “Template” actions or using “Custom” developer-provided code. Most widget properties have an associated Template action that can be used to manipulate that property in an event handler (either “Event” or “Macro”). For more information on properties and related actions, see the discussion on the Properties Window below.

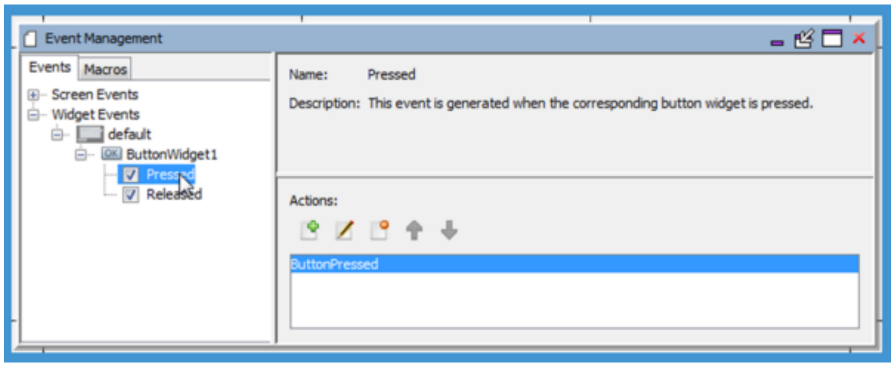
To explore these capabilities, let’s look at the Aria Quickstart project after the completion of the Adding an Event to the Aria Quickstart Demonstration Quick Start Guide.

Graphics Composer Events

The Graphics Composer Screen Designer shows that there is one layer and three widgets in this demonstration.



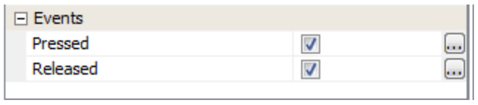
Of the three widgets shown above, only ButtonWidget1 can have events associated with it, one for button pressed and a second for button released. This can be seen in the Graphics Composer Event Manager window, which is available from the Tools menu:



Name: Pressed

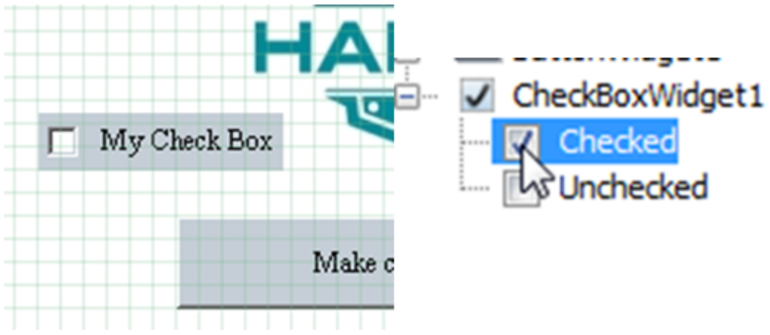
Description: This event is generated when the corresponding button widget is pressed.

The events shown under “ButtonWidget1” are mirrored in the widget’s properties. Selecting or clearing an event in one window does the same in the other window, thus enabling (selecting) or disabling (clearing) the corresponding event.

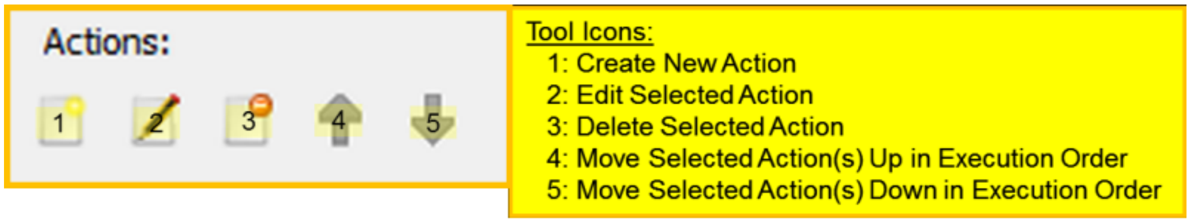


Events		
Pressed	✓	...
Released	✓	...

We can add a Check Box widget to the applications display and then use the Event Manager to assign actions to the widget’s events. A Check Box widget has two events, one for being “Checked” (i.e., selected) and another for being “Unchecked” (i.e., cleared). Enabling the “Checked” event then allows the selection of the action or actions for that event.



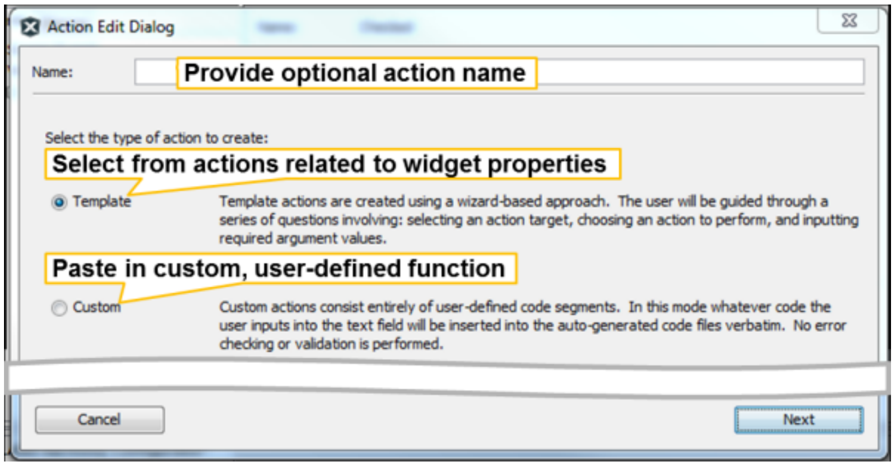
The Actions: sub-window has five tool icons for managing the actions associated with an event:



Actions:

Tool Icons:
1: Create New Action
2: Edit Selected Action
3: Delete Selected Action
4: Move Selected Action(s) Up in Execution Order
5: Move Selected Action(s) Down in Execution Order

Clicking the Create New Action icon ( ) opens the Action Edit dialog.



Action Edit Dialog

Name: Provide optional action name

Select the type of action to create:

Select from actions related to widget properties

Template

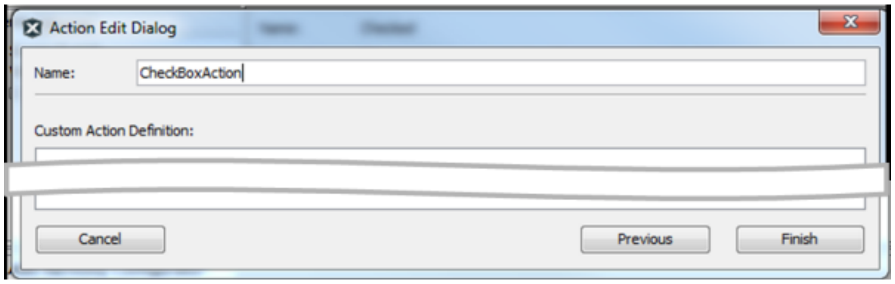
Template actions are created using a wizard-based approach. The user will be guided through a series of questions involving: selecting an action target, choosing an action to perform, and inputting required argument values.

Paste in custom, user-defined function

Custom

Custom actions consist entirely of user-defined code segments. In this mode whatever code the user inputs into the text field will be inserted into the auto-generated code files verbatim. No error checking or validation is performed.

If you select Custom and click **Next**, you will see the following dialog. Unfortunately, there is no C code error checking with this window. It just copies the code into `libaria.c` and `libaria.h`. If there is a problem with the code you will not know about it until you try to build your application. An alternative is just to type a comment such as `/*My event goes here*/`, generate the code, and then find out where this comment landed in the code. (Typically, inside `libaria_events.c`, or `libaria_macros.c`) You can then write the action routine from within the MPLAB X IDE editor and compile just that file to debug the code written.

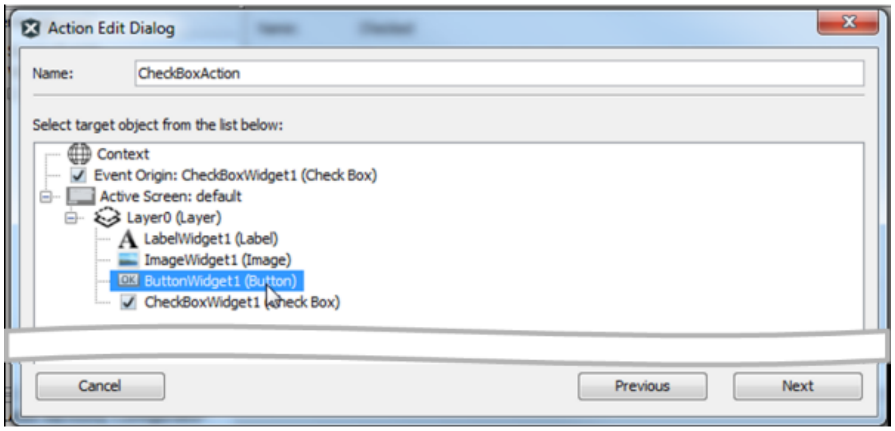


Action Edit Dialog

Name: CheckBoxAction

Custom Action Definition:

If you select Template, the Action Edit dialog will update, as follows. Select ButtonWidget1.



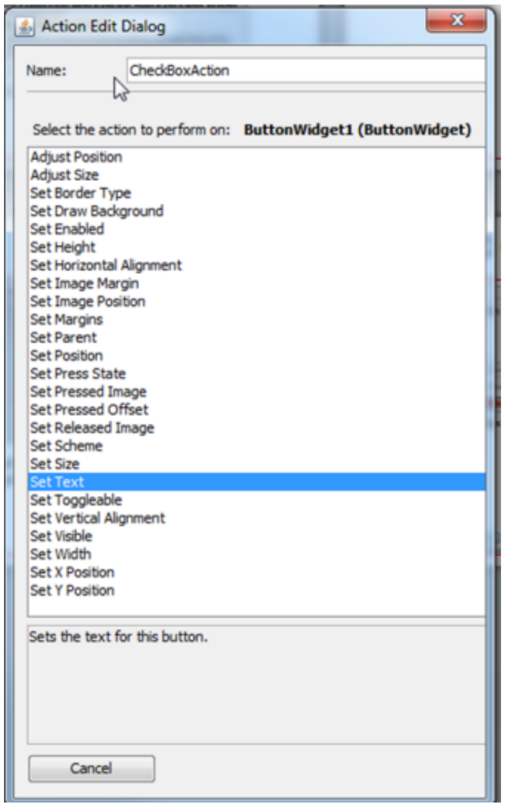
Action Edit Dialog

Name: CheckBoxAction

Select target object from the list below:

- Context
- Event Origin: CheckBoxWidget1 (Check Box)
- Active Screen: default
  - Layer0 (Layer)
    - LabelWidget1 (Label)
    - ImageWidget1 (Image)
    - ButtonWidget1 (Button)
    - CheckBoxWidget1 (Check Box)

As shown previously, you next need to select the widget that you want to manipulate with this action. Note that the event originated with CheckBoxWidget1, but the event's action can manipulate any of the existing widgets. In this case, ButtonWidget1 has been selected. Clicking Next will then bring up a list of the actions available in manipulating a button widget.



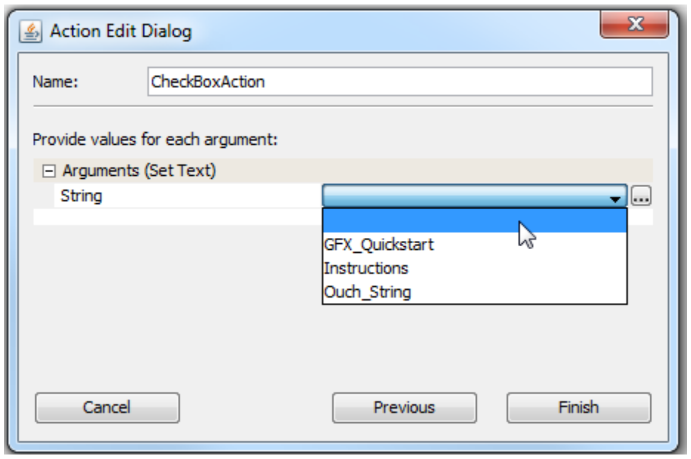
Name: CheckBoxAction

Select the action to perform on: **ButtonWidget1 (ButtonWidget)**

Adjust Position
Adjust Size
Set Border Type
Set Draw Background
Set Enabled
Set Height
Set Horizontal Alignment
Set Image Margin
Set Image Position
Set Margins
Set Parent
Set Position
Set Press State
Set Pressed Image
Set Pressed Offset
Set Released Image
Set Scheme
Set Size
Set Text
Set Toggleable
Set Vertical Alignment
Set Visible
Set Width
Set X Position
Set Y Position

Sets the text for this button.

You can select the “Set Text” action, which will then change the button’s text property, followed by NEXT, which will open a dialog to select the text string for this action.



Name: CheckBoxAction

Provide values for each argument:

Arguments (Set Text)

String

GFX_Quickstart
Instructions
Ouch_String

You can then select from the available (already defined) strings which text to use for the button’s text field. Press the Finish button to complete the definition of this action.

Screen Events

As shown previously, the Graphics Composer Event Manager, Events sub-tab supports screen events when the screen is visible (On Show) and hidden (On Hide). These events can define event handlers based on Template actions or Custom, user-defined code.

Widget Events

Not all widgets can generate an event. For example, a Label Widget has nothing to generate, it just sits there on the screen, labeling. Here is a list of the widgets that can generate an event:

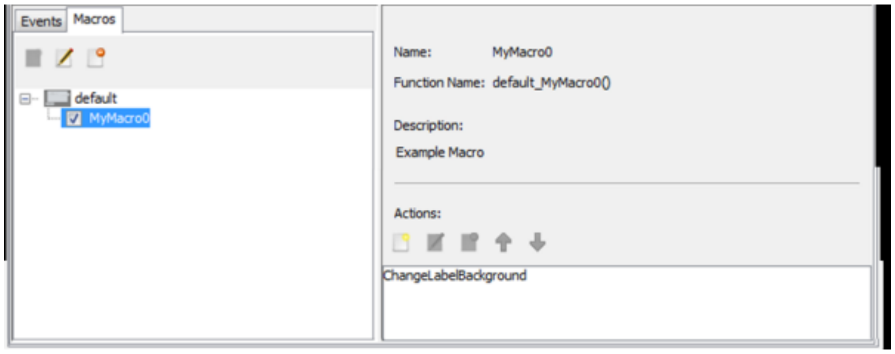
- Button – Pressed and Released events
- Check Box – Checked and Unchecked events
- Draw Surface – Draw Notification event
- Image Sequence – Image Changed event

- Key Pad – Key Click event
- List Wheel – Select Item Changed event
- List – Selection Changed event
- Progress Bar – Value Changed event
- Radio Button – Selected and Deselected event
- Scroll Bar – Value Changed event
- Slider Widget – Value Changed event
- Text Field – Text Changed event
- Touch Test – Point Added event

Graphics Composer Macros

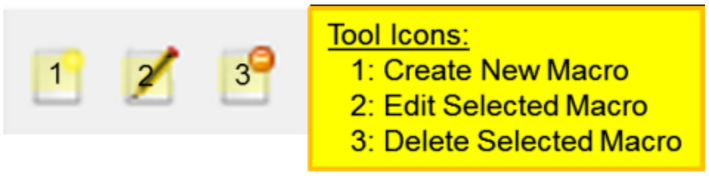
Macros implement event handlers for events that originate outside of graphics primitives such as widgets and are designed to change or manipulate widgets inside of the graphics part of an application. (Events that originate outside of graphics and don't touch the graphics part of the application are outside of the scope of the Graphics Event Manager and are not discussed here.)

The following figure shows a simple example of a macro.



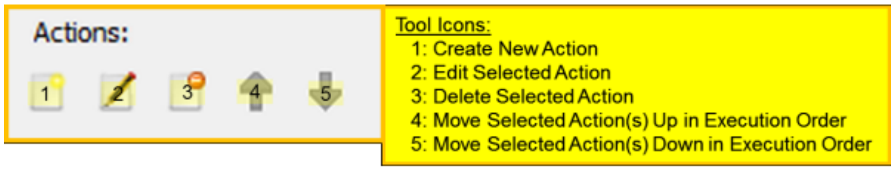
Name: MyMacro0
Function Name: default_MyMacro0()
Description:
Example Macro
ChangeLabelBackground

The toolbar for Macros has three icons.

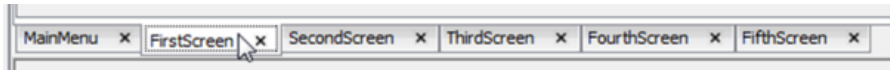


Creating a new macro and selecting its actions is just like that of a widget event:

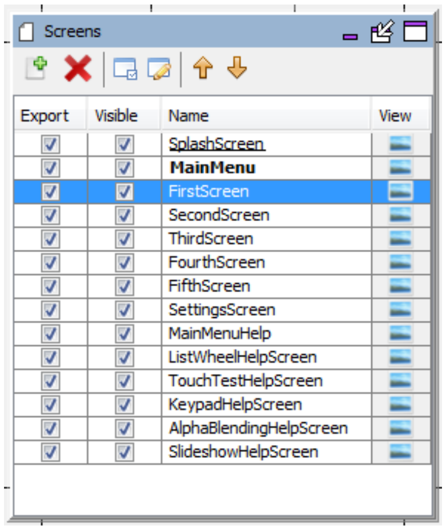
1. Create a new macro using the "Create New Macro" tool. The check box to the left of the new macro's name enables/disables the macro. Clearing it removes the macro from the next code generation.
2. Select the new macro and edit it using the second icon (shown previously).
3. In the Actions: window, select Create New Action. An optional name can be provided in the Name: box. You can then choose to use a Template and select a predefined action or Custom to create a customized action.


Actions:
1
2
3
4
5
Tool Icons:
1: Create New Action
2: Edit Selected Action
3: Delete Selected Action
4: Move Selected Action(s) Up in Execution Order
5: Move Selected Action(s) Down in Execution Order

4. If you chose a "Custom" action, proceed as discussed previous in Graphics Composer Events. When using templates the next step is to choose the target widget for the action. This choice is limited to those only the widgets in the currently "active" screen. If your application has multiple screens and the widget you are targeting is not part of the currently active screen you need to change the active screen.
   - Changing the active screen can be done by selecting the corresponding screen tab at the bottom of the Graphics Composer Screen Designer



- Alternately, you can switch using the Graphics Composer Manager:Screens tab



Export	Visible	Name	View
✓	✓	SplashScreen	
✓	✓	MainMenu	
✓	✓	FirstScreen	
✓	✓	SecondScreen	
✓	✓	ThirdScreen	
✓	✓	FourthScreen	
✓	✓	FifthScreen	
✓	✓	SettingsScreen	
✓	✓	MainMenuHelp	
✓	✓	ListWheelHelpScreen	
✓	✓	TouchTestHelpScreen	
✓	✓	KeypadHelpScreen	
✓	✓	AlphaBlendingHelpScreen	
✓	✓	SlideshowHelpScreen	

5. After selecting the target widget for this macro, click Next button to select an action related to this widget. (Just as with template-based widget events.) The macro can contain more than one action, targeting more than one widget.

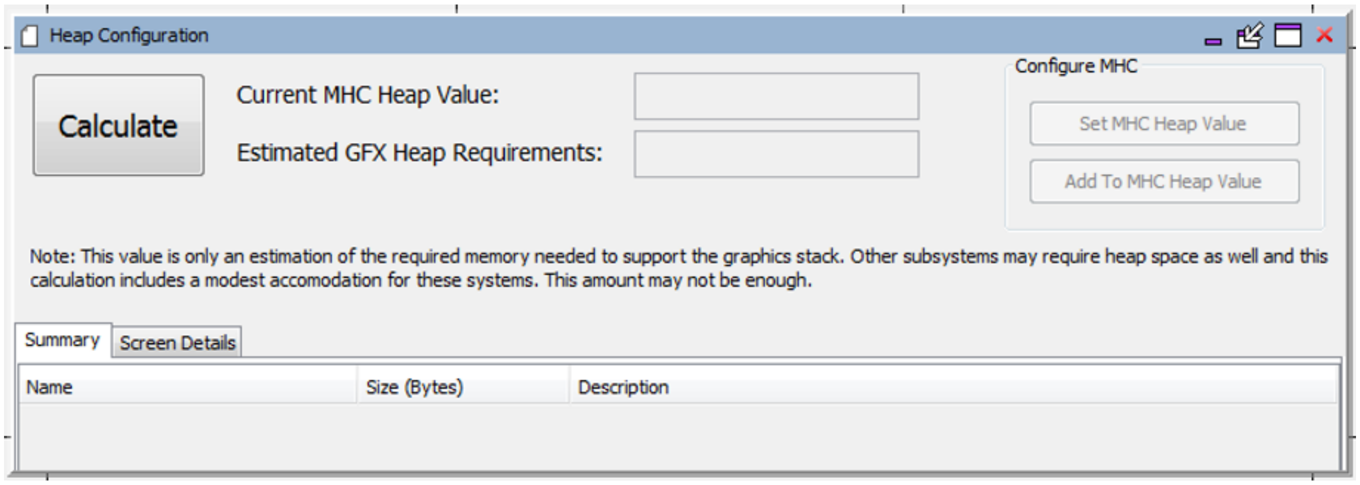
Heap Estimator

Provides information on heap space allocation.

Description

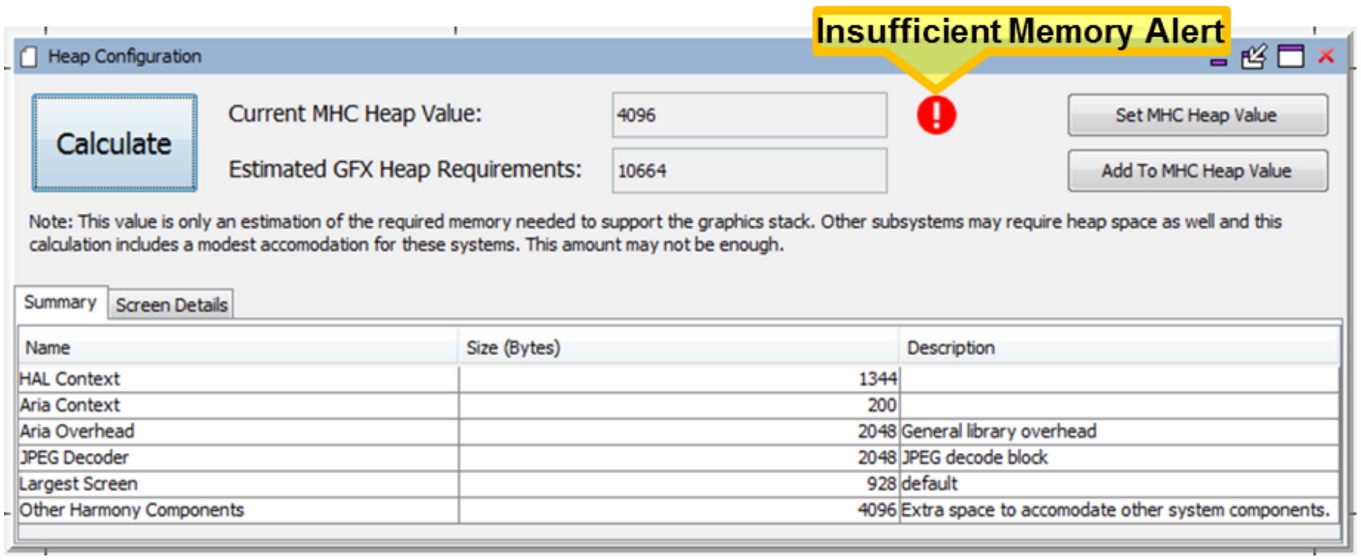
Many parts of a graphics design are implemented using memory allocated from the application's heap space. Therefore, it is important to allocate sufficient memory for the heap. This tool can estimate heap usage by the allocation based on the widgets, layers, screens, and decoders currently in the design.

When launching the tool from the Tools menu, the Heap Configuration window appears.



Note: This value is only an estimation of the required memory needed to support the graphics stack. Other subsystems may require heap space as well and this calculation includes a modest accomodation for these systems. This amount may not be enough.

Clicking **Calculate** estimates heap usage. The following figure shows what occurs within the Aria Quickstart demonstration if the heap space is only 4096 bytes:



Current MHC Heap Value: 4096

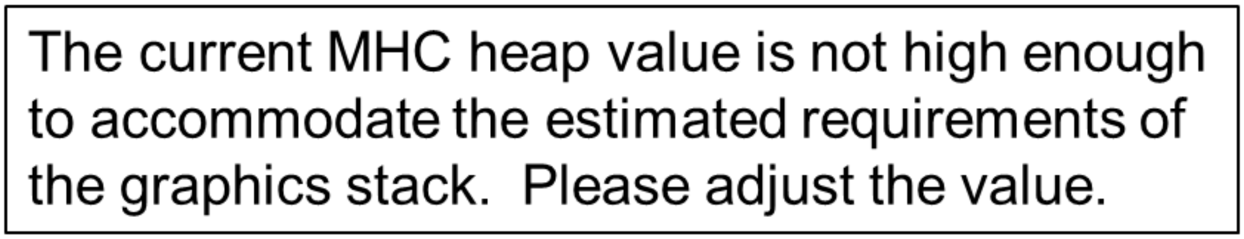
Estimated GFX Heap Requirements: 10664

Note: This value is only an estimation of the required memory needed to support the graphics stack. Other subsystems may require heap space as well and this calculation includes a modest accomodation for these systems. This amount may not be enough.

Name	Size (Bytes)	Description
HAL Context	1344	
Aria Context	200	
Aria Overhead	2048	General library overhead
JPEG Decoder	2048	JPEG decode block
Largest Screen	928	default
Other Harmony Components	4096	Extra space to accomodate other system components.

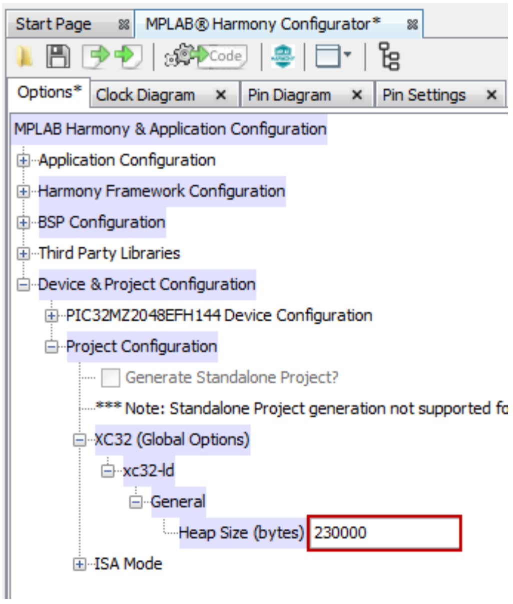
The Summary tab shows how the estimated heap requirements was derived by summing up all the sizes shown under the “Size (Bytes)” column. Note that the largest contribution comes from the screen requiring the largest heap allocation (in this case MainMenu).

If there is insufficient memory allocated to the heap, an exclamation point (!) appears in the window. If you hold your mouse pointer over this icon, the following message appears:


The current MHC heap value is not high enough to accommodate the estimated requirements of the graphics stack. Please adjust the value.

You can click **Set MHC Heap Value** to reset the heap allocation to match the estimated requirements. Selecting **Add to MHC Heap Value** adds the estimated heap requirements to the current heap value. (In the case above, this would change the heap allocation to 4096+10664 bytes.)

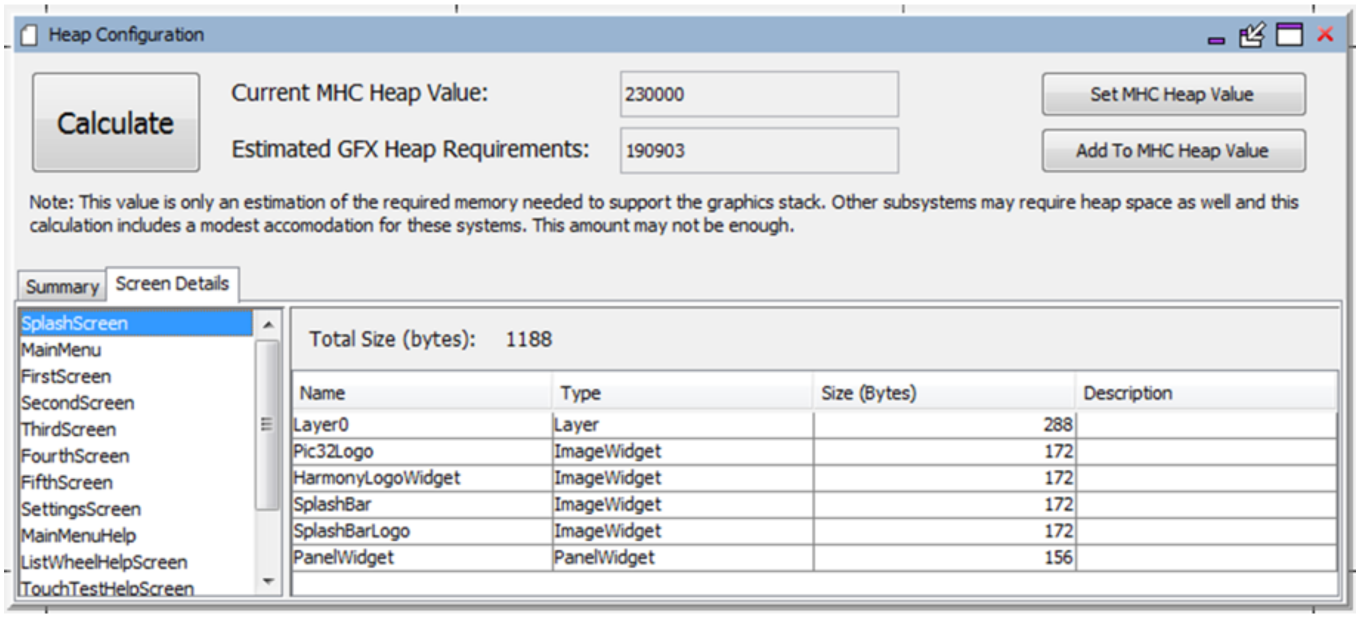
Alternately, you can set the heap allocation to a larger value by going to the MPLAB Harmony Configurator window, selecting the Options tab and setting the Heap Size within *Device & Project Configuration* > *Project Configuration*.


Start Page
MPLAB® Harmony Configurator*
Code
Options*
Clock Diagram
Pin Diagram
Pin Settings
MPLAB Harmony & Application Configuration
Application Configuration
Harmony Framework Configuration
BSP Configuration
Third Party Libraries
Device & Project Configuration
PIC32MZ2048EFH144 Device Configuration
Project Configuration
Generate Standalone Project?
*** Note: Standalone Project generation not supported fc
XC32 (Global Options)
xc32-ld
General
ISA Mode

Heap Size (bytes) 230000

The Screen Details tab (from the Aria Showcase demonstration) shows screen-by-screen the heap space needed for each layer and widget on the screen selected.

Note: After you have updated the Heap Size, either using the Heap Estimator tool or by directly editing the value as shown above, you must regenerate the project using the Generate Code button. This will update the actual heap size value used in building the application.



Current MHC Heap Value: 230000

Estimated GFX Heap Requirements: 190903

Note: This value is only an estimation of the required memory needed to support the graphics stack. Other subsystems may require heap space as well and this calculation includes a modest accomodation for these systems. This amount may not be enough.

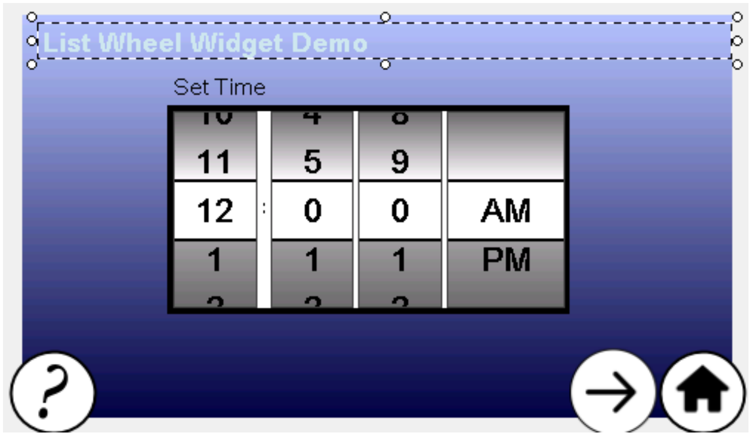
SplashScreen
MainMenu
FirstScreen
SecondScreen
ThirdScreen
FourthScreen
FifthScreen
SettingsScreen
MainMenuHelp
ListWheelHelpScreen
TouchTestHelpScreen

Total Size (bytes): 1188

Name	Type	Size (Bytes)	Description
Layer0	Layer	288	
Pic32Logo	ImageWidget	172	
HarmonyLogoWidget	ImageWidget	172	
SplashBar	ImageWidget	172	
SplashBarLogo	ImageWidget	172	
PanelWidget	PanelWidget	156	

Clicking the “Name” column will alphabetize the list. Clicking the “Size (Bytes)” column sorts the assets by size, with the largest at the top and smallest at the bottom.

This sub-tab can help in managing the application’s utilization of heap space. For example, excess use of cached backgrounds for widgets can become ruinously expensive, expanding the application’s need for heap well beyond the capabilities of the device. As an example, consider a screen label from the Aria Showcase demonstration.



The Heap Estimator tool shows that if caching is enabled for the label’s background, this widget requires 23699 bytes of heap to store the widget. Note that the label is twice the size of the text it contains, so one way of reducing the cost of the widget is to make it smaller, thereby reducing the number of background pixels that must be stored. If the label is resized, the heap allocation is reduced to 11688 bytes, which is a drop of appoximately 50%. Finally, if the background is changed from “Cache” to “Fill” the widget only needs 188 bytes.

The lesson learned is to use Cache as a background only for widgets where it is absolutely necessary and to make the “cached” widgets as small as possible.

Global Palette

Provides information on the Global Palette features.

Description

The Global Palette window is launched from the Graphics Composer’s Asset pull-down menu.

Using a Global Palette enables frame buffer compression for the LCC graphics controller. It creates a 256 color look up table (LUT) and then changes the entire user interface design to adhere to that LUT. Frame buffers are stored as 8 bits/pixel (bpp) indices rather than 16-32 bpp colors. The display driver performs a LUT operation to change each LUT index into a color before writing to the display/controller memory. This enables the use of double buffering, without using external memory, on devices that could not support it before. It also supports single buffering on larger displays. Of course, running the LUT requires more processing on the host. Currently only the LCC graphics controller supports this feature. The Aria demonstration Aria Basic Motion is an example of how using a Global Palette greatly improves the efficiency and capabilities of a design.

Enable the Global Palette by clicking on the Enable Global Palette check box in the window or using the *File > Settings menu*. the Global palette can always be disabled. MHGC will then restore the project back to its original configuration.

If the global palette is enabled you will have to change the MHC configuration of the Graphics Controller to match. For the LCC controller, enable

"Palette Mode". For the GLCD controller, change the Driver Settings > Fame Buffer Color Mode to "LUT8".

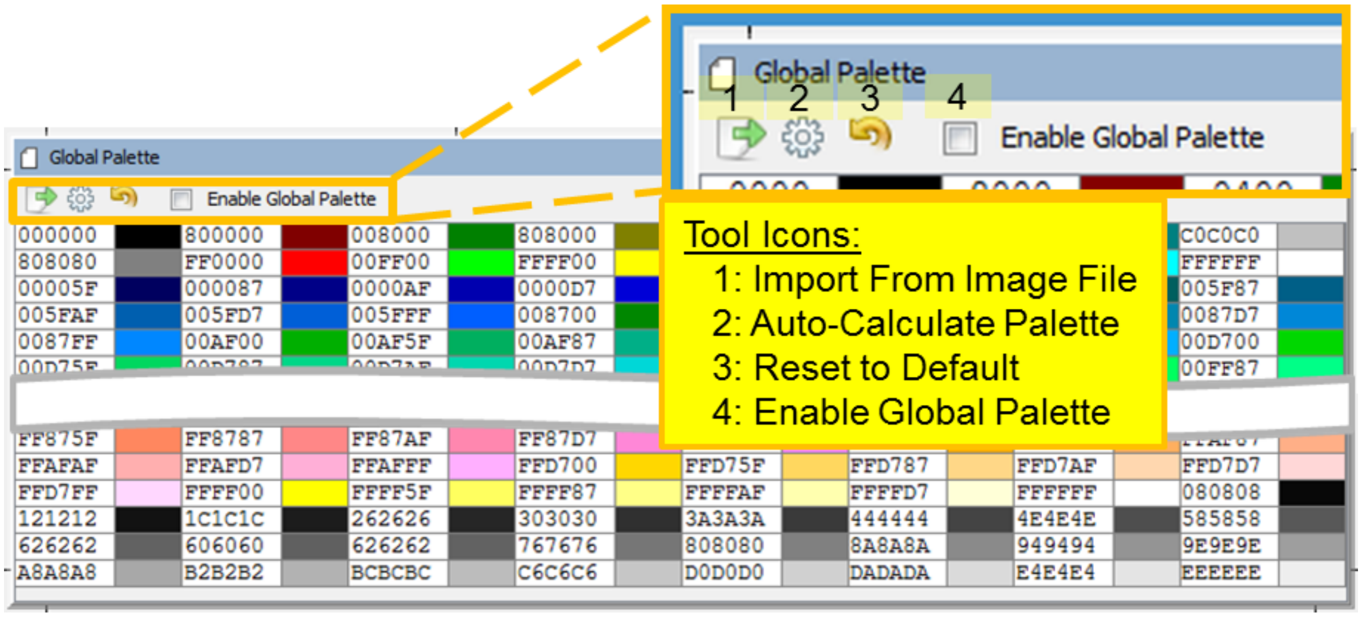
The results of enabling the Global Palette:

- 8bpp frame buffers. In the case of the most common demonstrations this means a 50% reduction in the size of the frame buffer.
- This also opens up the capability to support a single frame buffer for some larger displays.

What is lost by enabling the Global Palette:

- First and foremost - No Dynamic Colors. Dynamic colors are unlikely to match up with an entry in the global palette’s look-up table.
- No alpha blending capability. The level of alpha blending can be changed during run-time. (See No Dynamic Colors.)
- No JPEGs or PNGs. Again, no dynamic colors. All images in MGHC will be changed to the color mode of the project, and generated as Raw.
- No font anti-aliasing. Again, no dynamic colors. While the 8-bits/pixel for each glyph is known, the color of the text depends on the color scheme used, and color schemes can change at run time.
- Additional overhead when performing LUT (index->color) operations in the display driver.

The following figure shows the default “Global Palette” when Project Color Mode is set to RGB_888.



000000	800000	008000	808000	[illegible]	[illegible]	[illegible]	C0C0C0
808080	FF0000	00FF00	FFFF00	[illegible]	[illegible]	[illegible]	FFFFFF
00005F	000087	0000AF	0000D7	[illegible]	[illegible]	[illegible]	005F87
005FAF	005FD7	005FFF	008700	[illegible]	[illegible]	[illegible]	0087D7
0087FF	00AF00	00AF5F	00AF87	[illegible]	[illegible]	[illegible]	00D700
00D75F	00D787	00D7AF	00D7D7	[illegible]	[illegible]	[illegible]	00FF87
FF875F	FF8787	FF87AF	FF87D7	[illegible]	[illegible]	[illegible]	[illegible]
FFAFAF	FFAFD7	FFAFFF	FFD700	FFD75F	FFD787	FFD7AF	FFD7D7
FFD7FF	FFFF00	FFFF5F	FFFF87	FFFFAF	FFFFD7	FFFFFF	080808
121212	1C1C1C	262626	303030	3A3A3A	444444	4E4E4E	585858
626262	606060	626262	767676	808080	8A8A8A	949494	9E9E9E
A8A8A8	B2B2B2	BCBCBC	C6C6C6	D0D0D0	DADADA	E4E4E4	EEEEEE

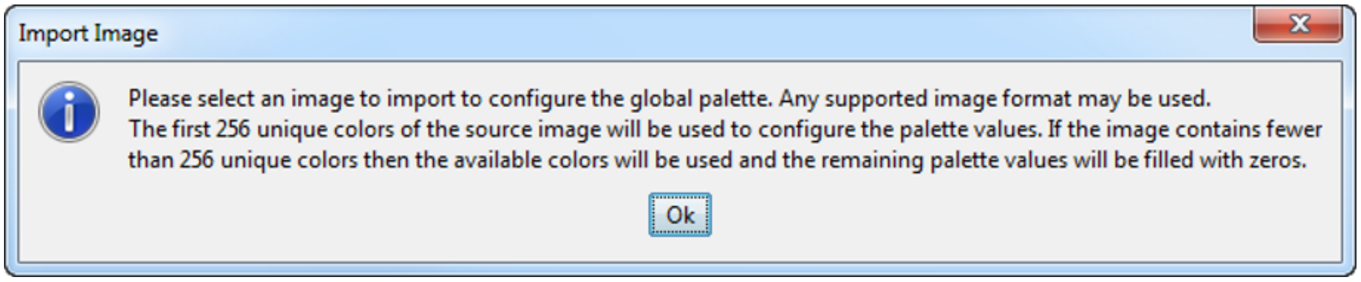
This default palette is good for designs that use a wide array of colors. MHGC also supports developing a custom palette by importing an image defining the palette or by analyzing the pixel colors already in use by the application’s images. The palette’s color mode is determined by the Project Color Mode, which is determined by the graphics controller.

Clicking on an entry in the palette with bring up the Color Picker dialog window, allowing you to edit the entry’s color.

Window Toolbar

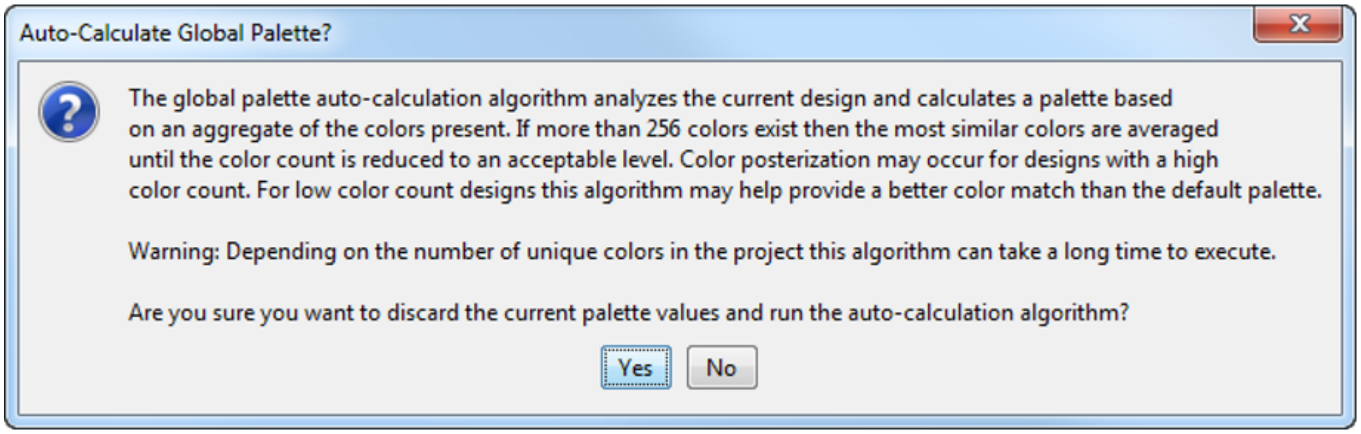
There are four icons on the toolbar:

1. **Import From Image File** - Importing a global palette from an image file. Selecting this brings up the following warning. Images can be imported as a BMP,.GIF, JPEG, and PNG (but not TIFF).



Please select an image to import to configure the global palette. Any supported image format may be used.
The first 256 unique colors of the source image will be used to configure the palette values. If the image contains fewer than 256 unique colors then the available colors will be used and the remaining palette values will be filled with zeros.

2. **Auto-Calculate Palette** – Calculates a new palette using the current design. Selecting this brings up the following warning.



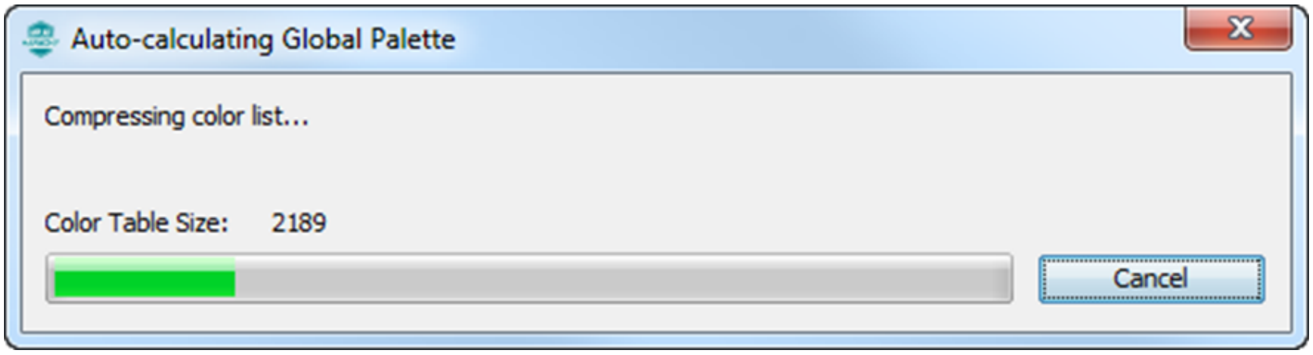
Auto-Calculate Global Palette?

The global palette auto-calculation algorithm analyzes the current design and calculates a palette based on an aggregate of the colors present. If more than 256 colors exist then the most similar colors are averaged until the color count is reduced to an acceptable level. Color posterization may occur for designs with a high color count. For low color count designs this algorithm may help provide a better color match than the default palette.

Warning: Depending on the number of unique colors in the project this algorithm can take a long time to execute.

Are you sure you want to discard the current palette values and run the auto-calculation algorithm?

- Selecting **Yes** opens a status window that shows the progress made in selecting a palette of 256 colors



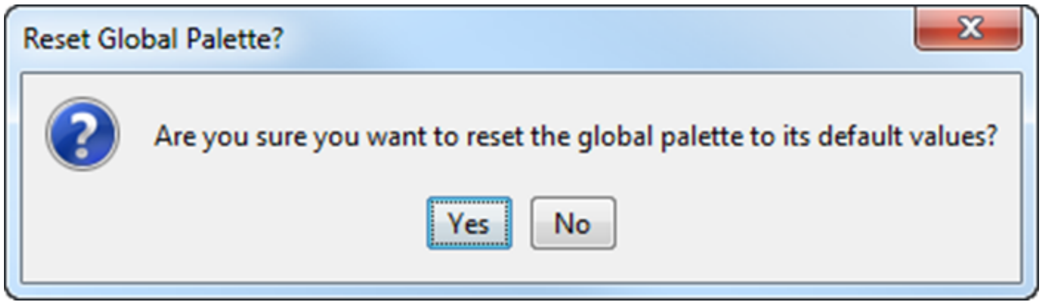
Auto-calculating Global Palette

Compressing color list...

Color Table Size: 2189

- This can be lengthy operation, but it will effectively generate a palette better tailored to the design. However, extreme (or rare) colors will be changed to nearby, more-plentiful colors, thereby eliminating some of the contrast in images. Whites will tend to darken and blacks lighten. This can be remedied by editing the calculated palette to whiten the whites, darken the blacks, and make other colors closer to the original. This of course may increase the posterization of the image, but that is a natural trade-off in using only 256 colors.

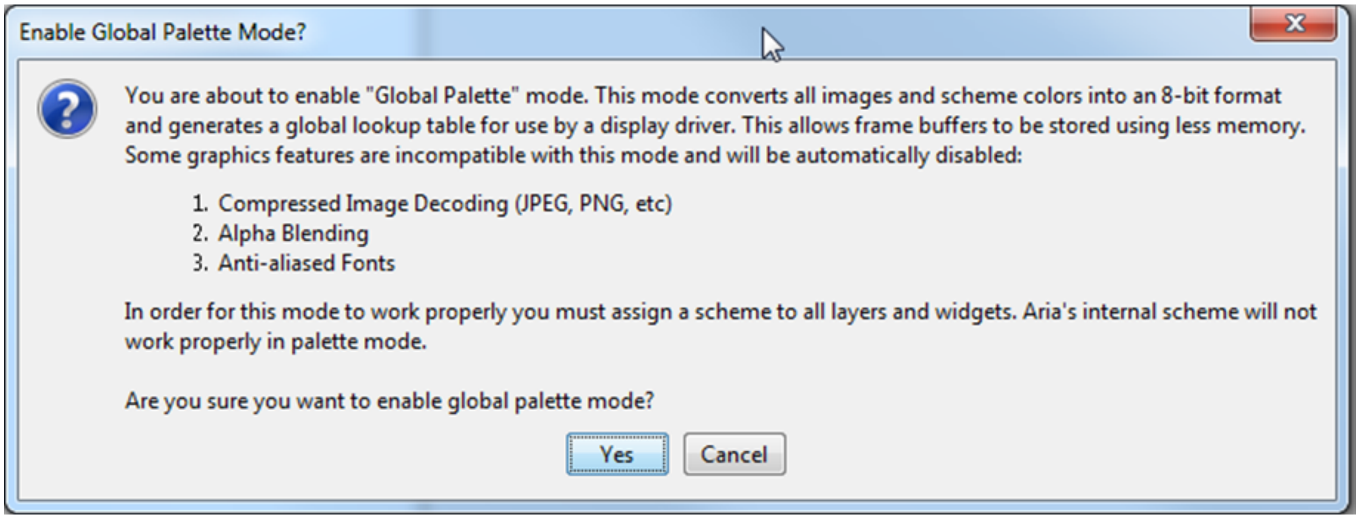
3. **Reset to Default** – This returns the Global Palette to its default values, which opens the Reset Global Palette dialog.



Reset Global Palette?

Are you sure you want to reset the global palette to its default values?

4. **Enable Global Palette** – This performs the same function as *File > Settings: Using a Global Palette*. Selecting this opens the Enable Global Palette Mode warning.


Enable Global Palette Mode?

You are about to enable "Global Palette" mode. This mode converts all images and scheme colors into an 8-bit format and generates a global lookup table for use by a display driver. This allows frame buffers to be stored using less memory. Some graphics features are incompatible with this mode and will be automatically disabled:

1. Compressed Image Decoding (JPEG, PNG, etc)
2. Alpha Blending
3. Anti-aliased Fonts

In order for this mode to work properly you must assign a scheme to all layers and widgets. Aria's internal scheme will not work properly in palette mode.

Are you sure you want to enable global palette mode?

Widget Colors

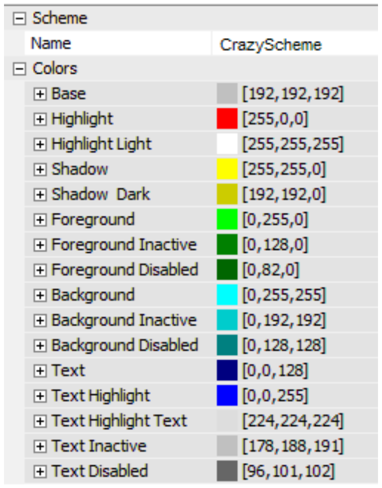
Provides information on widget coloring.

Description

Widget Colors

Widget coloring can be customized by creating additional color schemes and assigning these customized schemes to a subset of the widgets uses. For example, a ButtonColorScheme could be customized and used only for Button Widgets.

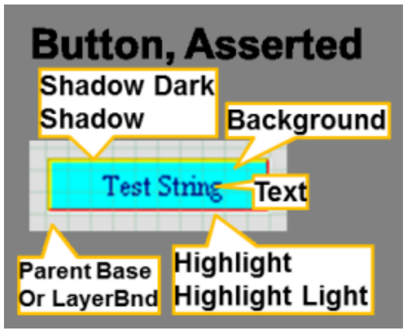
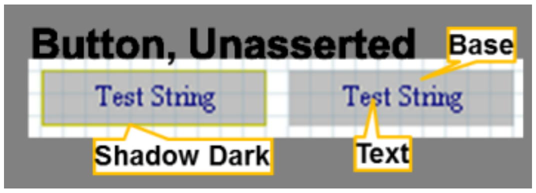
To help highlight the different colors available for each widget, a “CrazyScheme”, with extreme contrast among the 16 available colors, was used as the color scheme for each widget:

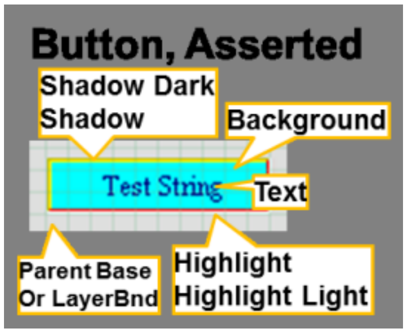


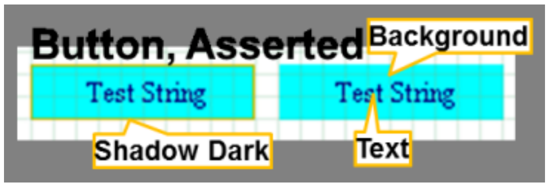
Scheme	
Name	CrazyScheme
Colors	
Base	[192,192,192]
Highlight	🟥 [255,0,0]
Highlight Light	[255,255,255]
Shadow	🟨 [255,255,0]
Shadow Dark	🟨 [192,192,0]
Foreground	🟩 [0,255,0]
Foreground Inactive	🟩 [0,128,0]
Foreground Disabled	🟩 [0,82,0]
Background	🟦 [0,255,255]
Background Inactive	🟦 [0,192,192]
Background Disabled	🟦 [0,128,128]
Text	🟦 [0,0,128]
Text Highlight	🟦 [0,0,255]
Text Highlight Text	[224,224,224]
Text Inactive	[178,188,191]
Text Disabled	[96,101,102]

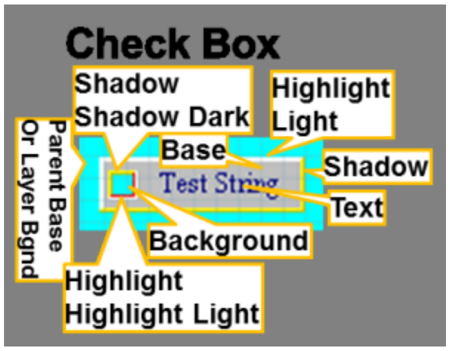
Use this color scheme to help identify the relevant colors for the widgets listed below.

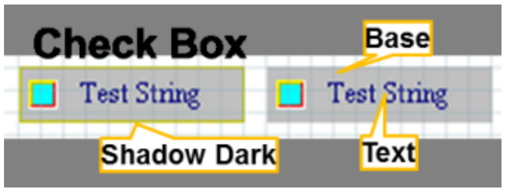
The left column shows the coloring assignments for a Bezel boarder. The right side shows Line/No Border color assignments.

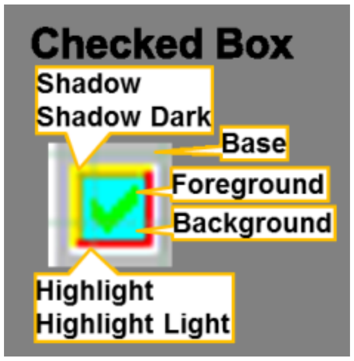
Widget With Bezel Border	Widget With Line or No Borders
 Button, Asserted Shadow Dark Shadow Background Test String Text Parent Base Or LayerBnd Highlight Highlight Light	 Button, Unasserted Base Test String Test String Shadow Dark Text

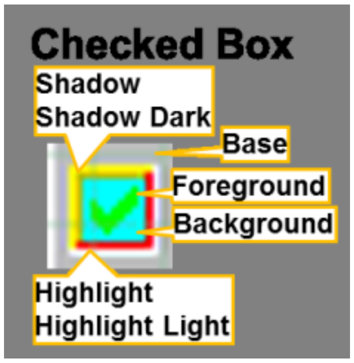

Button, Asserted
Shadow Dark
Shadow
Background
Test String
Text
Parent Base
Or LayerBnd
Highlight
Highlight Light

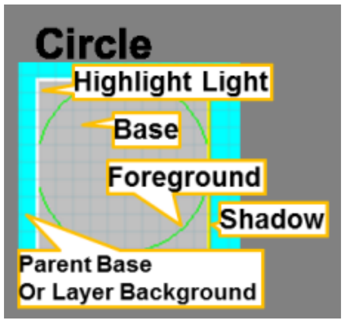

Button, Asserted
Background
Test String
Test String
Shadow Dark
Text

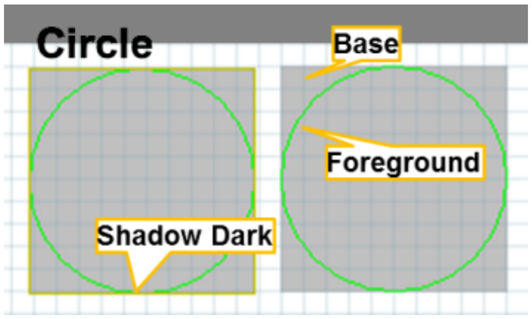

Check Box
Shadow
Shadow Dark
Highlight
Light
Parent Base
Or Layer Bgnd
Base
Test String
Shadow
Text
Background
Highlight
Highlight Light

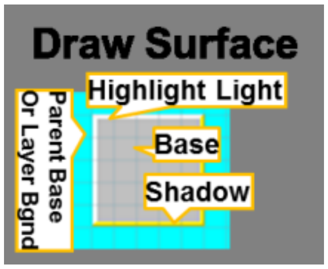

Check Box
Base
Test String
Test String
Shadow Dark
Text

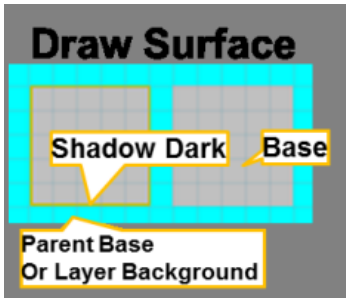

Checked Box
Shadow
Shadow Dark
Base
Foreground
Background
Highlight
Highlight Light

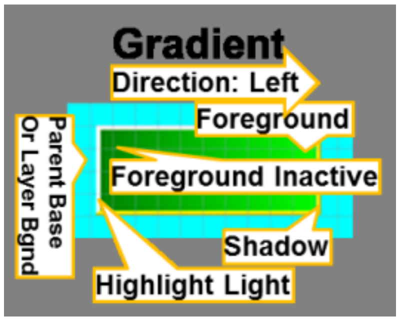
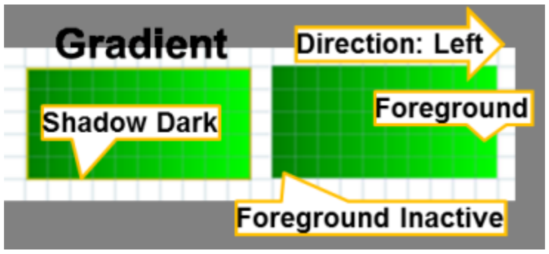
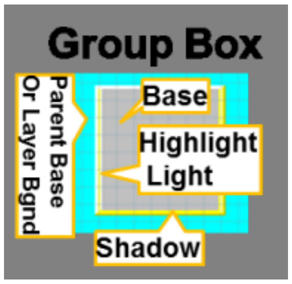
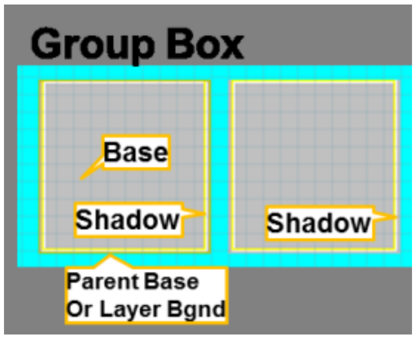
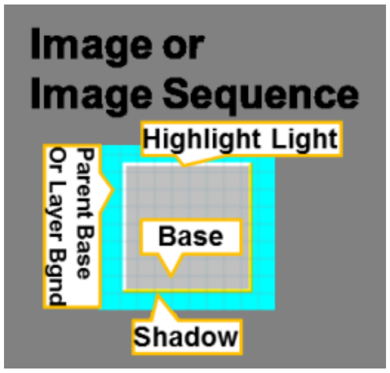
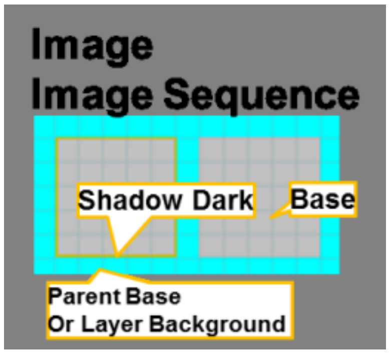
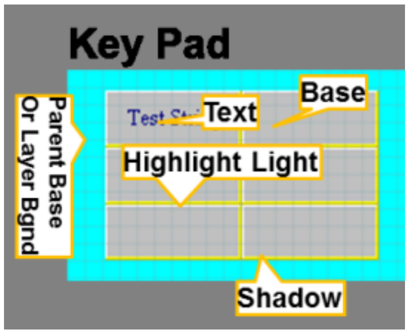
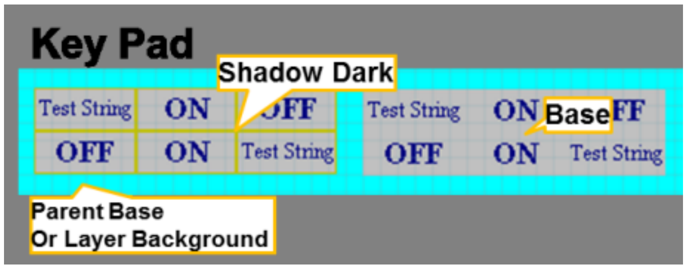
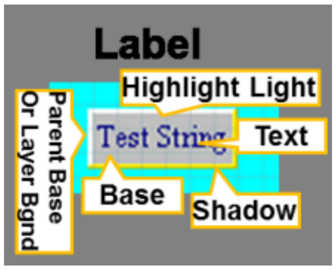
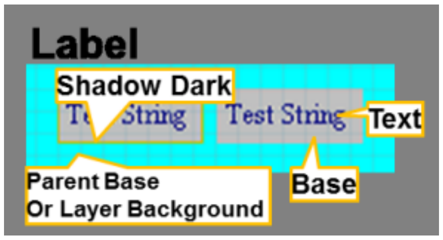

Checked Box
Shadow
Shadow Dark
Base
Foreground
Background
Highlight
Highlight Light

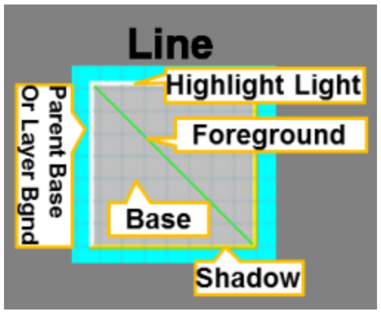

Circle
Highlight Light
Base
Foreground
Shadow
Parent Base
Or Layer Background

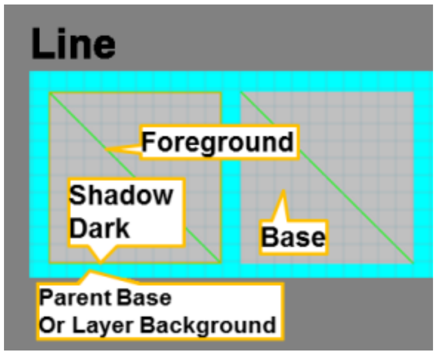

Circle
Base
Foreground
Shadow Dark

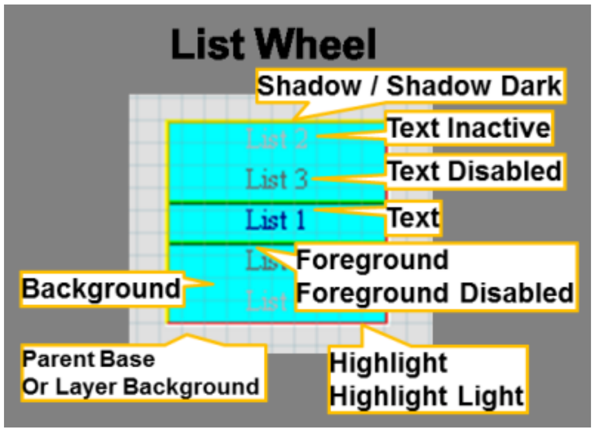

Draw Surface
Highlight Light
Parent Base
Or Layer Bgnd
Base
Shadow

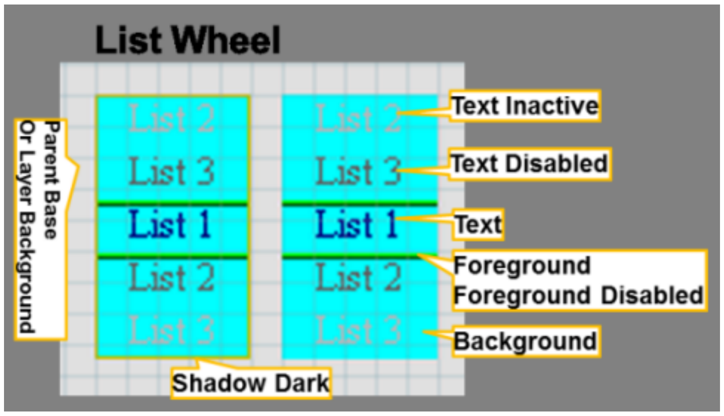

Draw Surface
Shadow Dark
Base
Parent Base
Or Layer Background

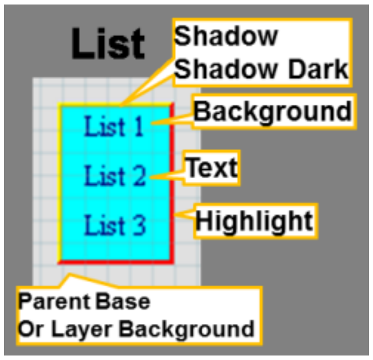
Gradient Direction: Left Foreground Foreground Inactive Parent Base Or Layer Bgnd Shadow Highlight Light	Gradient Direction: Left Shadow Dark Foreground Foreground Inactive
Group Box Parent Base Or Layer Bgnd Base Highlight Light Shadow	Group Box Base Shadow Shadow Parent Base Or Layer Bgnd
Image or Image Sequence Highlight Light Parent Base Or Layer Bgnd Base Shadow	Image Image Sequence Shadow Dark Base Parent Base Or Layer Background
Key Pad Parent Base Or Layer Bgnd Text Base Highlight Light Shadow	Key Pad Shadow Dark Test String ON OFF ON OFF ON Test String Base OFF ON Test String Parent Base Or Layer Background
Label Highlight Light Parent Base Or Layer Bgnd Test String Text Base Shadow	Label Shadow Dark Test String Test String Text Parent Base Or Layer Background Base

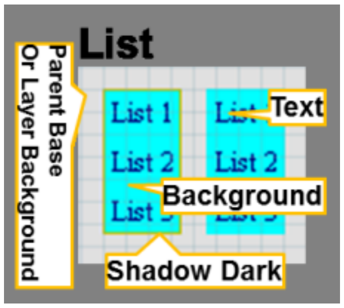

Line
Highlight Light
Foreground
Parent Base Or Layer Bgnd
Base
Shadow

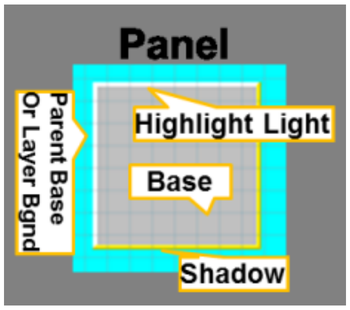

Line
Foreground
Shadow Dark
Base
Parent Base Or Layer Background

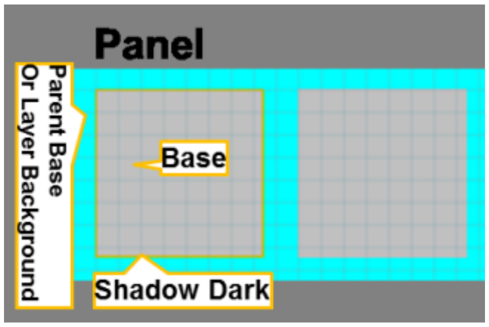

List Wheel
Shadow / Shadow Dark
Text Inactive
Text Disabled
Text
Foreground
Foreground Disabled
Background
List 2
List 3
List 1
Parent Base Or Layer Background
Highlight
Highlight Light

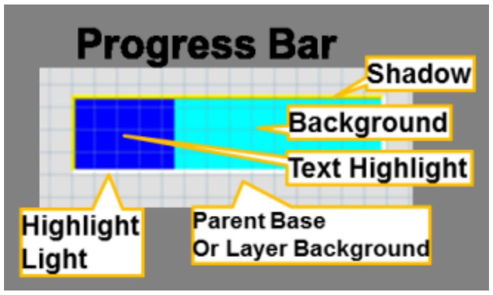

List Wheel
Parent Base Or Layer Background
List 2
List 3
List 1
Text Inactive
Text Disabled
Text
Foreground
Foreground Disabled
Background
Shadow Dark

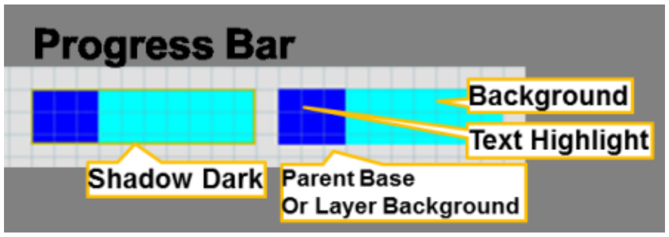

List
Shadow
Shadow Dark
Background
Text
Highlight
List 1
List 2
List 3
Parent Base Or Layer Background

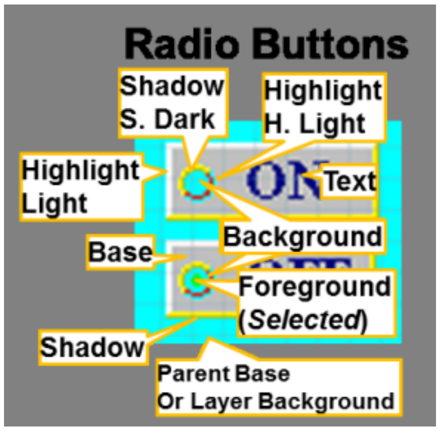

List
Parent Base Or Layer Background
List 1
List 2
Text
Background
Shadow Dark

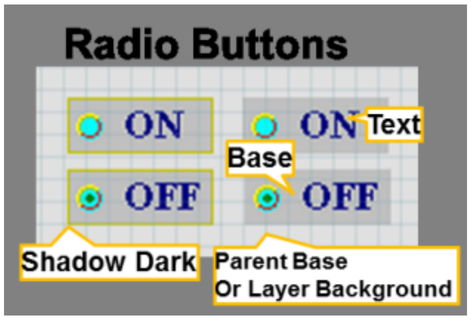

Panel
Parent Base Or Layer Bgnd
Highlight Light
Base
Shadow

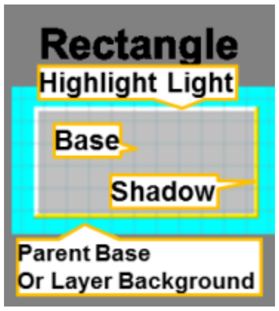

Panel
Parent Base Or Layer Background
Base
Shadow Dark

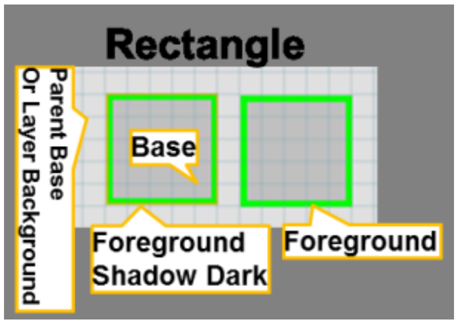

Progress Bar
Shadow
Background
Text Highlight
Highlight Light
Parent Base Or Layer Background

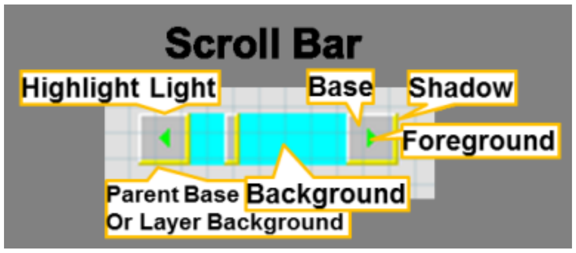

Progress Bar
Background
Text Highlight
Shadow Dark
Parent Base Or Layer Background

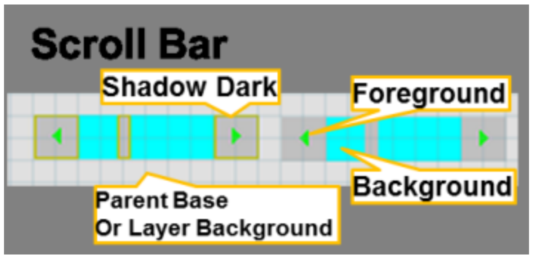

Radio Buttons
Shadow
S. Dark
Highlight
H. Light
Highlight
Light
ON
Text
Base
Background
Foreground
(*Selected*)
Shadow
Parent Base
Or Layer Background

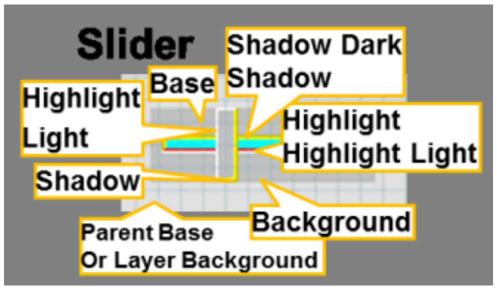

Radio Buttons
ON
ON
Text
Base
OFF
OFF
Shadow Dark
Parent Base
Or Layer Background

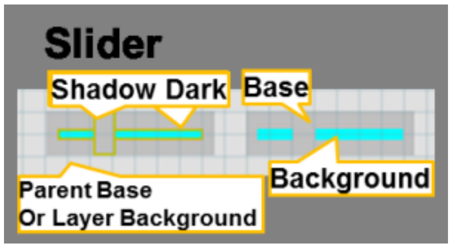

Rectangle
Highlight Light
Base
Shadow
Parent Base
Or Layer Background

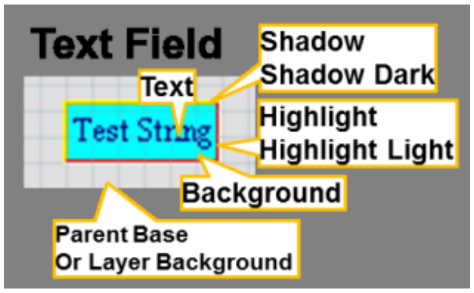

Rectangle
Parent Base
Or Layer Background
Base
Foreground
Shadow Dark
Foreground

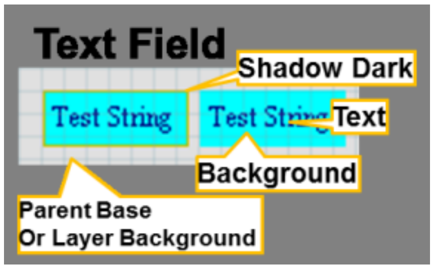

Scroll Bar
Highlight Light
Base
Shadow
Foreground
Parent Base
Or Layer Background
Background

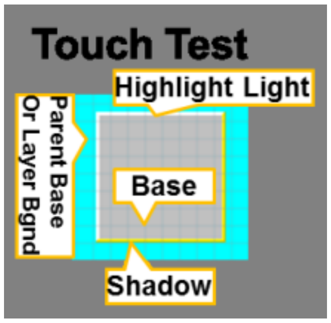

Scroll Bar
Shadow Dark
Foreground
Parent Base
Or Layer Background
Background

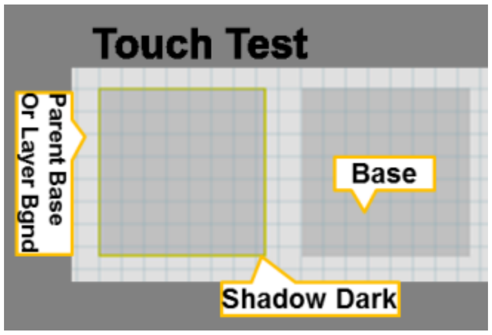

Slider
Shadow Dark
Shadow
Highlight
Light
Base
Highlight
Highlight Light
Shadow
Parent Base
Or Layer Background
Background

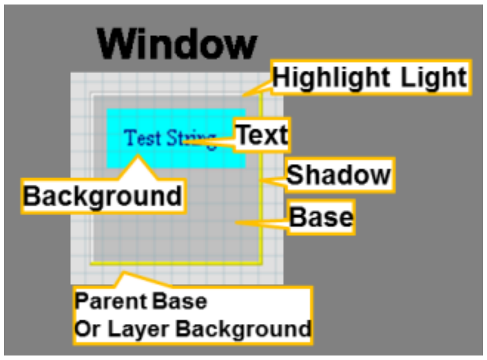

Slider
Shadow Dark
Base
Parent Base
Or Layer Background
Background

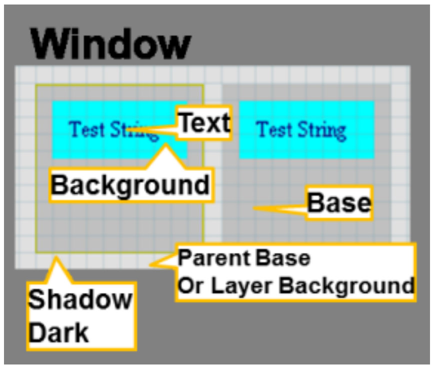

Text Field
Shadow
Shadow Dark
Text
Test String
Highlight
Highlight Light
Background
Parent Base
Or Layer Background


Text Field
Shadow Dark
Test String
Test String
Text
Background
Parent Base
Or Layer Background


Touch Test
Highlight Light
Parent Base Or Layer Bgnd
Base
Shadow


Touch Test
Parent Base Or Layer Bgnd
Base
Shadow Dark


Window
Highlight Light
Test String
Text
Shadow
Background
Base
Parent Base Or Layer Background


Window
Test String
Text
Test String
Background
Base
Parent Base Or Layer Background
Shadow Dark

Code Generation

This topic describes using the graphics composer to generate code.

Description

MPLAB Harmony Graphics Composer data is generated the same way as the rest of the project within MHC through the Generate button.

libaria_harmony.h/c – These files provide the interface that binds libaria to the overall MPLAB Harmony framework. They contain the implementations for the standard state management, variable storage, and initialization and tasks functions. If the touch functionality is enabled then the touch bindings are also generated in `libaria_harmony.c`.

libaria_init.h/c - These files contain the main initialization functions for the library state and screens. The header file contains all predefined information for the library state including screen IDs, schemes, and widget pointers. The main initialization function initializes all schemes and screens, creates all screen objects, and sets the initial state of the library context. As each screen must be capable of being created at any time, each screen has a unique create function that can be called at any time by the library. The `libaria_init.c` file contains these create functions.

libaria_events.h/c – The event files contain the definitions and implementations of all enabled MHGC events. Each event implementation will contain all generated actions for that event.

libaria_macros.h/c – The macro files contain the definitions and implementations of all defined MHGC screen macros. A macro is similar to an event in that it can contain actions. However, it is meant to be called from an external source such as the main application.

libaria_config.h – This file contains configuration values for the library. These are controlled through settings defined in the MHC settings tree.

gfx_display_def.c – This file contains generated definitions for enabled graphics displays.

gfx_driver_def.c – This file contains generated definitions for enabled graphics drivers.

gfx_processor_def.c – This file contains generated definitions for enabled graphics processors.

gfx_assets.h/c – These files contain generated asset data.

Advanced Topics

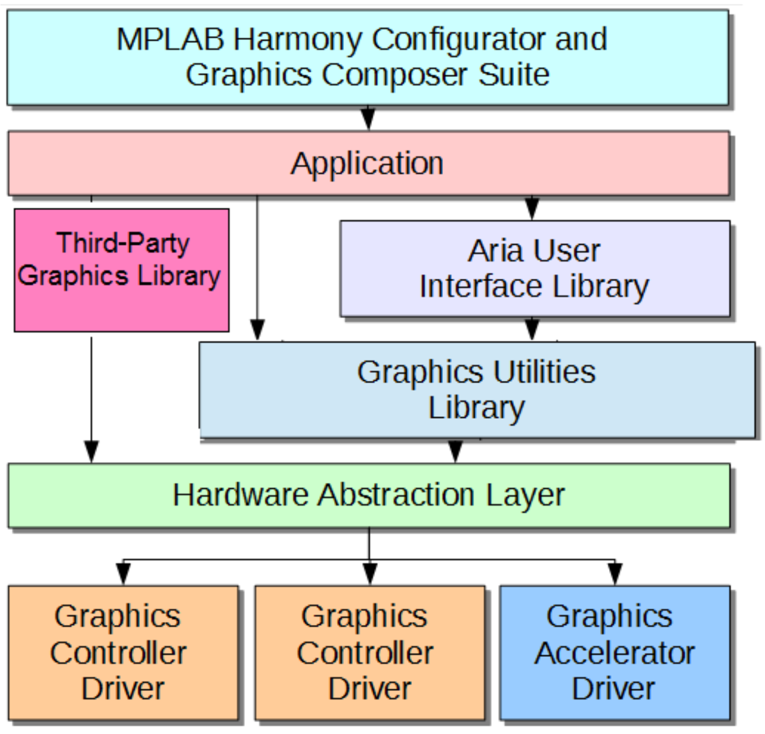
This section provides advanced information topics for MHGC.

Adding Third-Party Graphics Products Using the Hardware Abstsraction Layer (HAL)

This topic provides information on using the Hardware Abstraction Layer (HAL) to add third-party graphics products.

Description

The architecture of the MPLAB Harmony Graphics Stack is shown in the following diagram.


MPLAB Harmony Configurator and Graphics Composer Suite
Application
Third-Party Graphics Library
Aria User Interface Library
Graphics Utilities Library
Hardware Abstraction Layer
Graphics Controller Driver
Graphics Controller Driver
Graphics Accelerator Driver

Hardware Abstraction Layer (HAL)

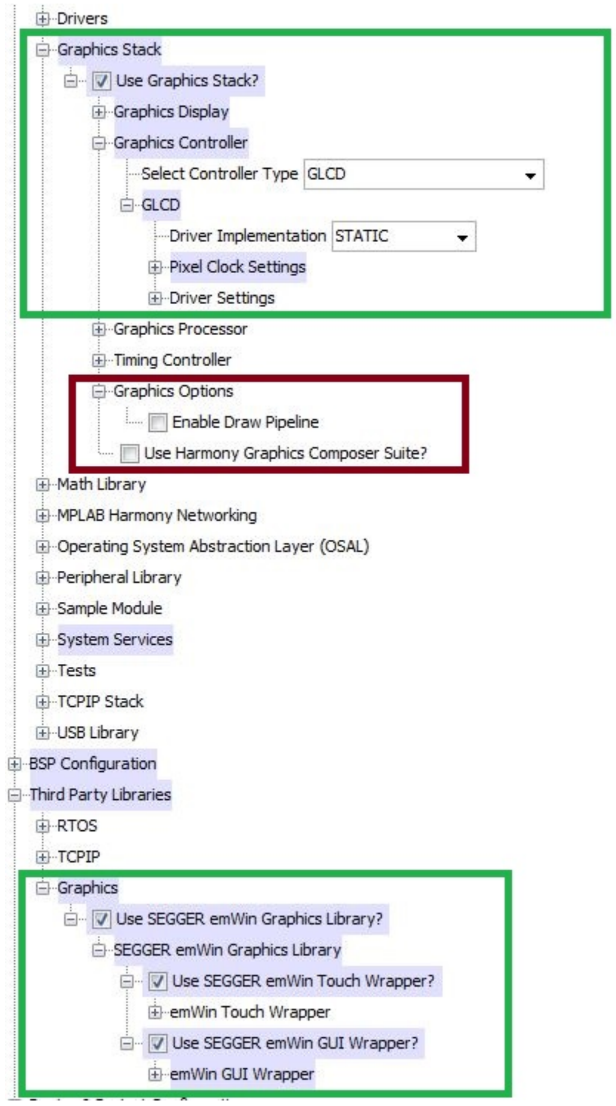
The HAL is a software layer that serves as a gatekeeper for all graphics controller and accelerator drivers. This layer is configured at initialization by the underlying graphics drivers and provides functionality such as buffer management, primitive shape drawing, hardware abstraction, and draw state management. This layer serves as a means of protection for the drivers, frame buffers, and draw state in order to prevent state mismanagement by the application.

Third-Party Graphics Library

The third-party graphics library can be used with the MPLAB Harmony framework to perform the graphics operations desired by the application. The third-party library has access to the HAL, which has been configured to service the frame buffer which is filled by the third-party graphics library.

The third-party graphics library can access the MPLAB Harmony framework drivers such as touch drivers, graphics controller driver, and display driver through the HAL. The draw pipeline and the user interface (UI) design files come from the third-party graphics library. The third-party graphics library needs the frame buffer location to fill the frame buffer with the pixel values. Or, in case of external controllers, it would need a function to access the controller drivers to output pixels on the display. The HAL provides the third-party graphics library with the frame buffer location or the API to communicate the pixel values to the external controllers.

The following figure from the MPLAB Harmony Configurator (MHC), shows the selections made in the Graphics Stack to enable the needed graphics display and controller features. Note that the Draw Pipeline for the MPLAB Harmony Graphics Stack has been disabled to assure that the third-party graphics alone is taking effect. The MPLAB Harmony Graphics Configurator (MHGC) is also not enabled, as the design tools from the third-party graphics library are used to generate the UI graphics. The `LCDConf.c` file has appropriate APIs for the third-party graphics library to communicate through the HAL with the display drivers and the framebuffer.



```
  Drivers
  Graphics Stack
    [✓] Use Graphics Stack?
      Graphics Display
      Graphics Controller
        Select Controller Type  GLCD
        GLCD
          Driver Implementation  STATIC
          Pixel Clock Settings
          Driver Settings
      Graphics Processor
      Timing Controller
      Graphics Options
        [ ] Enable Draw Pipeline
      [ ] Use Harmony Graphics Composer Suite?
  Math Library
  MPLAB Harmony Networking
  Operating System Abstraction Layer (OSAL)
  Peripheral Library
  Sample Module
  System Services
  Tests
  TCPIP Stack
  USB Library
BSP Configuration
Third Party Libraries
  RTOS
  TCPIP
  Graphics
    [✓] Use SEGGER emWin Graphics Library?
      SEGGER emWin Graphics Library
        [✓] Use SEGGER emWin Touch Wrapper?
          emWin Touch Wrapper
        [✓] Use SEGGER emWin GUI Wrapper?
          emWin GUI Wrapper
```

Example Demonstration Project

The Aria demonstration project, emwin_quickstart, has three configurations. Each configuration has an API named LCD_X_Config, which is generated with the relevant calls for SEGGER emWin to communicate with the display driver and obtain the frame buffer location pointer to write the pixel data to it. For PIC32MZ DA and PIC32MZ EF configurations, the frame buffer pointer address is provided to SEGGER emWin by the HAL. For the S1D controller on PIC32MX devices (pic32mx_usb_sk2_s1d_pictail_wqvga), The pixel write function pointers are assigned to the appropriate S1D driver APIs, which allow SEGGER emWin to write to the display controller.

Speed and Performance of Different Image Decode Formats in MHGC

Provides information and recommendations for image decode formats.

Description

MHGC supports various image formats and the MHGC Image Assets Manager provides the ability to convert and store a source image into to the following formats

- Bitmap RAW
- Bitmap Raw Run-Length Encoded (RLE)
- JPEG
- PNG
- Predecoded RAW Bitmap in DDR (PIC32MZ DA)

The following table shows the relative rendering time and Flash memory requirements of the different image formats in the MPLAB Harmony Graphics Library. The rendering time includes decoding the image and drawing it to the screen. This information is helpful when optimizing a MPLAB Harmony graphics project for performance and/or Flash memory space. For example, as shown by the red highlighted text in the table, a 40x40 pixel 16-bit RAW image renders 2.38 times faster and uses 2.59 times more Flash space than a JPEG image.

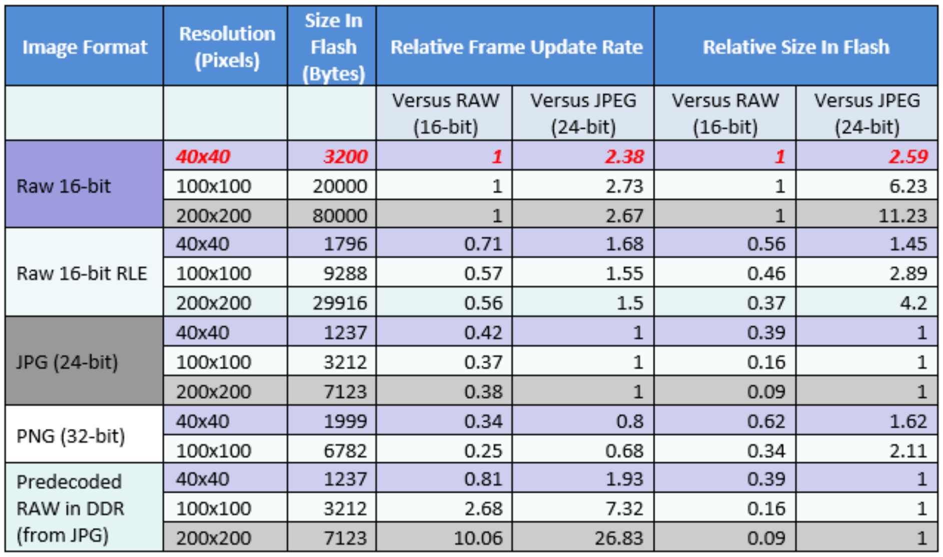


Image Format	Resolution (Pixels)	Size In Flash (Bytes)	Relative Frame Update Rate		Relative Size In Flash	
			Versus RAW (16-bit)	Versus JPEG (24-bit)	Versus RAW (16-bit)	Versus JPEG (24-bit)
Raw 16-bit	*40x40*	*3200*	*1*	*2.38*	*1*	*2.59*
	100x100	20000	1	2.73	1	6.23
	200x200	80000	1	2.67	1	11.23
Raw 16-bit RLE	40x40	1796	0.71	1.68	0.56	1.45
	100x100	9288	0.57	1.55	0.46	2.89
	200x200	29916	0.56	1.5	0.37	4.2
JPG (24-bit)	40x40	1237	0.42	1	0.39	1
	100x100	3212	0.37	1	0.16	1
	200x200	7123	0.38	1	0.09	1
PNG (32-bit)	40x40	1999	0.34	0.8	0.62	1.62
	100x100	6782	0.25	0.68	0.34	2.11
Predecoded RAW in DDR (from JPG)	40x40	1237	0.81	1.93	0.39	1
	100x100	3212	2.68	7.32	0.16	1
	200x200	7123	10.06	26.83	0.09	1

Predecoded Images in DDR (RAW)

For PIC32MZ DA devices with DDR, the MHGC Image Asset Manager provides an option to predecode images from Flash and store them into DDR as RAW images. The GPU is used to render the decoded image from DDR to the frame buffer. This provides a faster render time than an equivalent RAW image in Flash memory, specifically for large images (up to 10 times faster for a 200x200 image). Conversely, predecoding small images 40x40 pixels or smaller in DDR may not render faster due to the additional overhead of setting up the GPU.

Recommendations:

- If there is adequate DDR memory available, consider predecoding images to DDR for best performance
- Using JPEG images and predecoding them into DDR can provide the best rendering performance and most Flash memory savings.

Note: The images are decoded from Flash to DDR memory by the Graphics Library during initialization and may introduce delay at boot-up, depending on the number and size of the images.

RAW Images

RAW images provide fast rendering time, as there is no decoding needed. However, depending on image content, it can be two times larger than a Run-Length Encoded (RLE) image and about 3 to 10 times larger than a JPEG.

Recommendation:

For small images that are to be rendered frequently, consider using a RAW image for better performance

JPEG Images

JPEG images provide the most Flash space savings, but are slower to render compared to RAW and RAW RLE.

Recommendations:

- If images are large and not used frequently, consider using the JPEG image format to save flash memory space
- If DDR memory is available, consider predecoding JPEG images in DDR for better rendering performance

Run-Length Encoded RAW Images

In terms of rendering speed and size, RAW RLE images are in between RAW and other compressed formats like JPEG or PNG. Depending on the image contents, RAW RLE can be approximately 1.5 times faster than JPEG, but could be significantly larger in size for large images. Again, depending on the image content, RAW RLE can be about half the size and performance of a RAW image.

Recommendation:

If optimizing your application for both speed and flash size consider using RAW RLE images

PNG Images

Among the image formats, PNG is slowest to render and requires more memory to decode.

Recommendations:

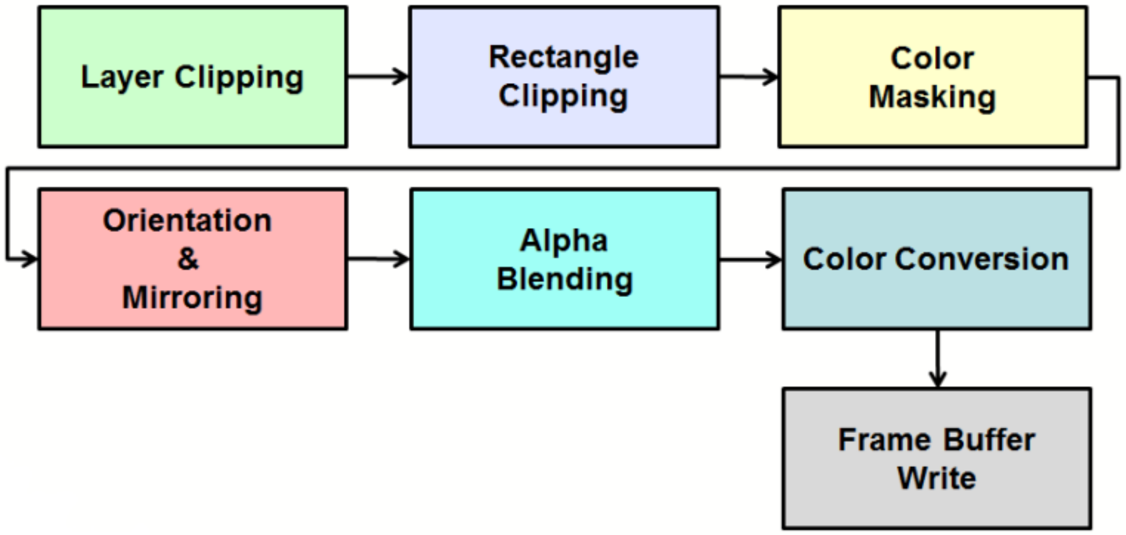
- Unless fine levels of alpha-blending are needed, it is better to use other image formats to achieve the best performance. Use the MHGC Asset Manager to convert the source PNG image and store it in a different image format.
- If you would like to use an image with a transparent background, it may be better to use a RAW RLE image with background color masking to achieve the same effect with better performance than a PNG. Color masking is supported in the MHGC Image Asset Manager.

Draw Pipeline Options

This section details how to use the Graphics Pipeline.

Description

The nominal rendering pipeline for an image is shown in the following figure.



The order of rendering for other widgets may differ. For example, for a colored rectangle the color mask is first checked. If the rectangle's fill matches the mask color defined then there is nothing to draw.

Graphics Pipeline

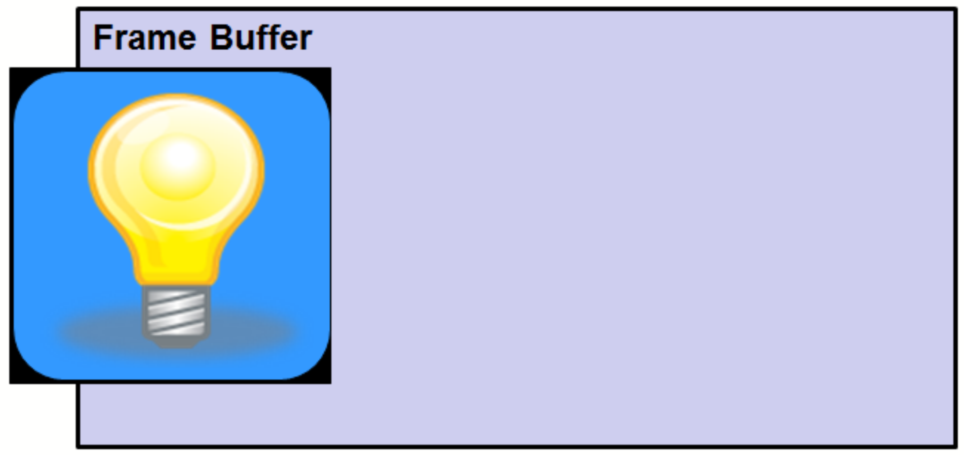
Provides information on the graphics pipeline.

Description

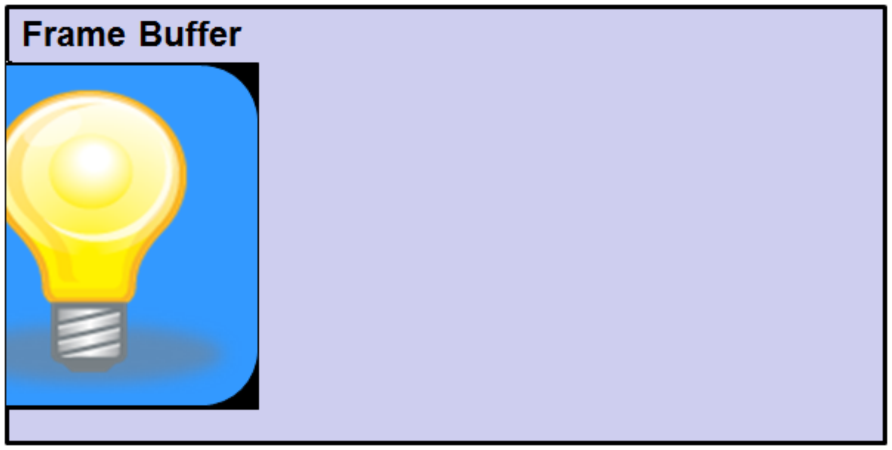
Layer Clipping

In order of the processing, Layer Clipping is first applied to the image. If the image extends beyond the edges of the layer that contains it then those pixels are not drawn. Failure to clip out-of-bound pixels can cause the application to crash. The following figures shows an example of layer clipping:

Before applying layer boundaries:



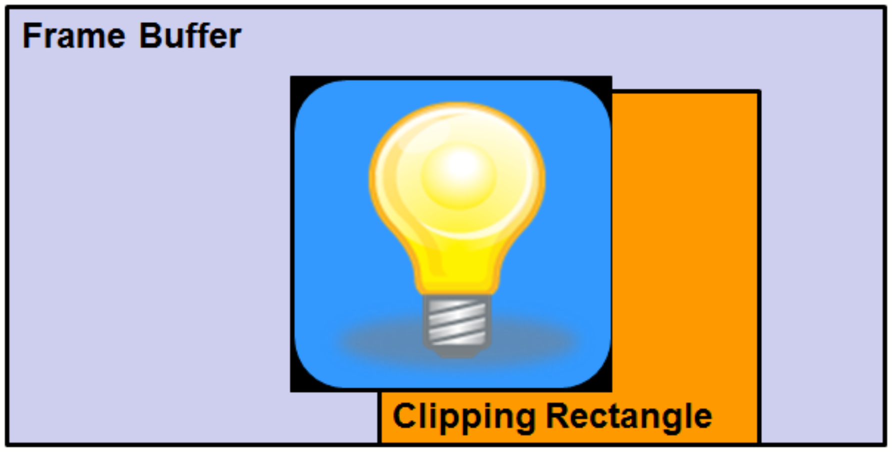
After applying layer boundaries:



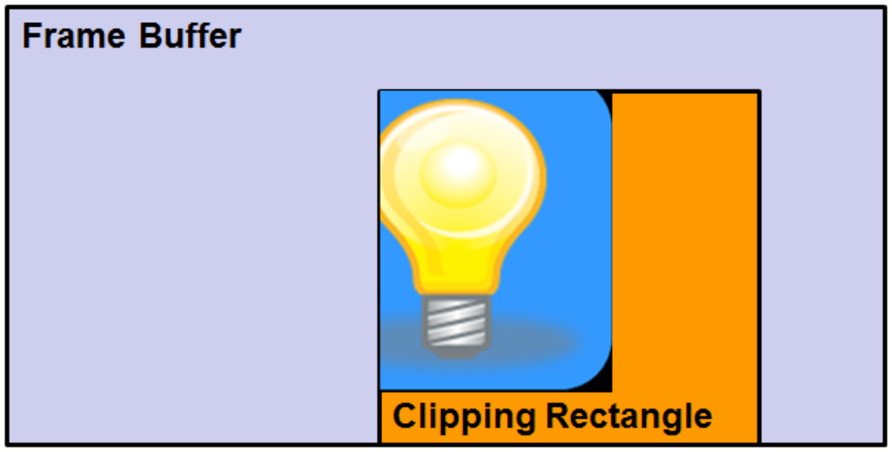
Rectangle Clipping

Next, the image is clipped to the boundaries of any widgets that contain it as a parent, such as a rectangle.

Before applying the clipping rectangle.:



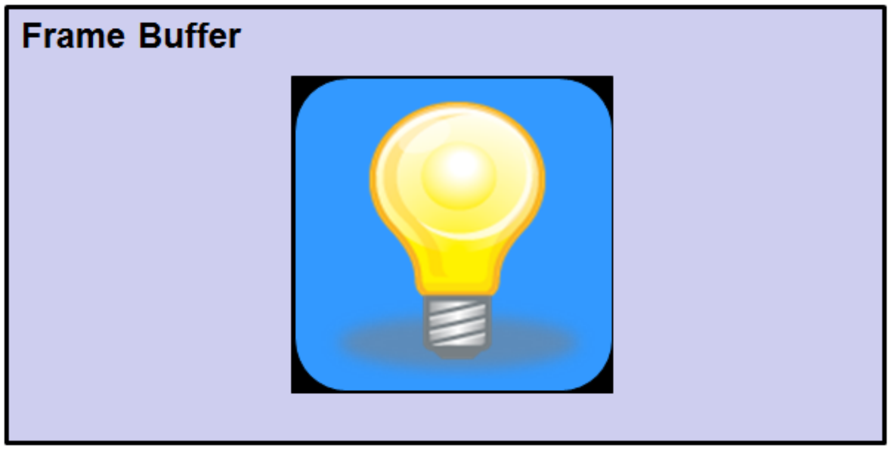
After applying the clipping rectangle:



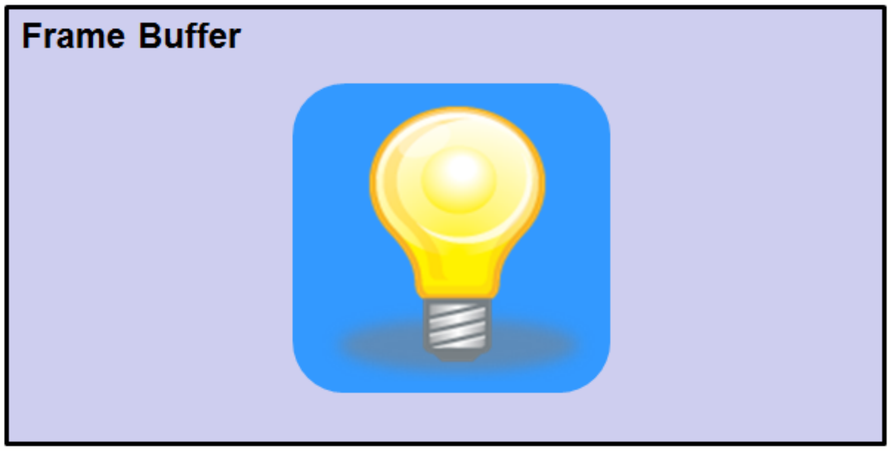
Color Masking of Pixels

Pixels in the image are matched to a mask color. If the colors match the pixel is discarded (not drawn). In the following example, the black border of the image is removed by defining the mask color to be black.

Before applying color mask:

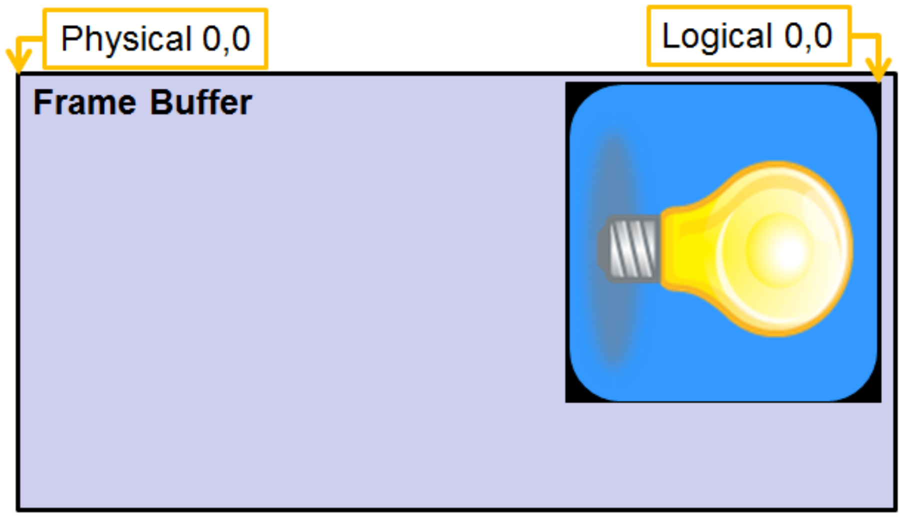


After applying color mask:

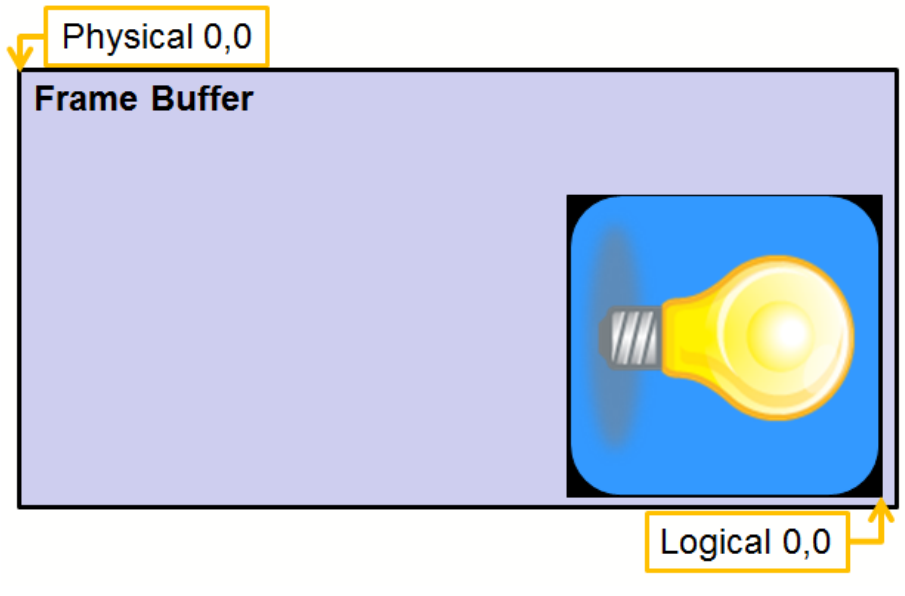


Orientation and Mirroring

The logical orientation of the graphics design may not match the physical layout of the display. Pixels may need to be reoriented from logical to physical space before being rendered.



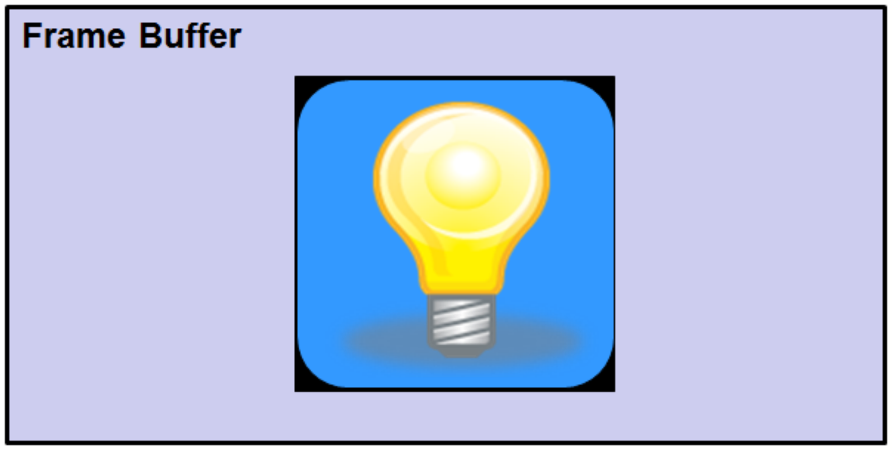
Pixels may also need to be flipped (mirrored) before being rendered.



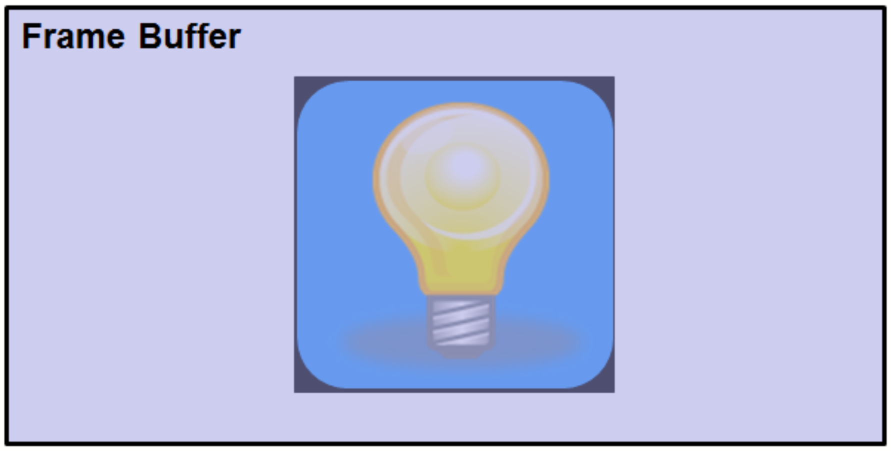
Alpha Blending

Each pixel drawn is a composite of the image color and the background color based on the alpha blend value defined by a global alpha value, the pixels alpha value, or both.

Before alpha blending:

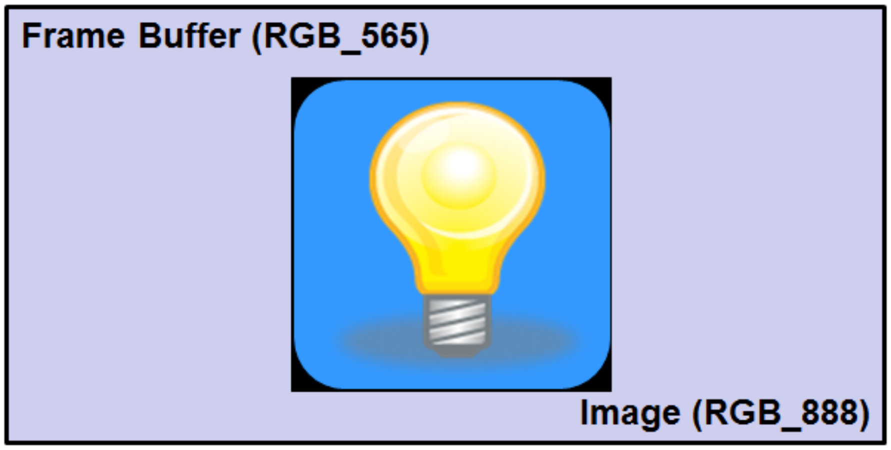


After alpha blending:



Color Conversion

The image color format may not be the same as the destination frame buffer. Each pixel must be converted before it is written. In the following example, the image is stored using 24 bits per pixel; however, the frame buffer uses 16 bits per pixel.



Frame Buffer Write

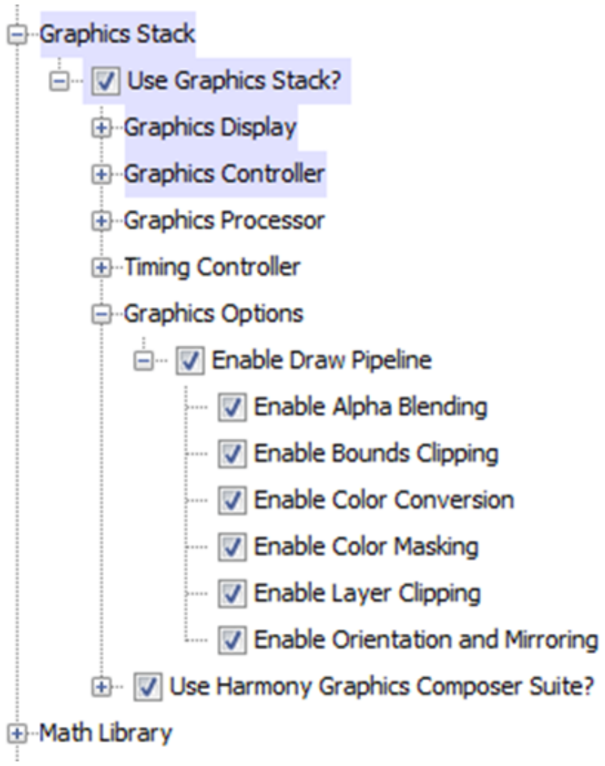
The final stage in rendering an image is to write each-color converted pixel to the frame buffer.

Graphics Pipeline Options

Provides information graphics pipeline options.

Description

Each stage in the graphics pipeline adds overhead to the rendering. Stages can be removed from processing using MPLAB Harmony Configurator (MHC) options for the Draw Pipeline, found by selecting *MPLAB Harmony Framework Configuration > Graphics Stack*.


- Graphics Stack
  - ✓ Use Graphics Stack?
    - Graphics Display
    - Graphics Controller
    - Graphics Processor
    - Timing Controller
    - Graphics Options
      - ✓ Enable Draw Pipeline
        - ✓ Enable Alpha Blending
        - ✓ Enable Bounds Clipping
        - ✓ Enable Color Conversion
        - ✓ Enable Color Masking
        - ✓ Enable Layer Clipping
        - ✓ Enable Orientation and Mirroring
    - ✓ Use Harmony Graphics Composer Suite?
- Math Library

For example, the Alpha Blending stage can be disabled if your graphics application does not use alpha blending. If the color mode of the display matches the color mode of all images you can disable Color Conversion. Disabling unneeded stages can improve performance and reduce code size.

Also, a graphics controller driver may add additional stages, or opt to bypass stages completely depending on the capabilities of the graphics hardware supported by the driver.

Improved Touch Performance with Phantom Buttons

This topic provides information on the use of phantom buttons to improve touch performance.

aria_coffeemaker Demonstration Example

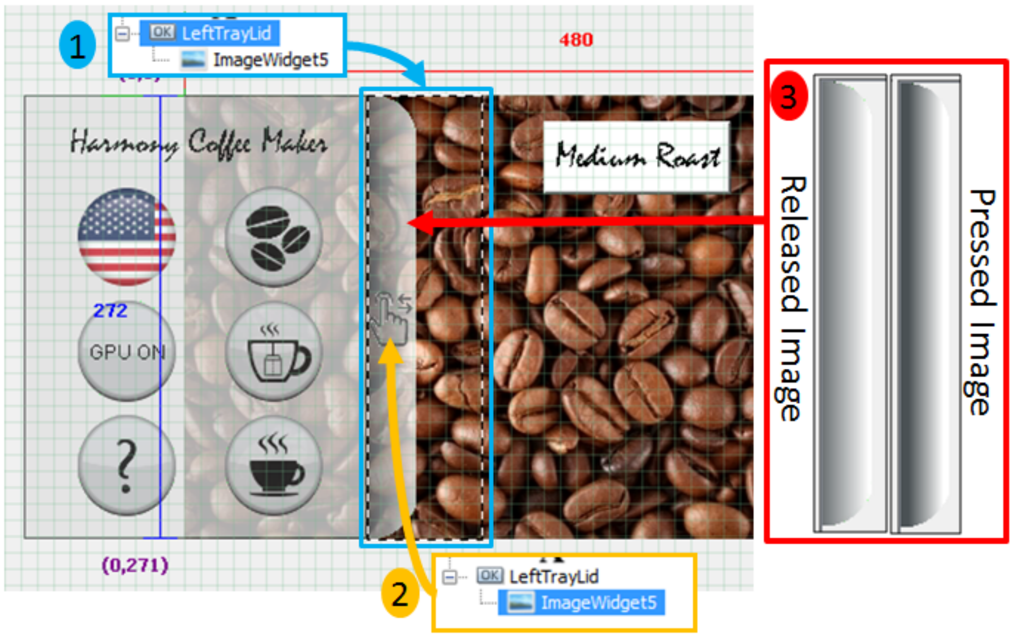
Provides image examples with buttons in the aria_coffeemaker demonstration.

Description

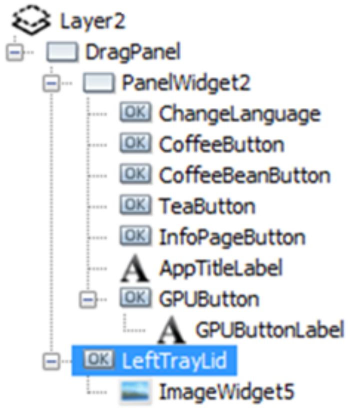
Small buttons are hard to activate on the screen. The use of phantom (invisible) buttons can improve touch performance without increasing the size of the visible footprint of the button on the display.

The aria_coffee_maker has a sliding tray on each side of the display. Sliding a tray in, or out, is accomplished by a phantom (invisible) button. Looking at the left tray, we see the three parts of this phantom button.

1. *LeftTrayLid*: An invisible button widget, whose outline is shown in blue. This area is the touch field.
2. *ImageWidget5*: An image widget containing a hand icon, providing a visual clue as to how to manipulate the tray.
3. *The Release Image and Pressed Image*: These are defined as part of the button widget properties. The Pressed Image has a darker coloring than the Released Image. This difference is what shows the user that the button has been pressed.

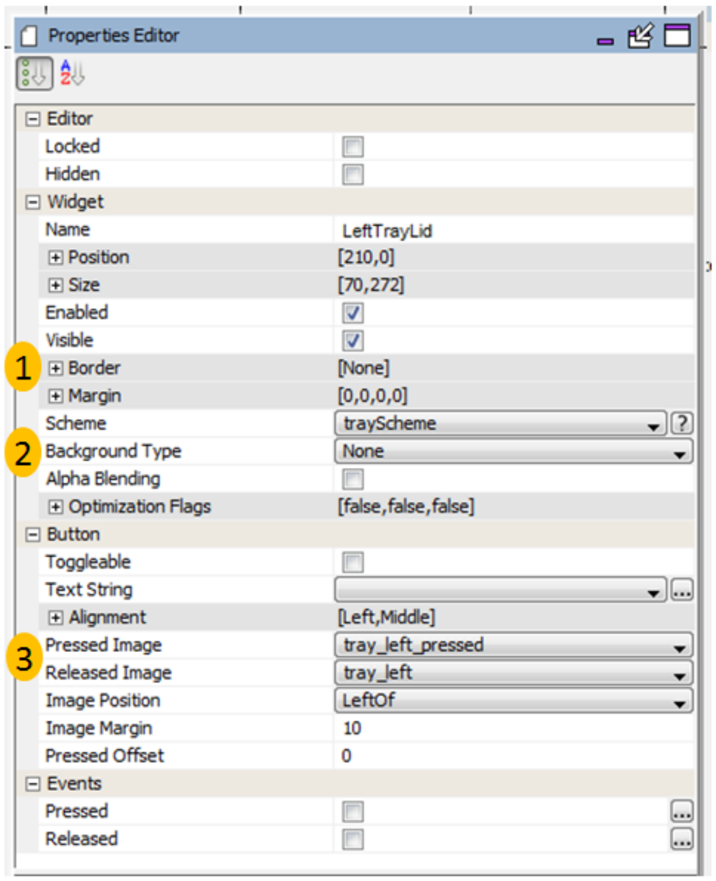


The drawing hierarchy for this part of the design is shows that ImageWidget5 is a daughter widget to the LeftTrayLid button widget.



Examining the properties of the LeftTrayLid button widget reveals more about how this works. The following figure demonstrates these three properties.

1. The *Border* is defined as *None*.
2. *Background Type* is defined as *None*.
3. The different images used will show when the button is *Pressed* or *Released*.



Editor	
Locked	☐
Hidden	☐
Widget	
Name	LeftTrayLid
Position	[210,0]
Size	[70,272]
Enabled	✓
Visible	✓
Border	[None]
Margin	[0,0,0,0]
Scheme	trayScheme
Background Type	None
Alpha Blending	☐
Optimization Flags	[false,false,false]
Button	
Toggleable	☐
Text String	
Alignment	[Left,Middle]
Pressed Image	tray_left_pressed
Released Image	tray_left
Image Position	LeftOf
Image Margin	10
Pressed Offset	0
Events	
Pressed	☐
Released	☐

By setting the border and background to *None*, the button is invisible. Only by providing different images for *Released* versus *Pressed* does the user know when the button has been pressed.

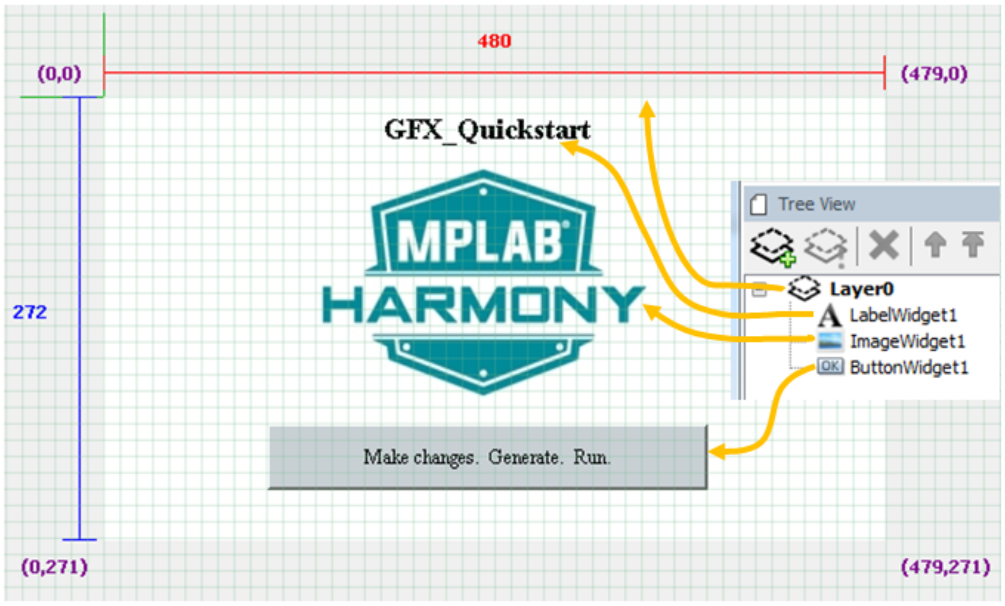
The actual touch region defined by the button is much larger than the images shown on the display. This extra area increases the touch response of the display.

Small Buttons Controlled by Phantom Buttons

Provides information on phantom button control of small buttons.

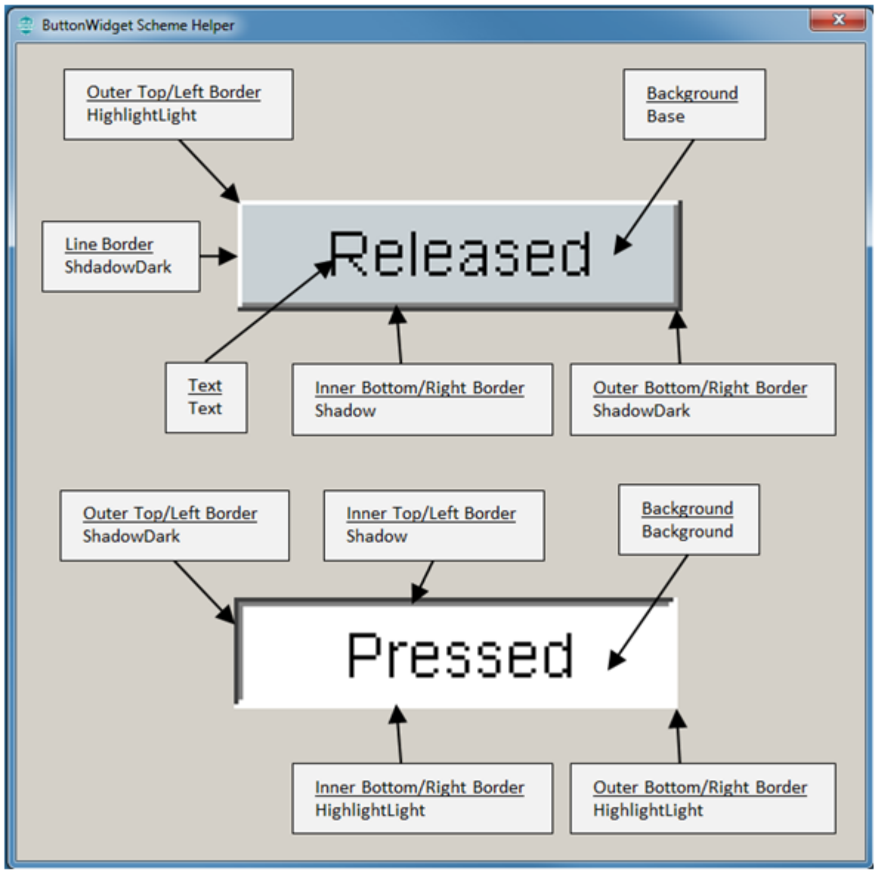
Description

When the border is not set to *None*, and the background is not set to *None*, the button widget provides a direct visible clue to the user when it is pressed. Which can be seen in the following figure with the button from aria_quickstart. In aria_quickstart, ButtonWidget1 has a bevel border, and a fill background.



Let's use aria_quickstart to demonstrate how to control ButtonWidget1 using a phantom button to surround it, thereby increasing touch responsiveness.

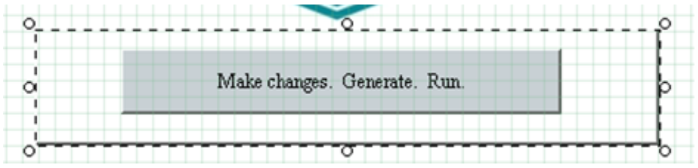
When using a bevel border and filled background, the button provides visible feedback when it is asserted.



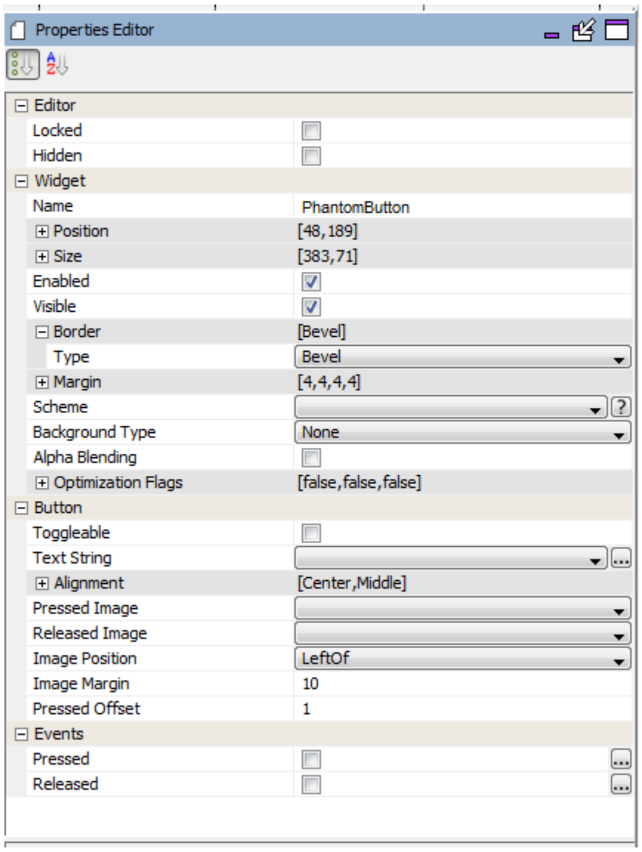
To use this feedback mechanism instead of images, there is a way to have a small button on the display, with a larger touch zone provided by another phantom button.

Steps:

1. Click on *ButtonWidget1* in the *Screen Designer* panel. Go to the *Properties Editor* panel for the widget and uncheck the *Enabled* property to disable the button. Enable *Toggleable* so that this button will have a memory.
2. Drag a new button from the Widget Tool Box panel and center it around *ButtonWidget1*. In the *Properties Editor* panel for this new button, change the name of the widget to *PhantomButton*. Change the *Background Type* to *None*. Leave the *Border* set as *Bevel* for now. The following figure displays the new button in the *Screen Designer* panel:



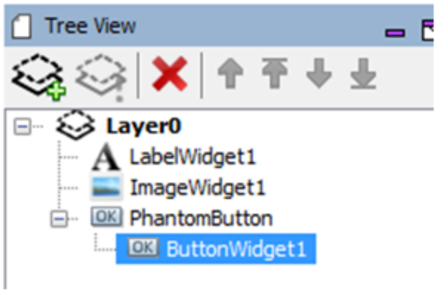
The *Properties Editor* panel should display the following information.



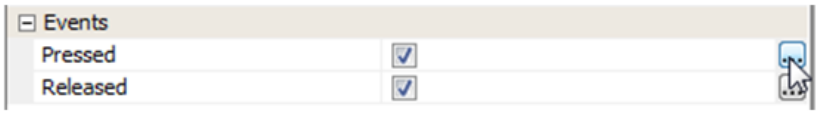
Properties Editor

⊟ Editor	
Locked	☐
Hidden	☐
⊟ Widget	
Name	PhantomButton
⊞ Position	[48,189]
⊞ Size	[383,71]
Enabled	✓
Visible	✓
⊟ Border	[Bevel]
Type	Bevel
⊞ Margin	[4,4,4,4]
Scheme	
Background Type	None
Alpha Blending	☐
⊞ Optimization Flags	[false,false,false]
⊟ Button	
Toggleable	☐
Text String	
⊞ Alignment	[Center,Middle]
Pressed Image	
Released Image	
Image Position	LeftOf
Image Margin	10
Pressed Offset	1
⊟ Events	
Pressed	☐
Released	☐

3. In the *Tree View* panel, drag *ButtonWidget1* to be a daughter widget of *PhantomWidget*. When *PhantomWidget* is moved, *ButtonWidget1* will move along with the parent.



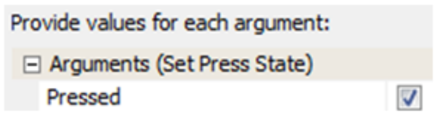
4. Click on *PhantomButton* again in the *Screen Designer panel* and move to the *Properties Editor*. Enable both the *Pressed* and *Released* events. Then click on the (…) icon to define the events. (See the following two steps.)



⊟ Events	
Pressed	✓
Released	✓

5. Defining the Pressed Event.

Click on the (…) icon. In the *Event Editor*, under *Pressed* dialog, click the *New* icon to define a new event. In the *Action Edit Dialog* that next appears, leave the selection on the template and hit the *Next* button. In the next window, select the target of the event. We want to change the state of *ButtonWidget1*, so select it and hit *Next*. The next dialog shows all the template actions that we can use to modify ButtonWidget1. Choose *Set Pressed State* and hit *Next*. Set the Argument to *Enable Pressed*. Name this event *Set Press state for ButtonWidget1* then hit Finish. Leave the Event Editor by hitting *Ok*.

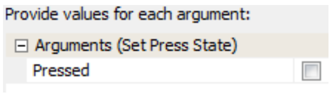

Provide values for each argument:

⊟ Arguments (Set Press State)

Pressed	✓

6. Defining the Released Event.

Click on the (…) icon. In the *Event Editor*, under *Released dialog*, click the New icon to define a new event. In the *Action Edit Dialog* that next appears, leave the selection on the template and hit the *Next* button. In the next window, select the target of the event. We want to change the state of ButtonWidget1, so select it and hit *Next*. Choose *Set Pressed State* and hit *Next*. Leave the *Argument* disabled. Name this event *Unset Press state for ButtonWidget1* then hit Finish. Leave the Event Editor by hitting *Ok*.


Provide values for each argument:

⊟ Arguments (Set Press State)

Pressed	☐

7. Generate the application from the MPLAB Harmony Configurator main menu.

8. From the MPLAB main menu, build and run the project. To verify that ButtonWidget1 does change, click outside of the original boundaries.

9. As a final step, hide the PhantomButton by changing its border to *None*. Next, Generate the code again from MHC. Finally, build and run the project from MPLAB and see how much easier it is to assert ButtonWidget1 using a phantom button.

Importing and Exporting Graphics Data

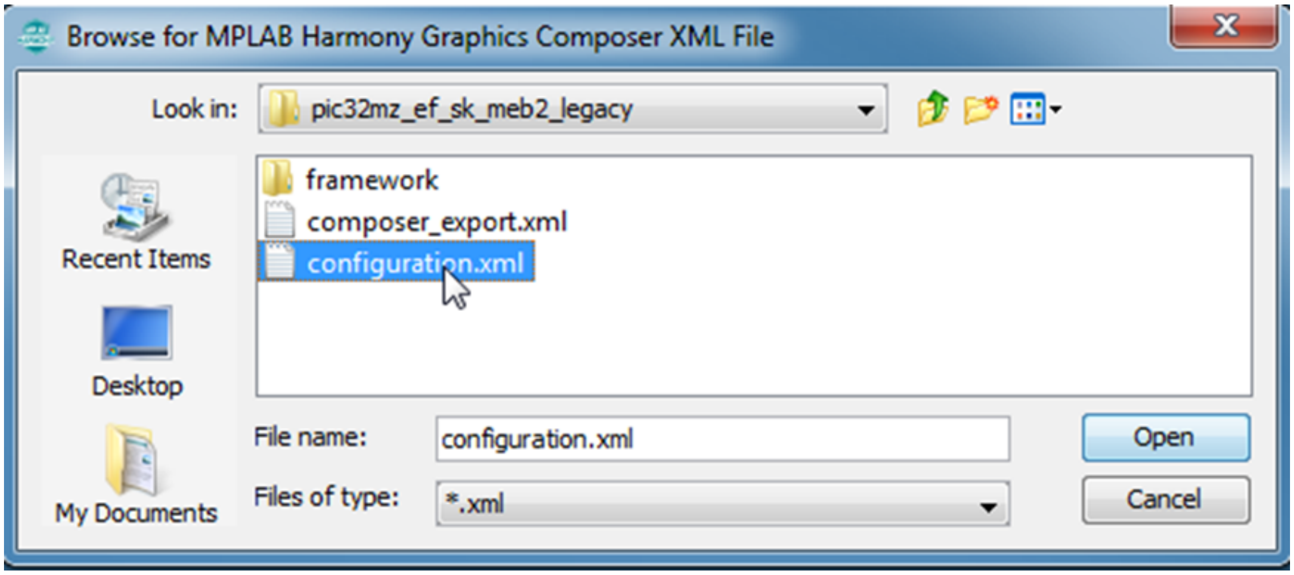
This topic provides information on importing and exporting graphics composer-related data.

Description

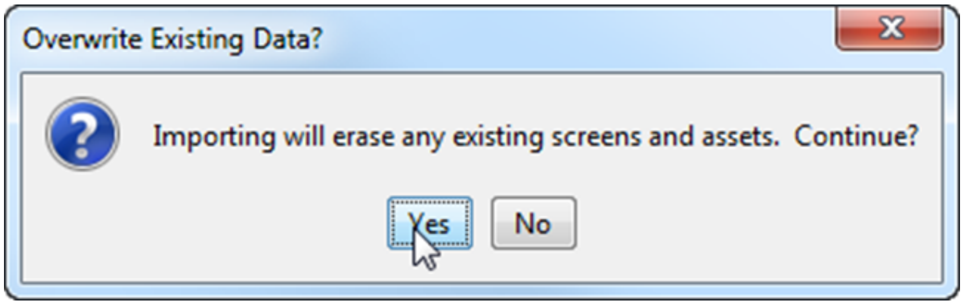
The MPLAB Harmony Graphics Composer (MHGC) provides the capability for users to import and export graphics designs. The user can export the state of an existing graphics composer configuration or import another graphics composer configuration from another project.

Importing Data

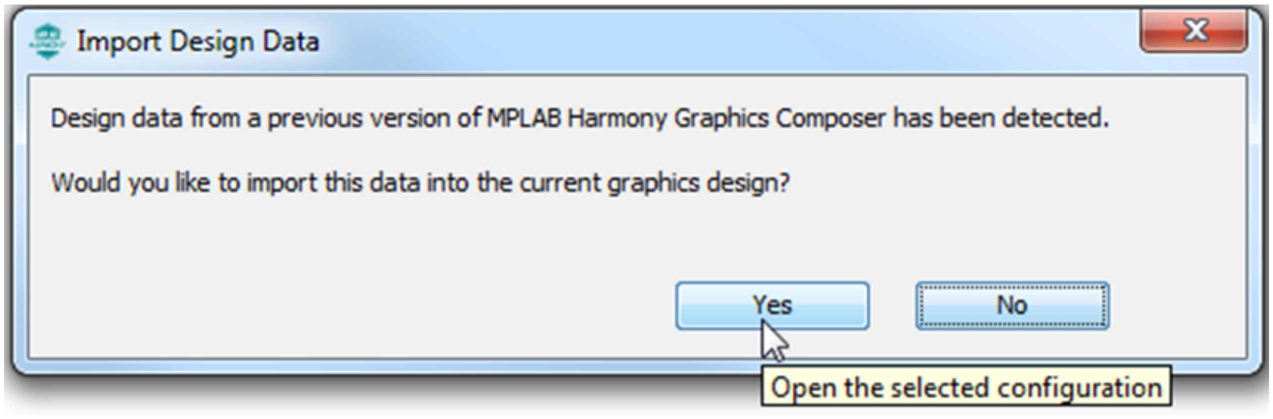
1. To import a graphics design into MHGC, select *File* > *Import*. The Browse for MPLAB Harmony Graphics Composer XML file dialog appears, which allows the selection of a previously exported Graphics Composer `.xml` file, or the `configuration.xml` file that contains the desired graphics image.


Browse for MPLAB Harmony Graphics Composer XML File
Look in: pic32mz_ef_sk_meb2_legacy
Recent Items
Desktop
My Documents
framework
composer_export.xml
configuration.xml
File name: configuration.xml
Files of type: *.xml
Open
Cancel

2. After selecting a file and clicking **Open**, you will be prompted whether to overwrite existing data.


Overwrite Existing Data?
Importing will erase any existing screens and assets. Continue?
Yes
No

3. If you selected a `composer_export.xml` file, clicking **Yes** will replace the current graphics design with the new design.
4. Otherwise, if you selected a `configuration.xml` file, you will be prompted to import the data into the current graphics design. Click **Yes** to replace the current graphics design with the new design.

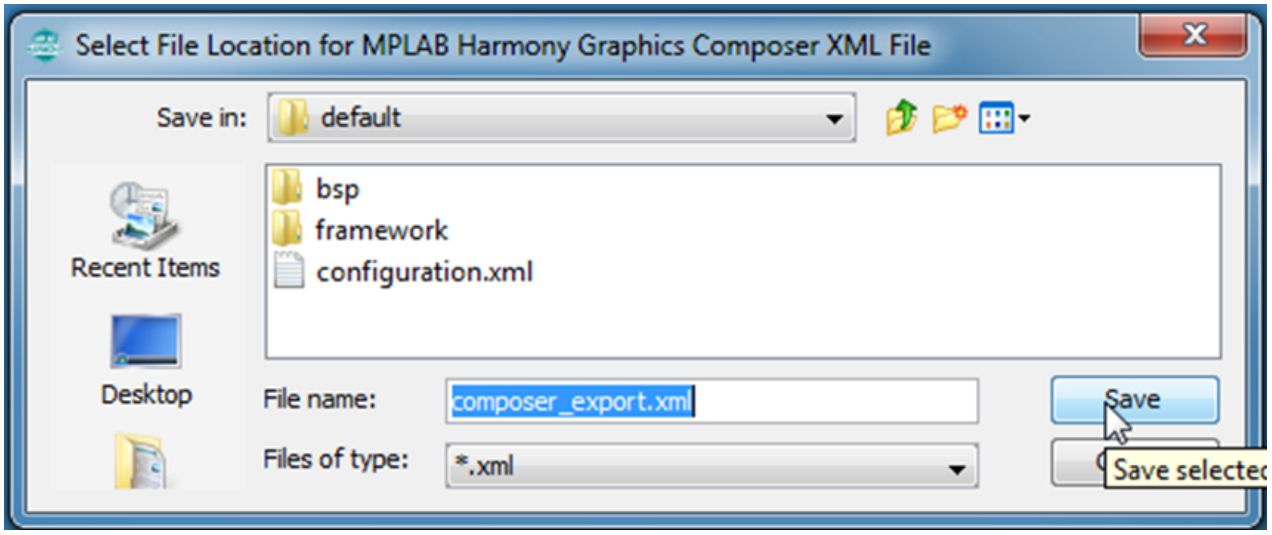

Import Design Data

Design data from a previous version of MPLAB Harmony Graphics Composer has been detected.

Would you like to import this data into the current graphics design?

Exporting Data

1. To export a graphics design from MHGC, select *File* > *Export*. The Select File Location for MPLAB Harmony Graphics Composer XML file dialog appears.


Select File Location for MPLAB Harmony Graphics Composer XML File

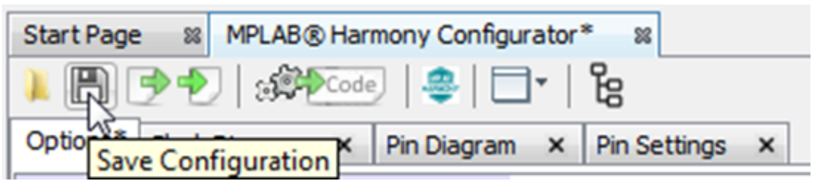
Save in: default

bsp
framework
configuration.xml

File name: composer_export.xml

Files of type: *.xml

2. To export a graphics design using a `configuration.xml` file, use the **Save Configuration** utility from the MPLAB Harmony Configurator (MHC) toolbar.



Index