MPLAB® Harmony Help - Volume II - MPLAB Harmony Configurator (MHC)

MPLAB Harmony Integrated Software Framework v1.11

Volume II: MPLAB Harmony Configurator (MHC)

This volume provides user and developer-specific information on the MPLAB Harmony Configurator (MHC).

Introduction

This topic provides an overview of the MPLAB Harmony Configurator (MHC).

Description

The MPLAB Harmony Configurator (MHC) is a MPLAB X IDE plug-in. It must be installed into your MPLAB X IDE installation to be used. See the Installing MHC section for information on installing the MHC plug-in.

MPLAB Harmony Configurator User's Guide

This section provides user information on using the MHC.

Installing MHC

This topic provides information on installing the MHC plug-in.

Description

Installing the MHC Plug-in

1. Start MPLAB X IDE and select *Tools > Plugins*.

2. Select the **Downloaded** tab and click **Add Plugins...**.

3. In the Add Plugins dialog, navigate to the MHC `com-microchip-mplab-modules-mhc.nbm` plug-in file, which is located in `<install-dir>/utilities/mhc`, and then click **Open**.
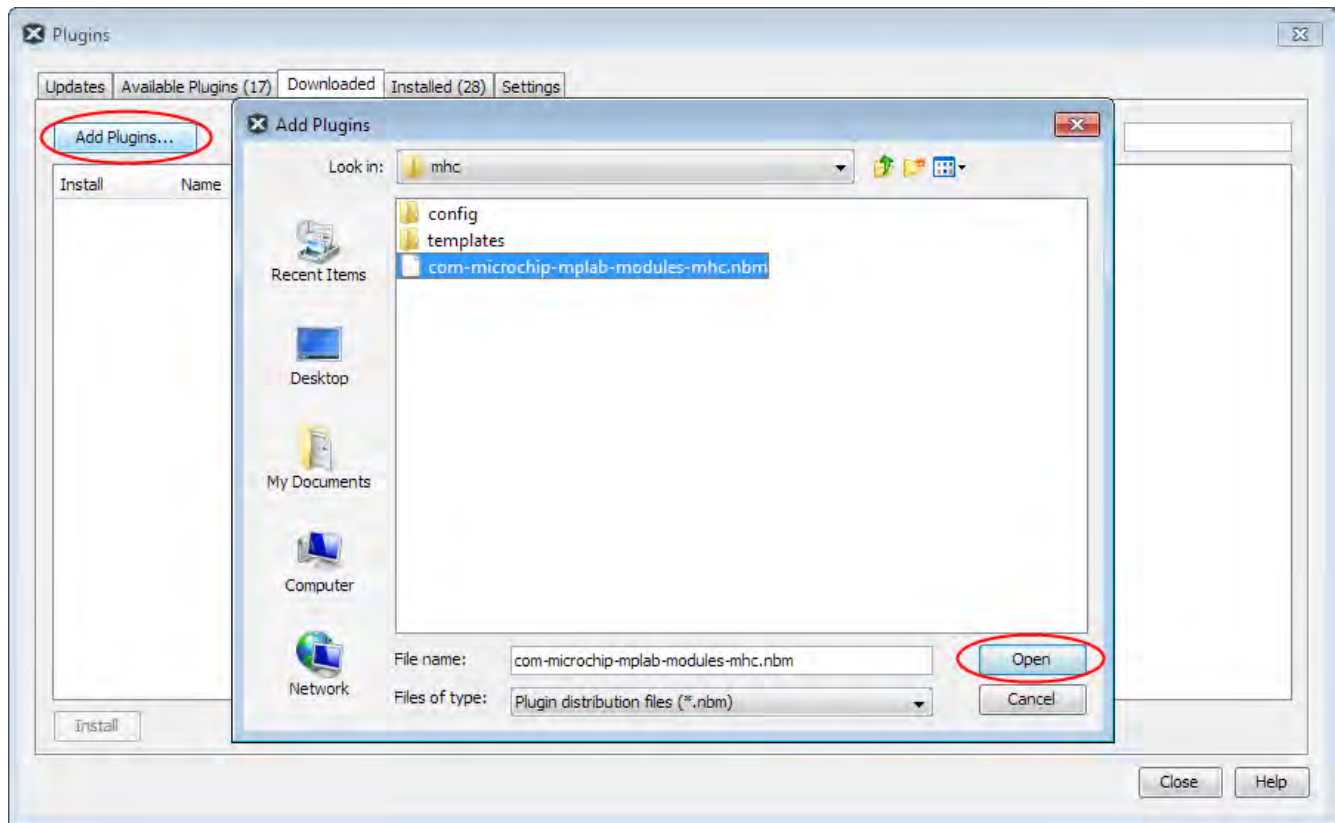
4. Ensure that the Install check box for the plug-in is selected and click **Install**.
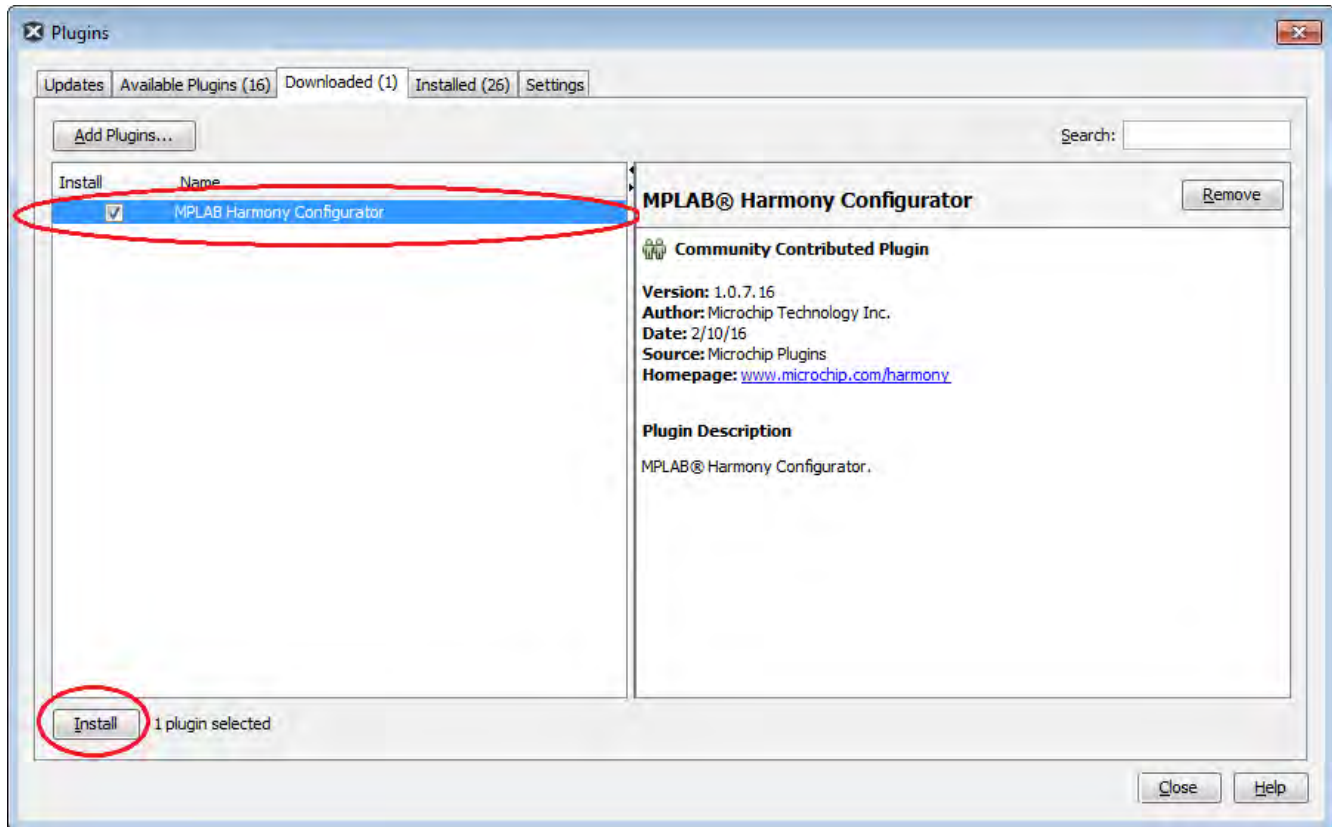
5. Follow the prompts from the installation and continue until the installation completes. (Do not be concerned if the version you are installing is *signed* but not *trusted*, simply click **Continue**). Once the installation has finished you can close the **Plugins** dialog.

6. To verify the installation, select *Tools > Plugins* and select the **Installed** tab. The MHC plug-in you installed should be included in the list.

MPLAB Harmony Configurator Interface

This section describes the MHC interface.

Description

This section provides a basic overview of the MHC user interface. For detailed information on using MHC to create a MPLAB Harmony application, refer to Using MHC to Create a New Application.

Initial Interface Configuration

The following figure shows the initial interface configuration for MHC.
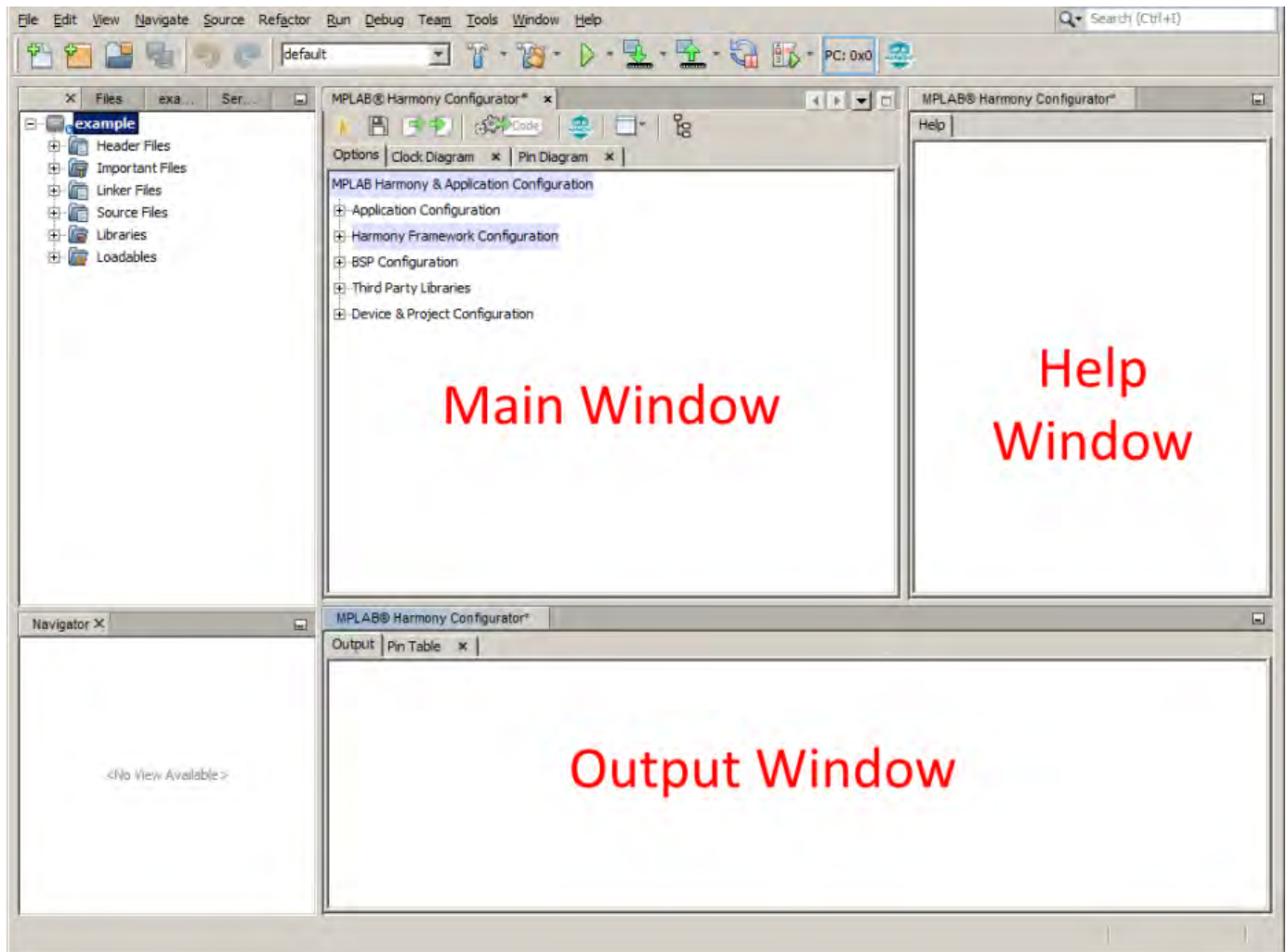
Main Window

This view shows the available configuration options for the selected Microchip device, which is arranged in a hierarchal tree structure. Click the check box to enable a specific component. The options for the enabled component will appear.

Help Window

When a tree component is interacted with, the corresponding help information is displayed in the Help Window.
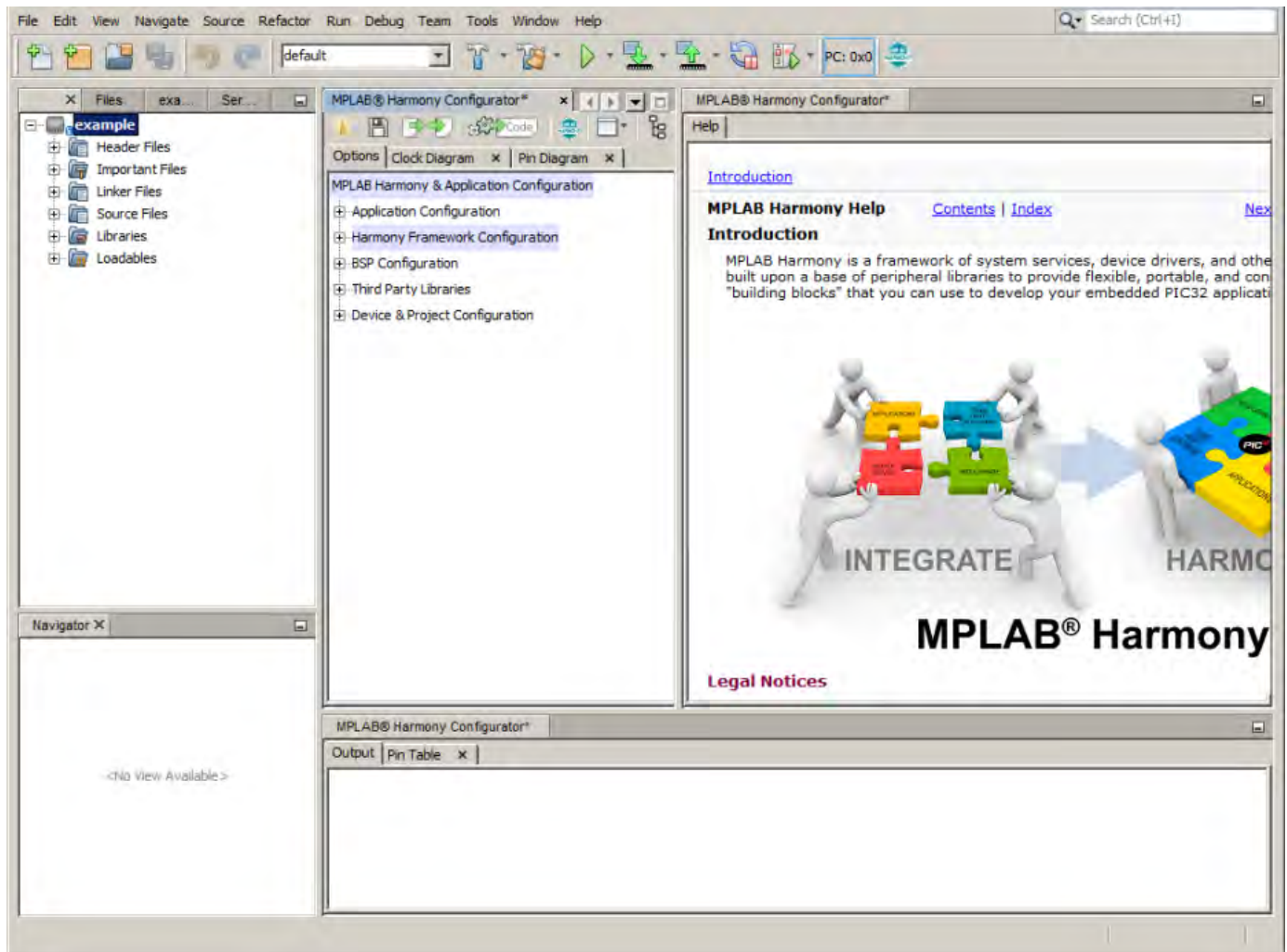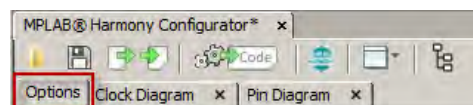
Output Window

The output window displays various log messages about the actions taken by the MPLAB Harmony Configurator.

Main Window Toolbar

The main window contains a context-sensitive toolbar. This toolbar provides both global and tab-specific functionality.
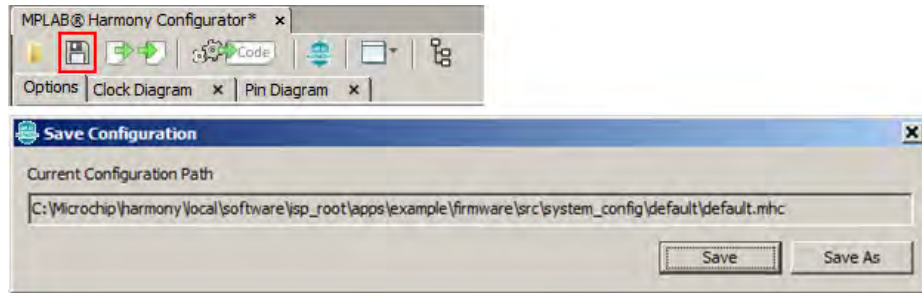
When viewing the **Options** tab, this toolbar contains the following functionality:
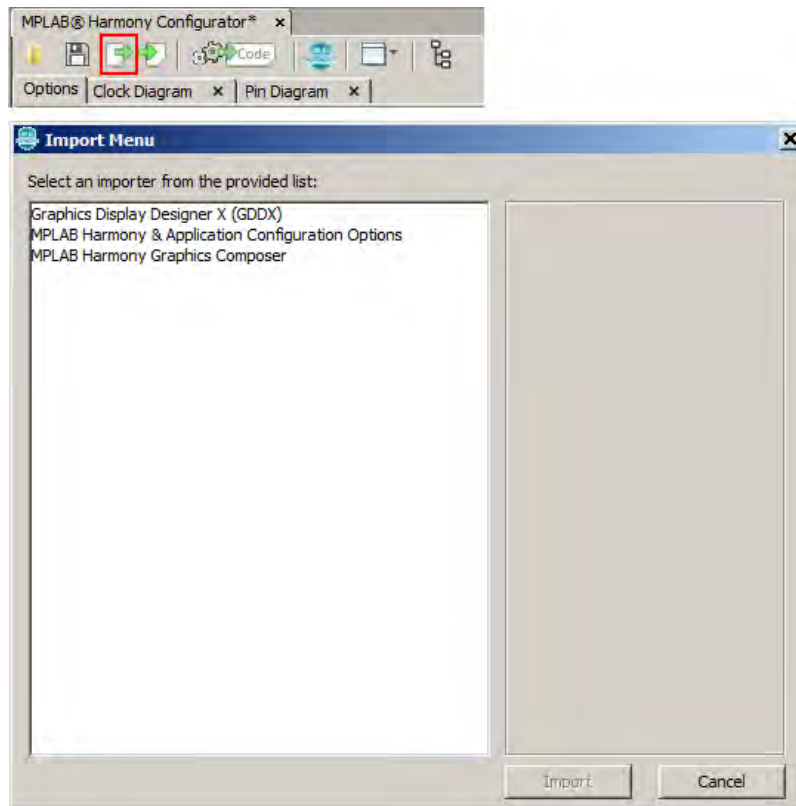

Open: Select the Open icon to open a saved `.mhc` configuration into the current Option tree.


Save: Select the Save icon to save the current Option tree into the last used `.mhc` file or click **Save As** to save to a new file.

Import: Selecting the Import icon opens the import data dialog. This dialog can be used to import different types of information into the current project.
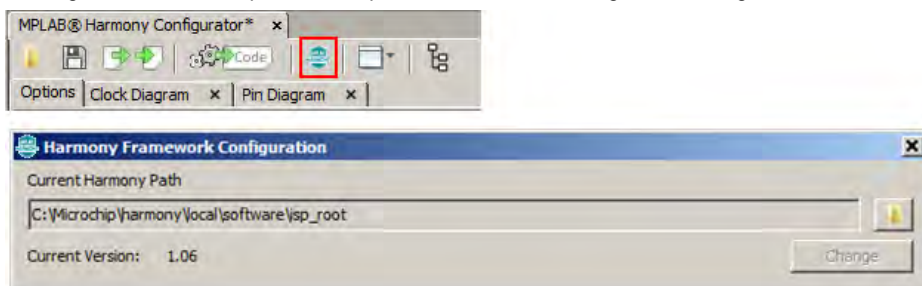

Export: Selecting the Export icon opens the export data dialog. This dialog can be used to export different types of information from the current project.
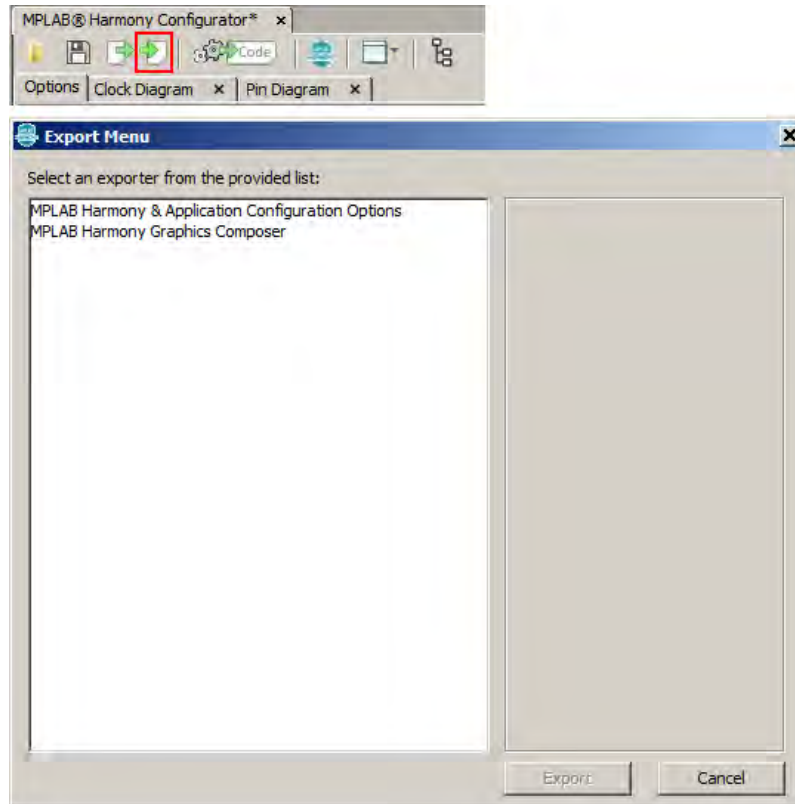
Generate: Selecting the Generate icon opens the project file generation dialog.
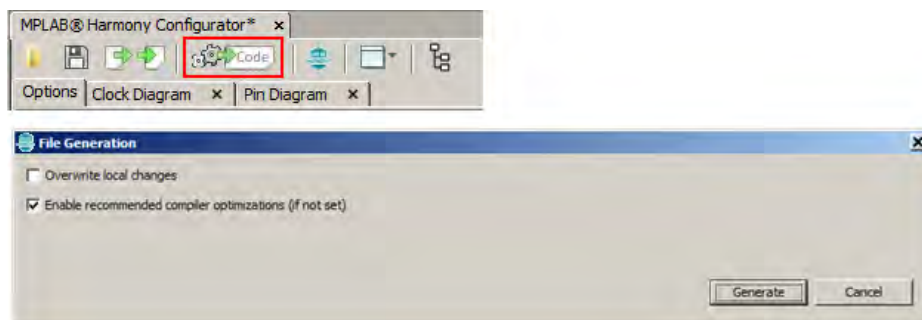
Framework Options: Selecting the Framework Options icon opens the framework configuration dialog.

Application Launcher: Selecting the Application Launcher icon provides the ability to quickly launch applications such as the clock configurator, pin configurator, or the MPLAB Harmony Graphics Composer.
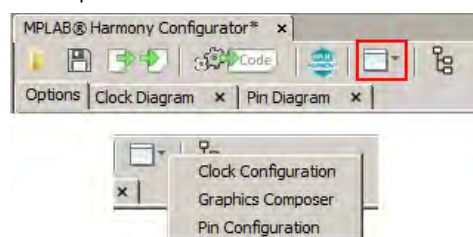
Option Tree View: Selecting the Option Tree View icon toggles the option tree between global and active view.

Project Generation

Once all of the desired options have been selected from the configuration tree, the next step is file generation, which is 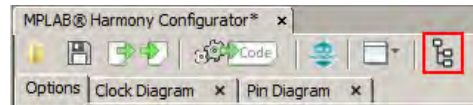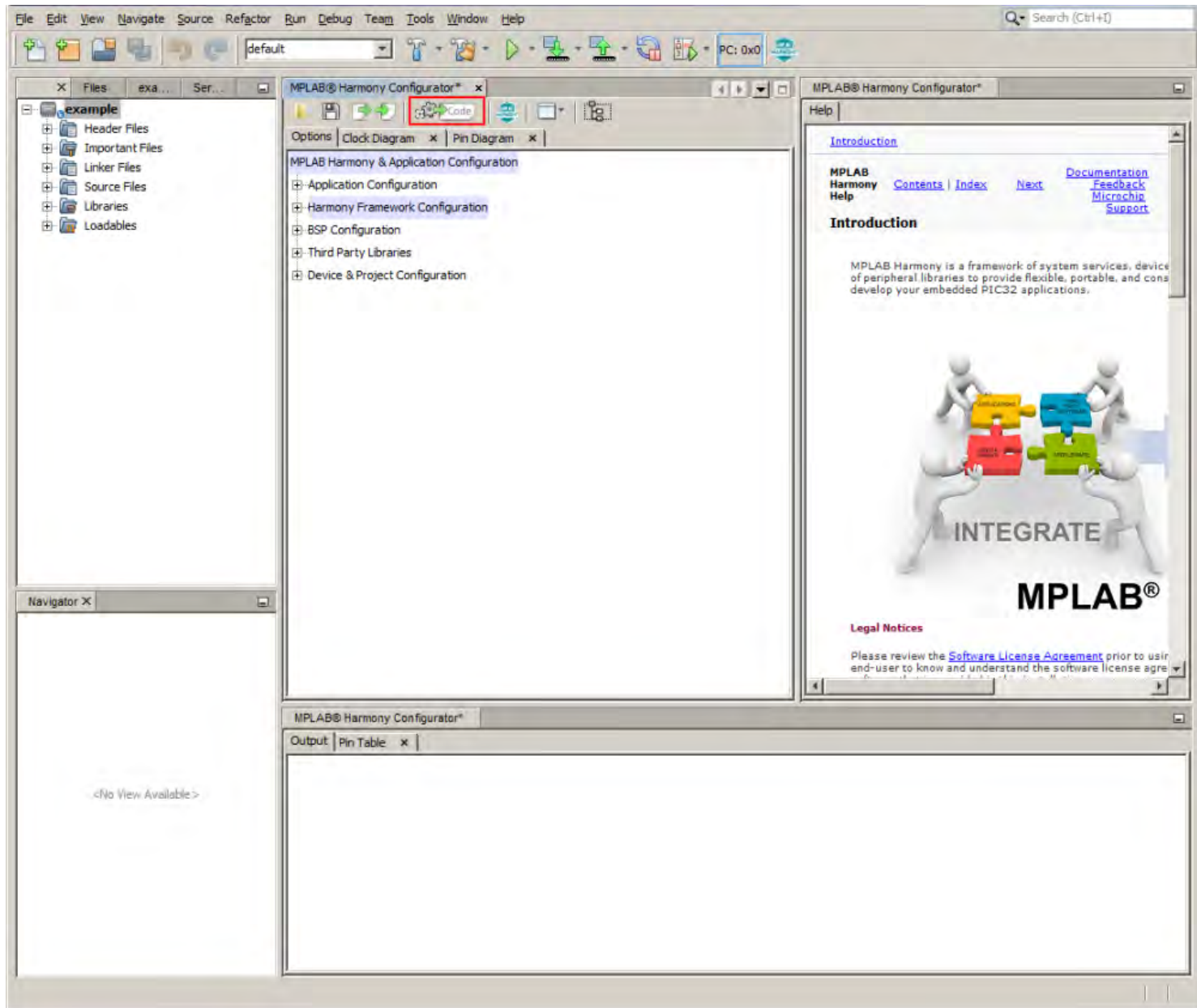done by clicking **Generate** in the main window. Various options for generation are displayed in the File Generation dialog.

- Overwrite local changes – Automatically overwrites any local changes made by the user. A merge window will be displayed for all locally changes files if this option is not selected.
- Enable recommended compiler optimizations (if not set) – A compiler optimization level of at least 'O1' is highly recommended for MPLAB Harmony projects. This option will set the compiler optimization level to 'O1' if no optimization level is currently set.
- The Generate button will cause all of the selected components and options to be processed and output as valid code files. These files will be automatically added to the project.

Using MHC to Create a New Application

Provides information on creating a new MHC project.

Introduction

This section provides an introduction to creating your own MPLAB Harmony applications using the MPLAB Harmony Configurator (MHC).

Description

MPLAB Harmony provides a MPLAB Harmony Configurator (MHC) MPLAB X IDE plug-in that can be installed in MPLAB X IDE to help you create your own MPLAB Harmony applications.

To create a new MPLAB Harmony application with MHC, follow these three steps:

- Step 1: Create the New Harmony Project
- Step 2: Add and Configure Required Libraries/Modules
- Step 3: MPLAB Harmony Application Structure and Developing the Application

Note: If you are a Microchip Libraries for Applications (MLA) user, and will be porting your application from the MLA TCP/IP, File System, USB Device, Graphics, or peripheral libraries to the MPLAB Harmony equivalents, refer to Porting to MPLAB Harmony for more information.

Prerequisites

This topic describes the prerequisites for creating your own MPLAB Harmony applications using MHC.

Description

This tutorial assumes that you have already completed these steps before you start:

1. Installed the MPLAB X IDE (http://www.microchip.com/mplabx).
2. Installed MPLAB Harmony (http://www.microchip.com/harmony).
3. Installed the MPLAB XC32 C/C++ Compiler (http://www.microchip.com/xc32).
4. Set up a working PIC32 development platform (http://www.microchip.com/32bit).

You can download the MPLAB X IDE, MPLAB Harmony and the MPLAB XC32 C/C++ Compiler from the links provided. If you do not already have a PIC32 development platform, you can learn more about the PIC32 family and determine which hardware platform best meets your development needs by visiting the 32-bit website listed previously.

This tutorial also assumes that you have some familiarity with the MPLAB X IDE, embedded C-language programming and PIC32 microcontrollers. If you are unsure how to complete some of the steps in this tutorial, please refer to the documentation for the item on which you have questions. You may also seek assistance from your peers on the Microchip discussion forums (http://www.microchip.com/forums) or from the Microchip support staff (www.microchip.com/support).

Once you have everything installed, connected, and up and running you are ready to begin creating your own MPLAB Harmony applications.

Step 1: Create the New Project

To create a new MPLAB Harmony project, you first need to create a new MPLAB X IDE project and the basic set of source code files and functions that are necessary for a properly formed MPLAB Harmony application.

Description

To create a new MHC project:

1. Select *File > New Project* or click the New Project icon in MPLAB X IDE.
2. In Categories, select **Microchip Embedded** and in Projects select **MPLAB Harmony Project** from the list of available project templates, and then click **Next** to launch the Microchip Harmony Configurator Project Wizard.
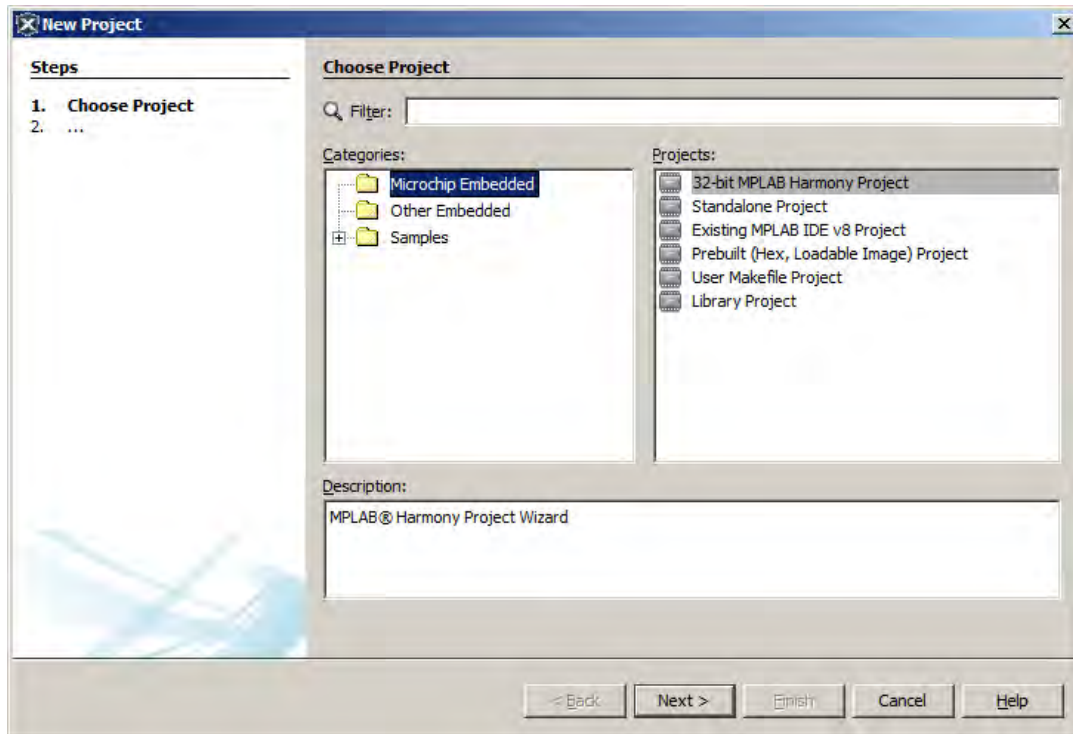
3. Specify the following in the New Project dialog:
   - Harmony Path (path to the folder containing Harmony framework: `<install-dir>`)
   - Project Location (the default project path is the `apps` folder within the selected MPLAB Harmony path)
   - Project Name
   - Configuration Name (optional)
   - Target Device (when a valid harmony path is selected, the device selection menu will be filled)
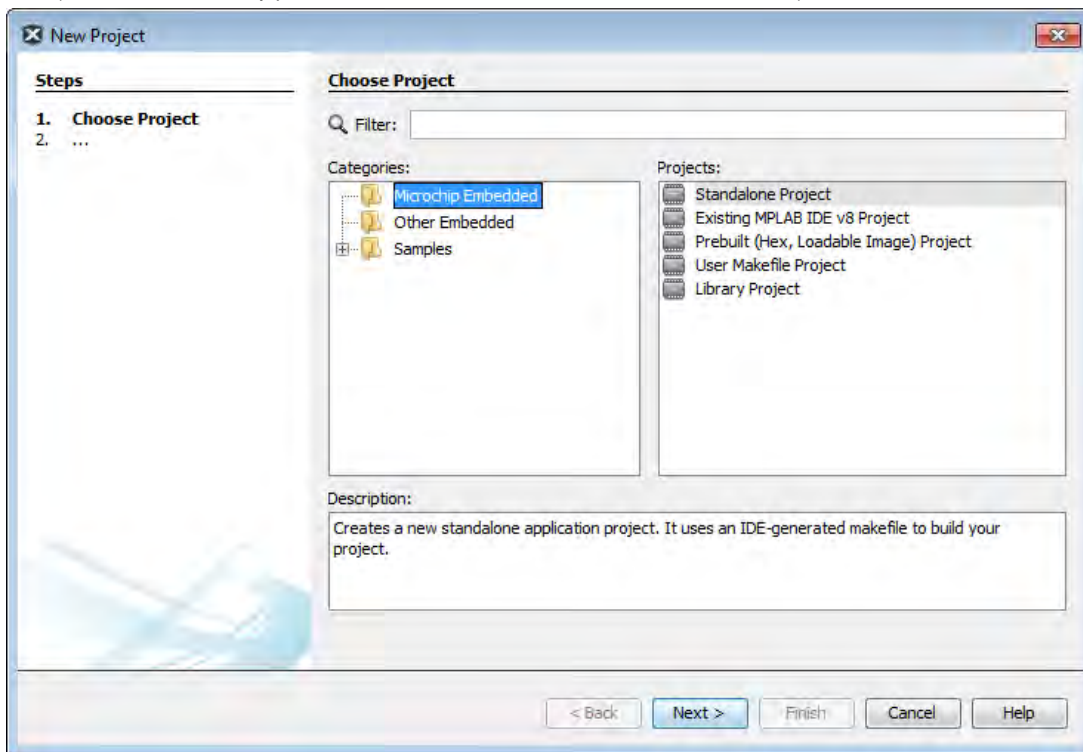
4. A MPLAB Harmony project will be created and the MPLAB Harmony Configurator will open. Refer to MPLAB Harmony Configurator for additional information.
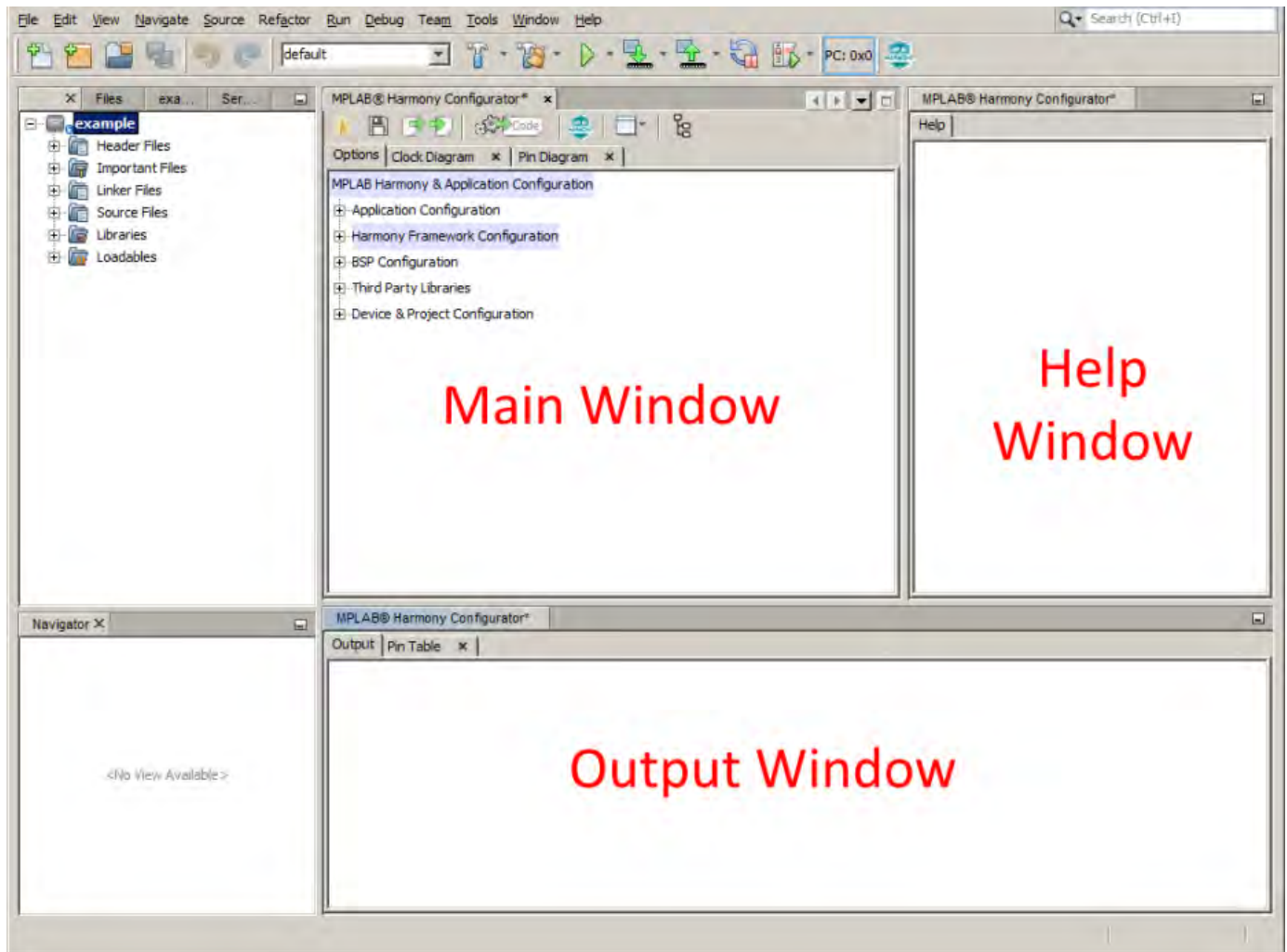
Step 2: Add and Configure the Required Libraries and Modules

This topic describes how to configure the MPLAB Harmony library modules.

Description

1. In the Main window, expand the Device Configuration tree and select the desired device configuration settings.
2. Expand the MPLAB Harmony Project Configuration tree and select and configure the desired libraries.
3. If use of a Board Support Package is desired, expand the BSP Configuration tree and select the desired BSP.
4. When complete, generate and save the configuration.
5. Develop your application logic using the selected libraries.

At this point, you should be able to build, debug, and step through the application. Effectively, you have a running MPLAB Harmony system; however, it is not yet ready to do anything. Next, you will develop your application state machine logic and make sure the system does what you want it to do.

Step 3: MPLAB Harmony Application Structure and Developing the Application

This topic describes the steps necessary to maintain the state machines.

Description

main.c

The main.c file contains calls to the SYS_Initialize function, which initializes MPLAB Harmony modules, as well as applications. It also contains the main task execution, which calls tasks for all selected MPLAB Harmony modules, as well as the application task function, APP_Tasks.

app.c

The `app.c` file contains the APP_Initialize function that is used to place an application into its initial state. It will be called from the SYS_Initialize function. The APP_Task function, which is also contained in the `app.c` file, implements the application state machine logic. Add application code

to this task as desired.

Refer to the example applications located in the `<install-dir>/apps/` folder within your MPLAB Harmony installation for example applications for various MPLAB Harmony modules. Related documentation is available in the *Applications Help > Examples* section.

Porting a Legacy PLIB to MPLAB Harmony

Provides an example on how to port a legacy (i.e., prior to MPLAB Harmony) USART Peripheral Library (PLIB) demonstration application to a MPLAB Harmony application using the MPLAB Harmony Configurator (MHC).

Description

A detailed procedure for porting the legacy UART PLIB Interrupt demonstration application (`<compiler-install-dir>/examples/plib_examples/uart/uart_interrupt`) to MPLAB Harmony is provided in the Framework Help > Peripheral Library Help > Peripheral Library Porting Example .

In this example, the following assumptions are made:

- The PIC32MX795F512L device will be used; however, the process described in this section is applicable for other PIC32 devices with appropriate changes
- The Explorer 16 Development Board is the hardware used in this example
- For the v1.33 MPLAB XC32 C/C++ Compiler, the `examples` folder is not present. To view the legacy USART PLIB example, refer to v1.31 or earlier of the MPLAB XC32 C/C++ compiler.

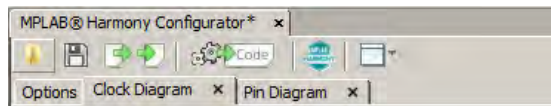## Configuring the Oscillator Module Using the MHC Clock Configurator

Provides information configuring the Oscillator module using the MHC Clock configurator

Description

The MHC Clock Configurator is a component of the MPLAB Harmony Configurator (MHC) MPLAB X IDE plug-in. Its function is to provide a graphical user interface to configure the Oscillator module.

While simulating the normal operation of the Oscillator module, the MHC Clock Configurator contains interactive controls, dynamic output, and visual warnings to help guide the user in establishing the desired system clock configuration.

The MHC Clock Configurator is launched automatically when the MHC is launched. It is in the form of a tab panel in MPLAB X IDE. Clicking the MPLAB Harmony Clock Configuration tab will open the MHC Clock Configurator.

The clock configurator screen can also be accessed using the main window toolbar application launch feature. Simply click the application launch icon and select **Clock Configuration**.
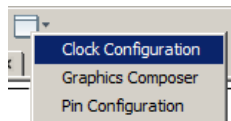
Another way to access the MHC Clock Configurator is via the Clock System Service section in MHC Harmony & Application Configuration tree view. Pressing the Execute button at the Launch Clock Configurator topic will either bring the tab panel into focus or launch the MHC Clock Configurator, if the tab panel was closed.
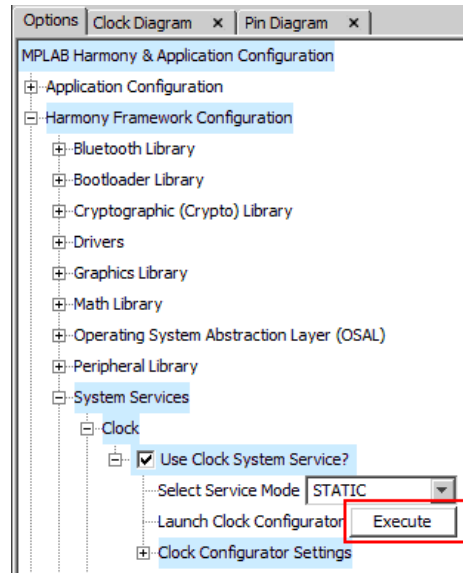
Note: The MHC Clock Configurator is one option to configure the Oscillator Module. Another option is to configure directly via the MPLAB Harmony & Application Configuration tree structure. The majority of the settings captured in the MHC Clock Configurator exist under the Clock Configurator Settings node in the Clock System Service, while the remainder are in the Device Configuration section.

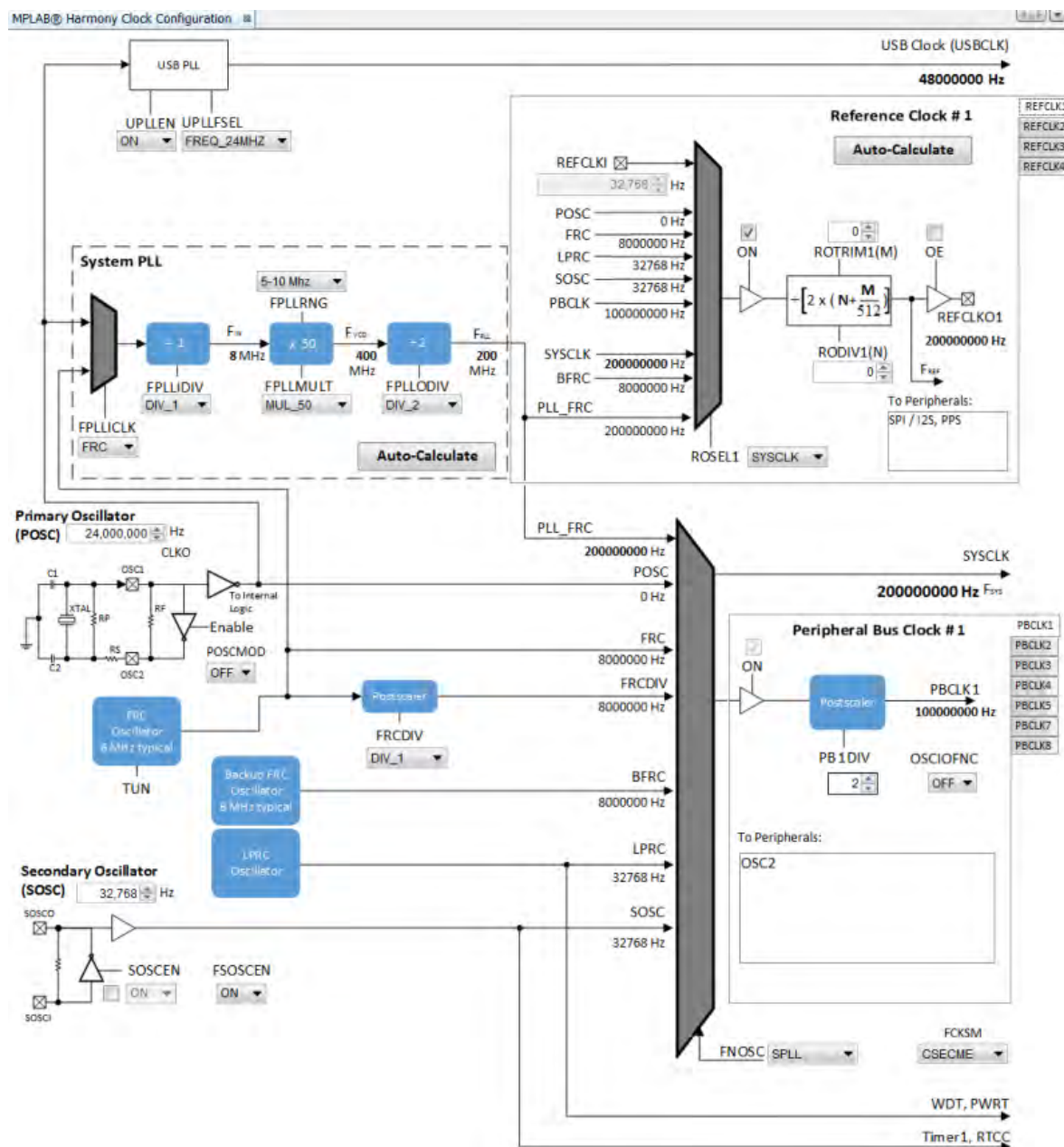## Clock Configuration for PIC32MZ Family Devices

Provides configuration information for PIC32MZ family devices.

Description

The MHC Clock Configurator's support of configuring the Oscillator Module of a PIC32MZ family device is divided into the following sub-sections:

- Configuring System Clock Frequency
- Configuring the Peripheral Bus Clocks
- Configuring the Reference Clocks
- Using the SPLL Divider Auto-Calculate Feature

For details regarding the operation of the Oscillator module, refer to the **"Oscillator"** chapter in the *"PIC32MZ Embedded Connectivity (EC) Family Data Sheet"* (DS60001191). This document is available for download from the Microchip website (www.microchip.com).

Configuring the System Clock Frequency

Provides information on configuring the system clock frequency for PIC32MZ family devices.

Description

There are a total of five external and internal oscillator options as clock source:

- Internal Fast RC (FRC) Oscillator divided by the FRCDIV bits in the OSCCON register
- Internal Low-Power RC (LPRC) Oscillator
- Secondary Oscillator (SOSC)
- Primary Oscillator (POSC) (POSCMOD: HS or EC)
- System PLL (SPLL)

The device configuration bit FNOSC is represented as a drop-down with the above selections in the MHC Clock Configuration. The current selection is represented in **bold**.
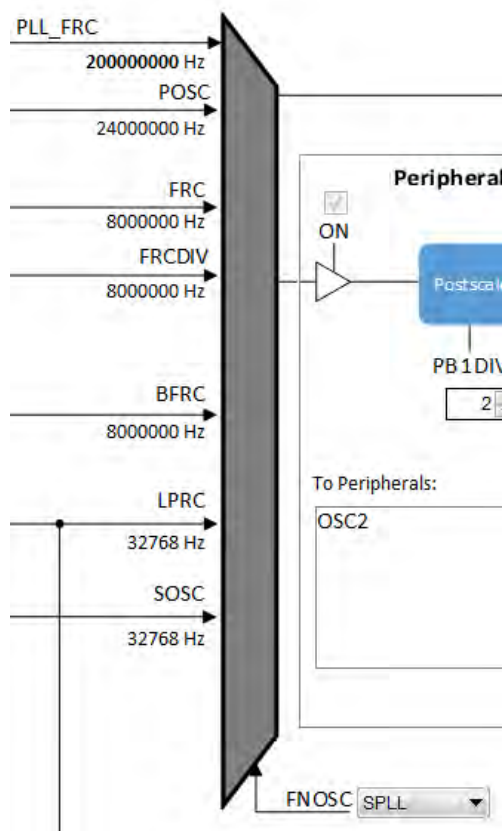
The Primary Oscillator (POSC) and Secondary Oscillator (SOSC) are customizable external clock sources. For the POSC, the device configuration bit, POSCMOD, needs to be set to EC or HS. If FNOSC is set to SOSC, the device configuration bit, FSOSCEN, should be set to ON. SOSCEN is set post-initialization. There is an option to override FSOSCEN with SOSCEN.
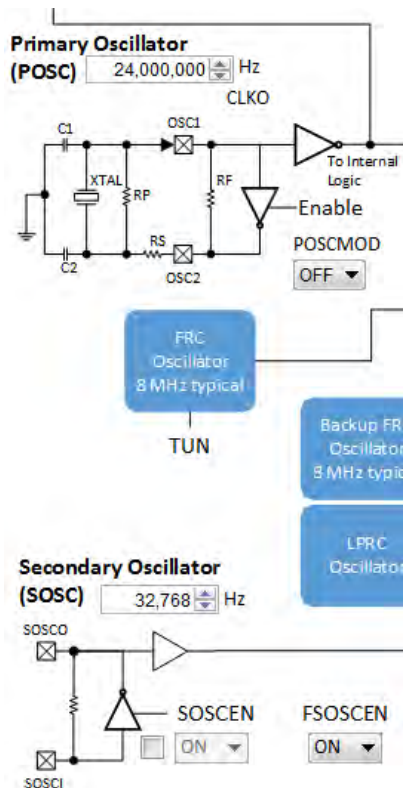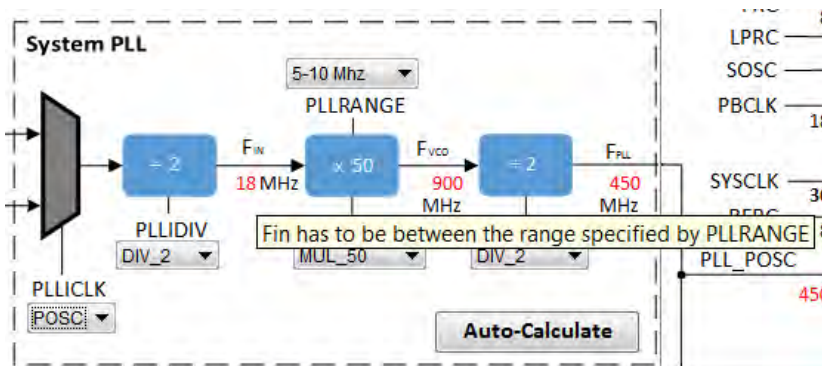

The output system clock frequency (SYSCLK) is displayed on the left side. This value (in Hz) corresponds to System Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.
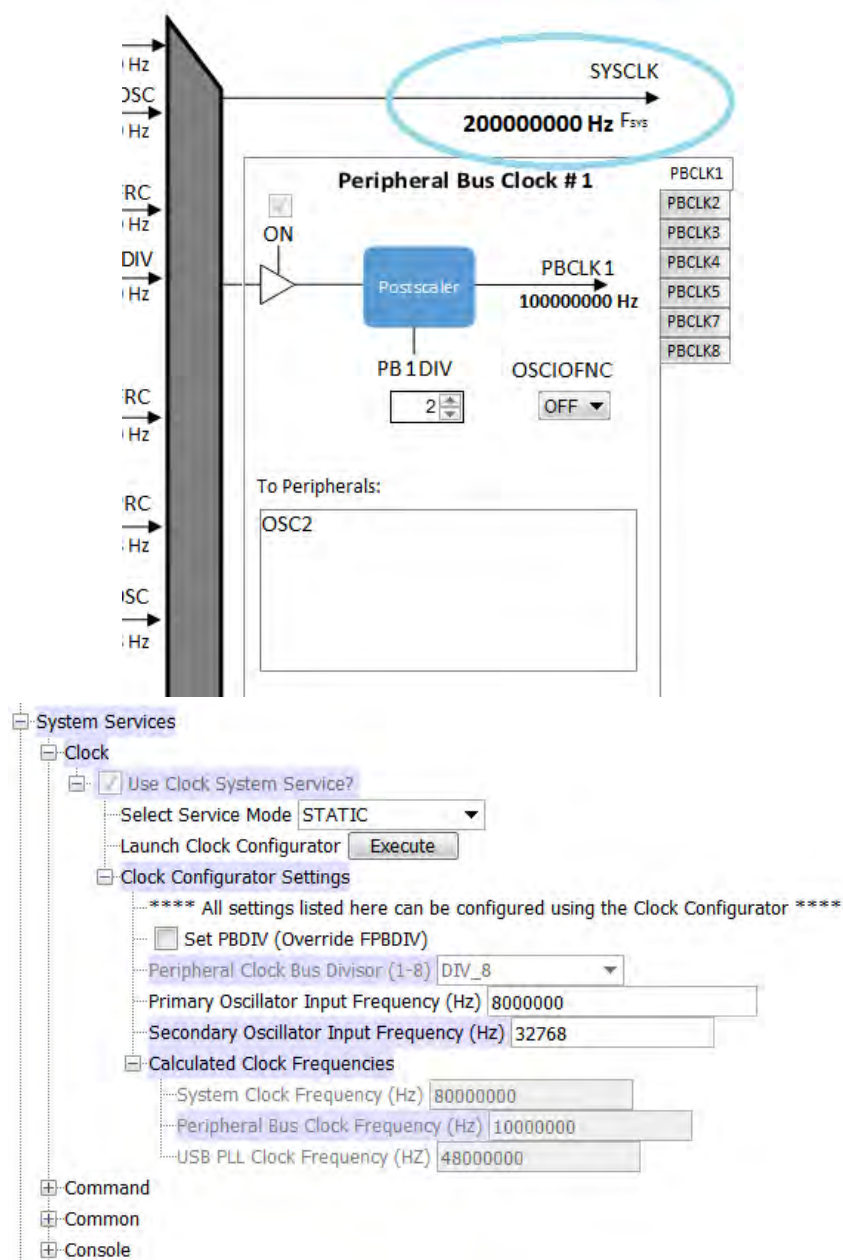
Certain frequency values may be displayed in red when the input value does not meet specification and may cripple performance of the device. An example is shown in the following figure, when the HS Oscillator Mode is selected for POSCMOD and the POSC input frequency set is outside of the 4 MHz - 32 MHz range. A dynamic help tip will also appear if the user hovers over the POSCMOD control or any of the red text.
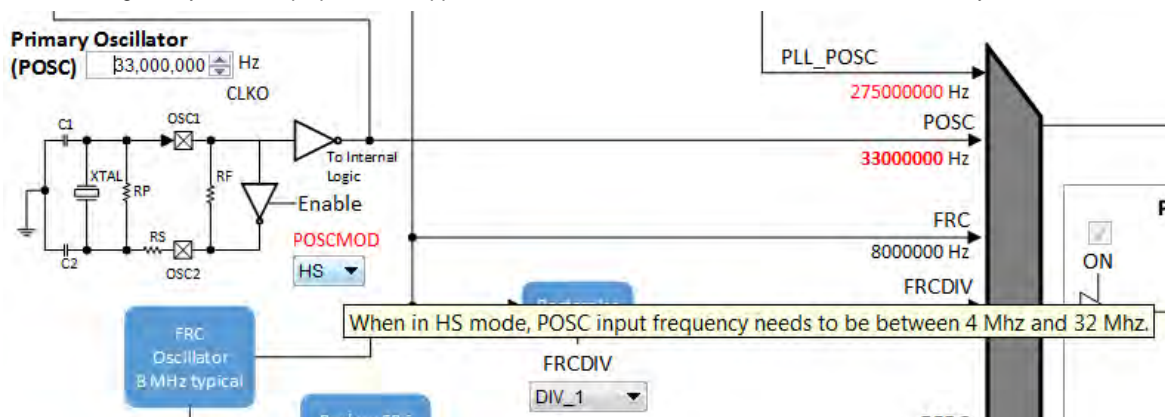
Another example is the SPLL, where FPLL (60 MHz – 120 MHz), FVCO (80 MHz – 240 MHz), and FIN (range specified by PLLRANGE) will appear as red text, including an explanation tool tip, if they fall outside of their respective required ranges.
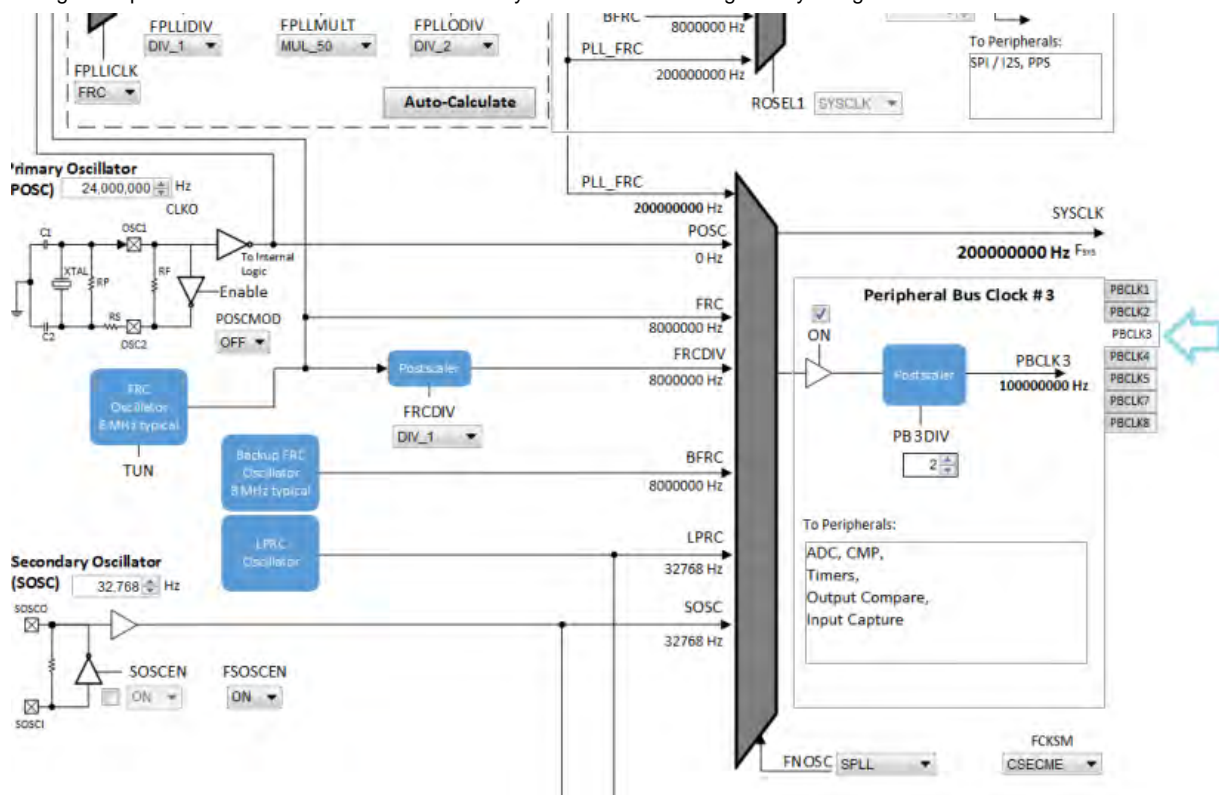
Configuring the Peripheral Bus Clocks

Provides information on configuring the peripheral bus clocks for PIC32MZ family devices.

Description

Each of the eight Peripheral Bus Clocks on the PIC32MZ family devices can be configured by using the tabs on the left.

The output frequency is in **bold**. The "To Peripherals" window provides a reminder of which peripherals each clock is driving.

This value (in Hz) corresponds to Peripheral Bus Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

> **Note:**   It is important to know the acceptable clock range for the peripherals. The Clock Configurator will NOT provide a warning if the output peripheral clock frequency falls outside of the specified range of the peripheral.

Configuring the Reference Clocks

Provides information on configuring the reference clocks for PIC32MZ family devices.

Description

Each of the four Reference Clocks on the MZ Family of device can be configured by using the tabs on the left.
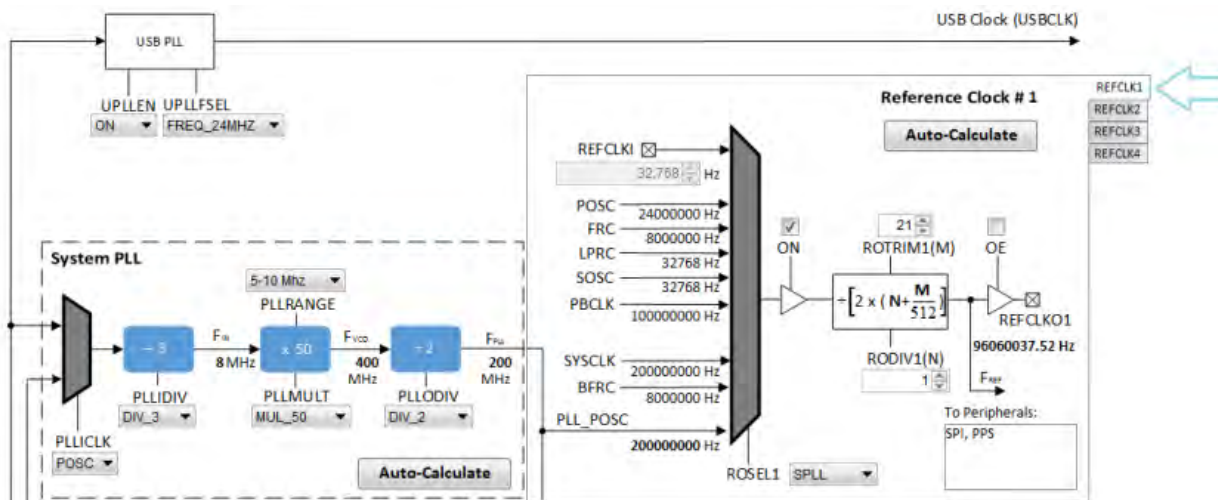
The clock input source (ROSELx), divider (RODIVx), trim value (ROTRIMx) are independently configurable. The output frequency (REFCLKOx) is in **bold**.

This value (in Hz) corresponds to Reference Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in the MHC Harmony & Application Configuration tree view.

Using the Reference Clock Auto-Calculate Feature

Provides information on the reference clock auto-calculate feature for PIC32MZ family devices.

Description

The MHC Clock Configurator is equipped with the ability to help the user establish the closest possible match to a user-desired target reference clock frequency. The Auto-Calculate feature is designed to determine the divider and trim values in the each of the four reference clocks based on a user requested clock output frequency.

The feature can be accessed via the Auto-Calculate button in the Reference Clock section of the Clock Configurator.
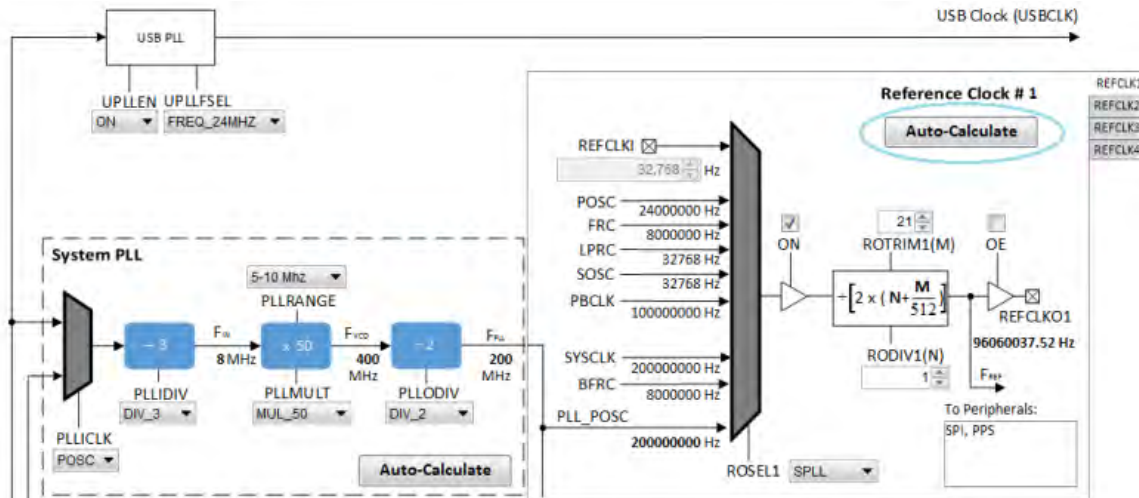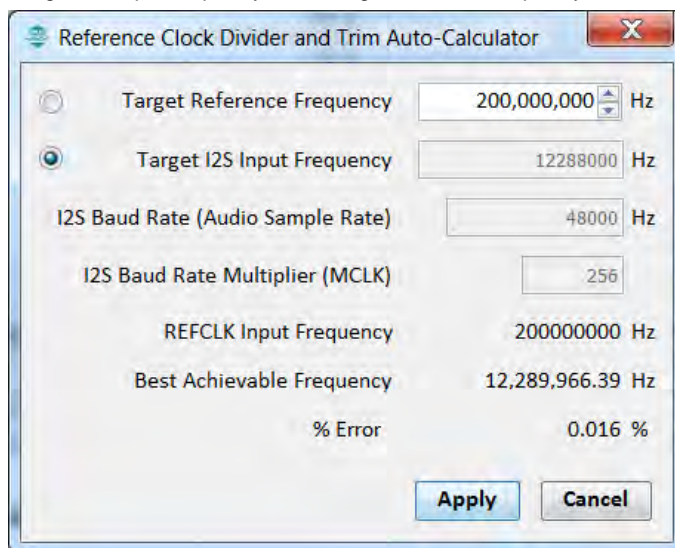

Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.
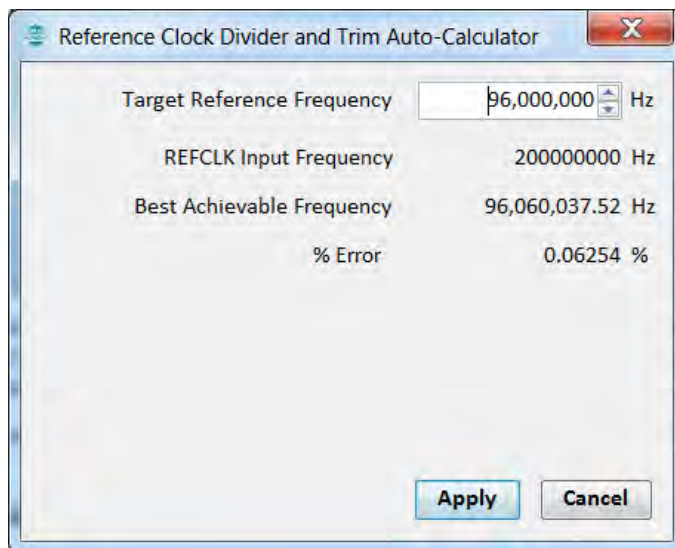
Enter the desired target reference frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the Reference Clock Divider (RODIVx) and Trim (ROTRIMx) combination, as well as the percentage discrepancy from the desired value, if any. The REFCLK Input Frequency is determined based on selection at ROSELx.

If the I2S driver is selected as part of the configuration, the Reference Clock Divider and Trim Auto-Calculator dialog opens automatically reconfigured with the option to use the target I2S input frequency as the target reference frequency.

Clicking the **Apply** button will cause the MHC Clock Configurator to update the Reference Clock divider and trim to establish the closest achievable frequency.

Using the SPLL Divider Auto-Calculate Feature

Provides information on the SPLL auto-calculate feature for PIC32MZ family devices.

Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target system clock frequency. The Auto-Calculate feature is designed to determine the divider and multiplier values in the SPLL-based on a user requested system clock frequency.

The feature can be accessed via the Auto-Calculate button in the SPLL section of the Clock Configurator.
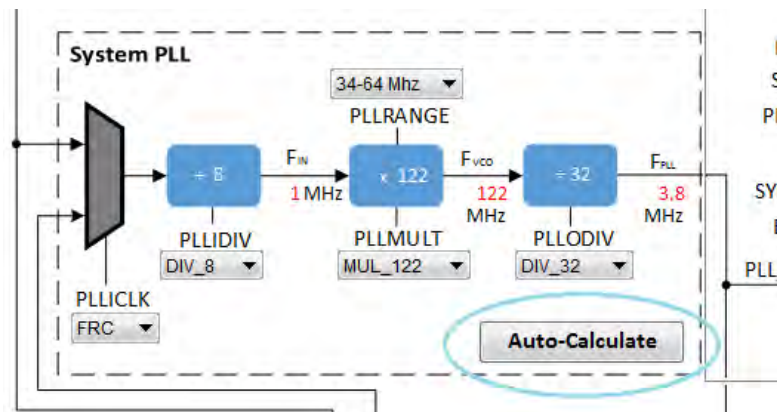
Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.
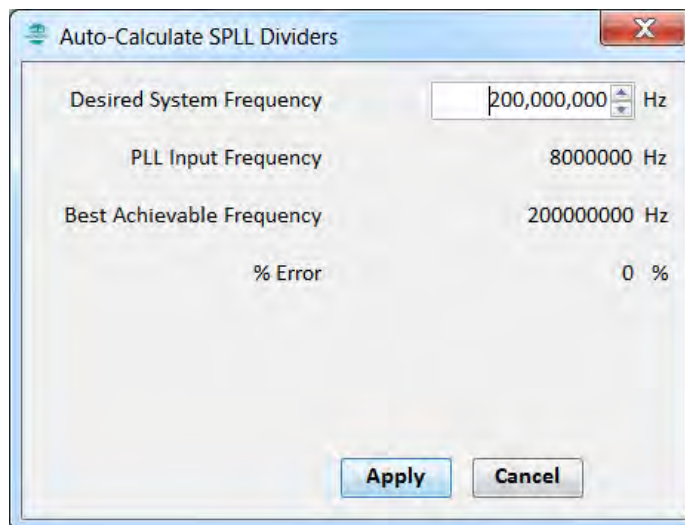
Enter the desired system clock frequency (remember to press the key ENTER), and the dialog window will display the best achievable frequency that can be provided by the SPLL divider/multiplier combination, as well as the percentage discrepancy from the desired value, if any. The PLL Input Frequency is determined based on selection at PLLICLK (FRC or POSC).

Clicking the **Apply** button will cause the MHC Clock Configurator to update the SPLL dividers and multiplier to establish the closest achievable frequency.

📝 **Note:**   The Auto-Calculate feature will also update the PLLRANGE setting to satisfy the necessary FIN frequency.

Clock Configuration for PIC32MX Family Devices

Provides configuration information for PIC32MX family devices.

Description

The MHC Clock Configurator's support of configuring the Oscillator Module of a MX Family Device is divided into the follow sub-sections:

- Configuring the System Clock Frequency
- Configuring the Peripheral Bus Clock
- Configuring the Reference Clock
- Configuring the USB PLL
- Using the SPLL Divider Auto-Calculate Feature

For details regarding the operation of the Oscillator module, refer to the **"Oscillator"** chapter in the specific PIC32MX device data sheet:

- PIC32MX1XX/2XX (DS60001168)
- PIC32MX1XX/2XX/5XX 64/100-pin Family (DS60001290)
- PIC32MX320/340/360/420/440/460 (DS60001143)
- PIC32MX330/350/370/430/450/470 (DS60001185)
- PIC32MX5XX/6XX/7XX (DS60001156)

Each of these documents are available for download from the Microchip website (www.microchip.com).

The following figure shows the configuration screen for PIC32MX1XX/2XX, PIC32MX 330/350/370/430/450/470, and PIC32MX1XX/2XX/5XX

64/100-pin Family devices.
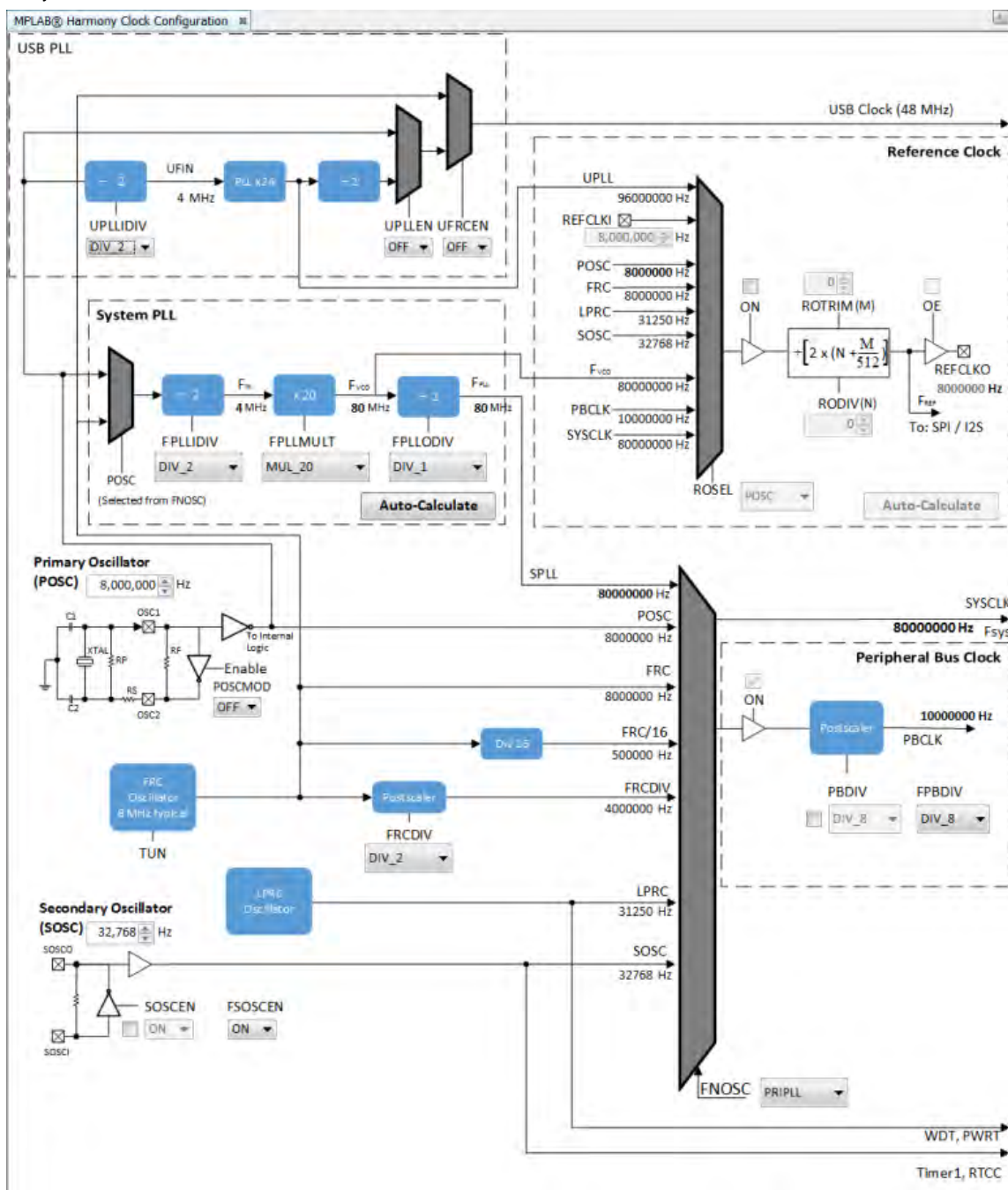
The next figure shows the configuration screen for PIC32MX320/340/360/420/440/460 and PIC32MX5XX/6XX/7XX devices.
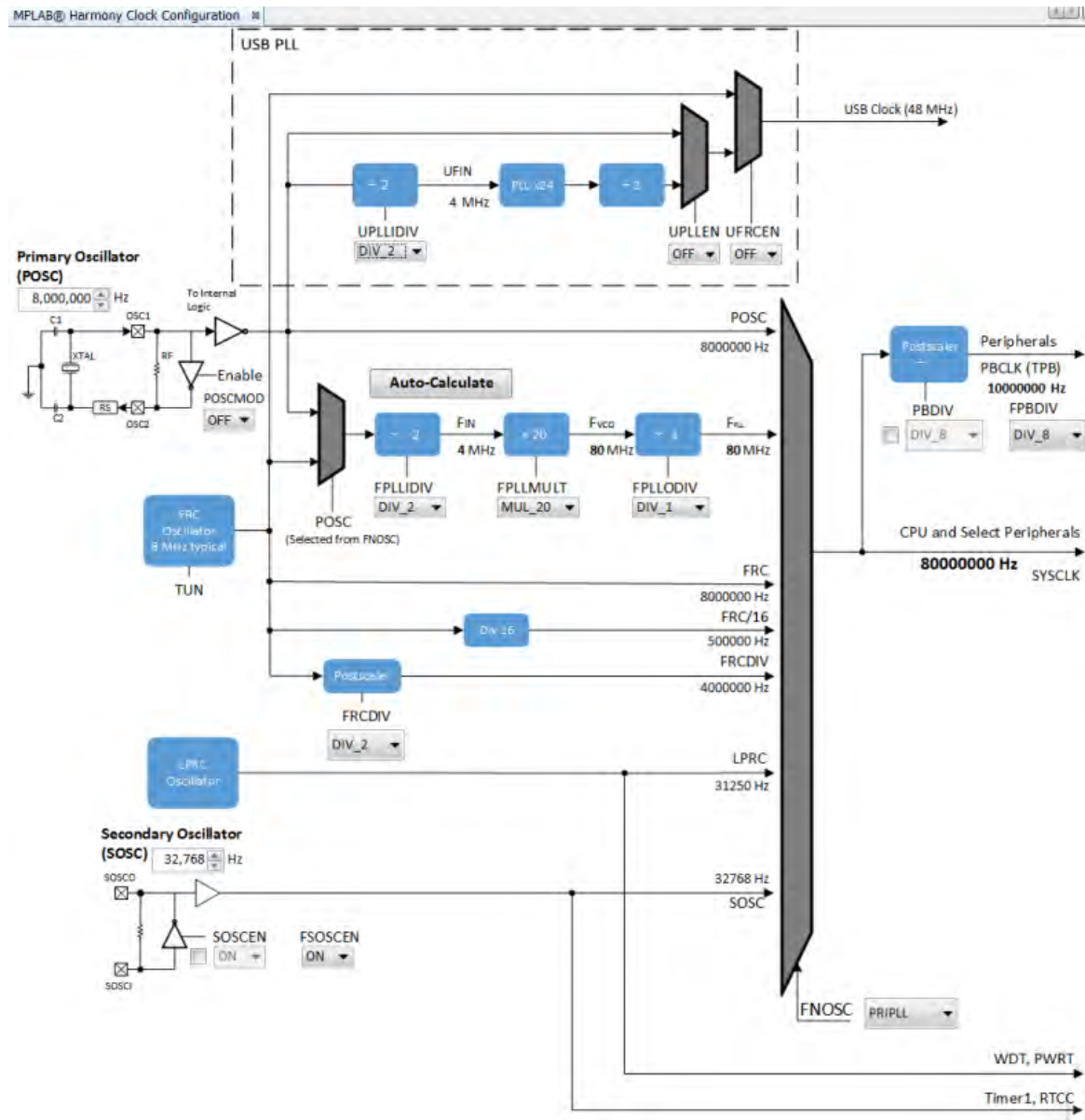
Configuring the System Clock Frequency

Provides information configuring the system clock frequency for PIC32MX family devices.

Description

There are a total of five external and internal oscillator options as clock source:

• Internal Fast RC Oscillator (FRC) divided by the FRCDIV bits in the OSCCON register
• Internal Fast RC Oscillator (FRC) divided by 16
• Internal Low-Power RC (LPRC) Oscillator
• Secondary Oscillator (SOSC)
• Primary Oscillator with PLL module (PRIPLL)
• Primary Oscillator (POSCMOD: XT, HS, or EC)
• Internal Fast Internal RC Oscillator with PLL module via Postscaler (FRCPLL)
• Internal Fast Internal RC Oscillator (FRC)

The device configuration bit FNOSC is represented as a drop-down with the above selections in the MHC Clock Configuration. The current selection is represented in **bold**.
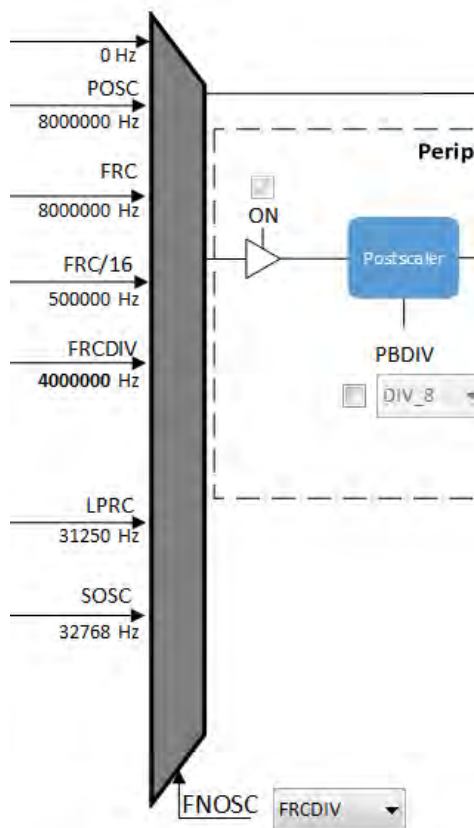
Primary Oscillator (POSC) and Secondary Oscillator (SOSC) are customizable external clock source. For POSC, the device configuration bit POSCMOD needs to be set to EC, XT, or HS. If FNOSC is set to SOSC, the device configuration bit FSOSCEN needs to be set to ON.
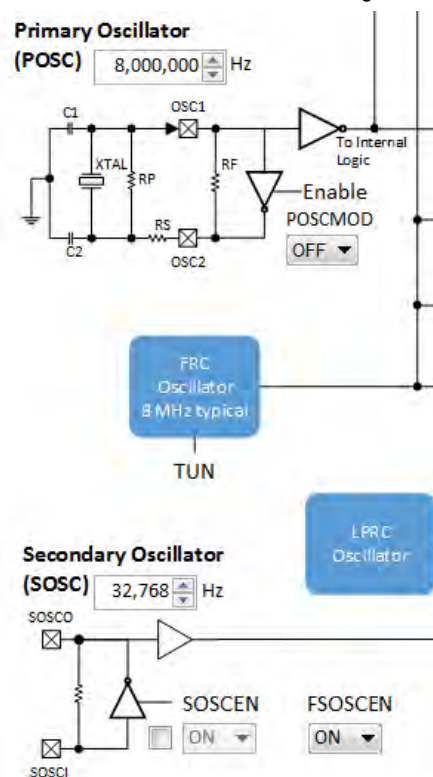
The output system clock frequency (SYSCLK) is displayed on the left side. This value (in Hz) corresponds to System Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.
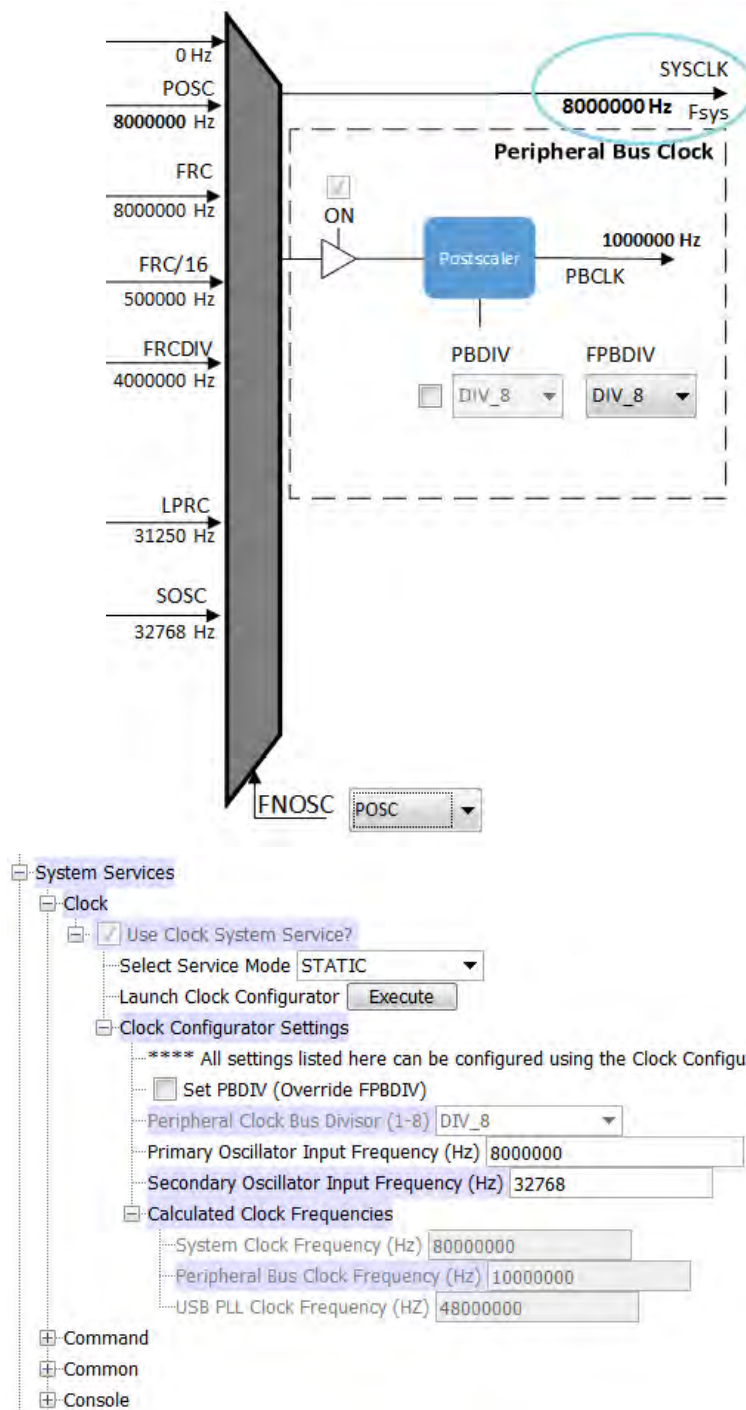
Certain frequency values may be displayed in red when the input value does not meet specification and may cripple performance of the device. An example is shown in the following figure, when the XT Oscillator Mode is selected for POSCMOD and the POSC input frequency set is outside of the 3 MHz - 10 MHz range. A dynamic help tip will also appear if the user hovers over the POSCMOD control or any of the red text.
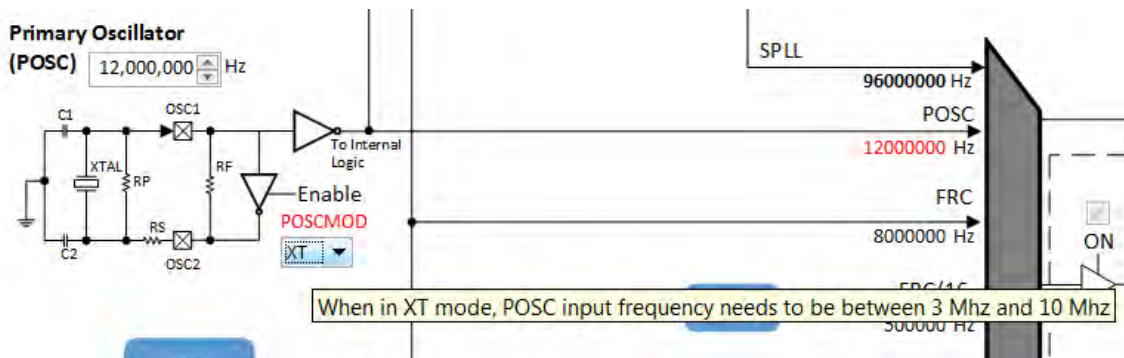
Another example is the SPLL, where FPLL (40 MHz – 120 MHz), FVCO (60 MHz – 120 MHz), and FIN (3.92 MHz – 5 MHz) will appear in red text, including an explanation tool tip, if they fall outside of their respective required ranges.
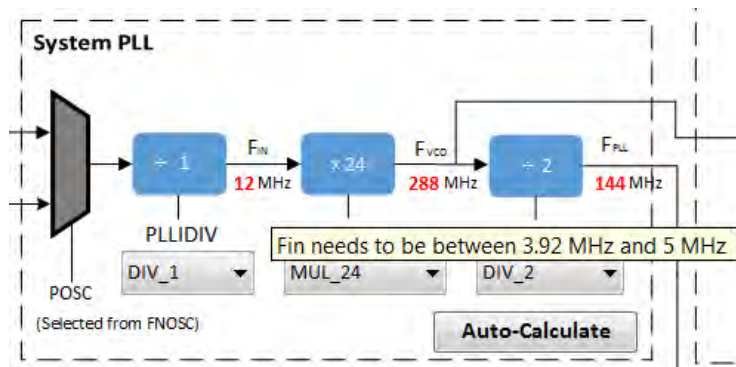
Configuring the Peripheral Bus Clock

Provides information on configuring the peripheral bus clock for PIC32MX family devices.

Description
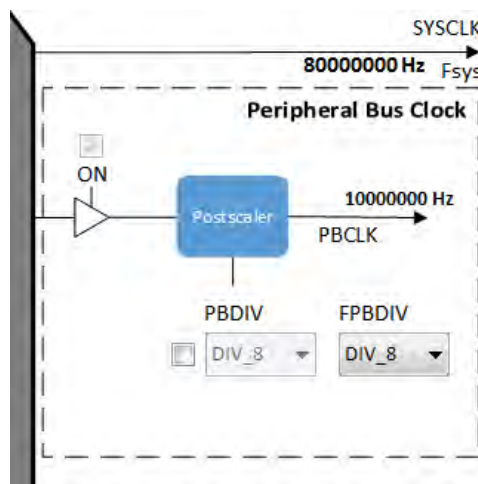
The Peripheral Bus Clock on the MX Family of device can be configured on the left.

The output frequency is in **bold**. This value (in Hz) corresponds to Peripheral Bus Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.

> **Note:** It is important to know the acceptable clock range for the peripherals. The Clock Configurator will NOT provide a warning if the output peripheral clock frequency falls outside of specified range of the peripheral.

Configuring the Reference Clock

Provides information on configuring the reference clock for PIC32MX family devices.

Description

The Reference Clock on the PIC32MX1XX/2XX, PIC32MX 330/350/370/430/450/470, and PIC32MX1XX/2XX/5XX 64/100-pin Family devices can be configured in the section labeled Reference Clock on the upper right area of the screen.

The clock input source (ROSEL), divider (RODIV), trim value (ROTRIM) are independently configurable. The output frequency (REFCLKO) is in bold.

This value (in Hz) corresponds to Reference Clock Frequency under Calculated Clock Frequencies in the Clock System Service section in MHC Harmony & Application Configuration tree view.
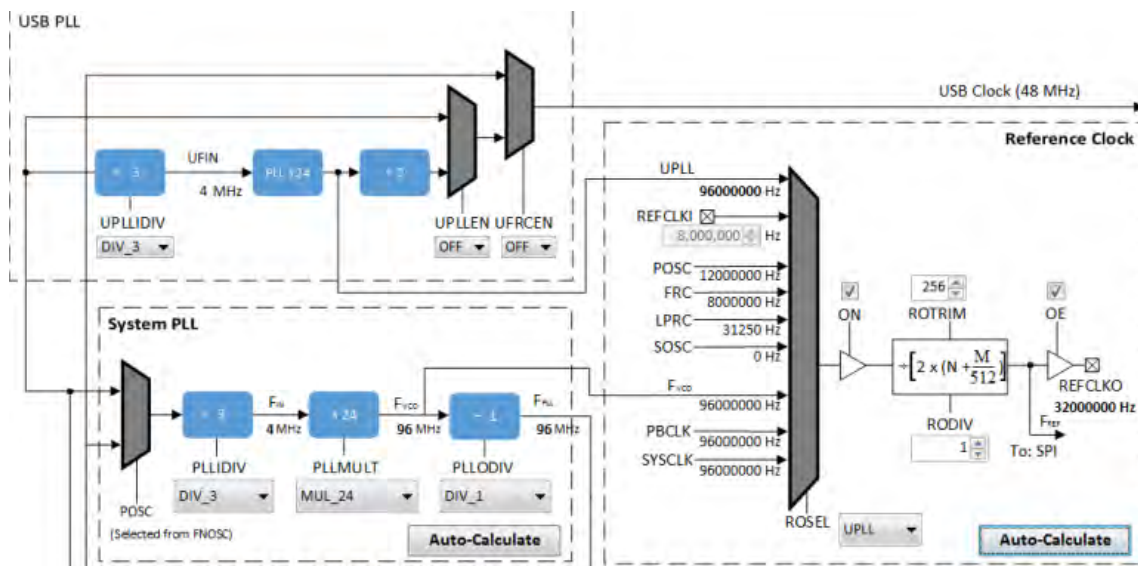
Using the Reference Clock Auto-Calculate Feature

Provides information on the reference clock auto-calculate feature for PIC32MX family devices.

Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target reference clock frequency. The Auto-Calculation feature is designed to determine the divider and trim values for the reference clock based on a user requested clock output frequency.

The feature can be accessed via the Auto-Calculate button in the Reference Clock section of the Clock Configurator.
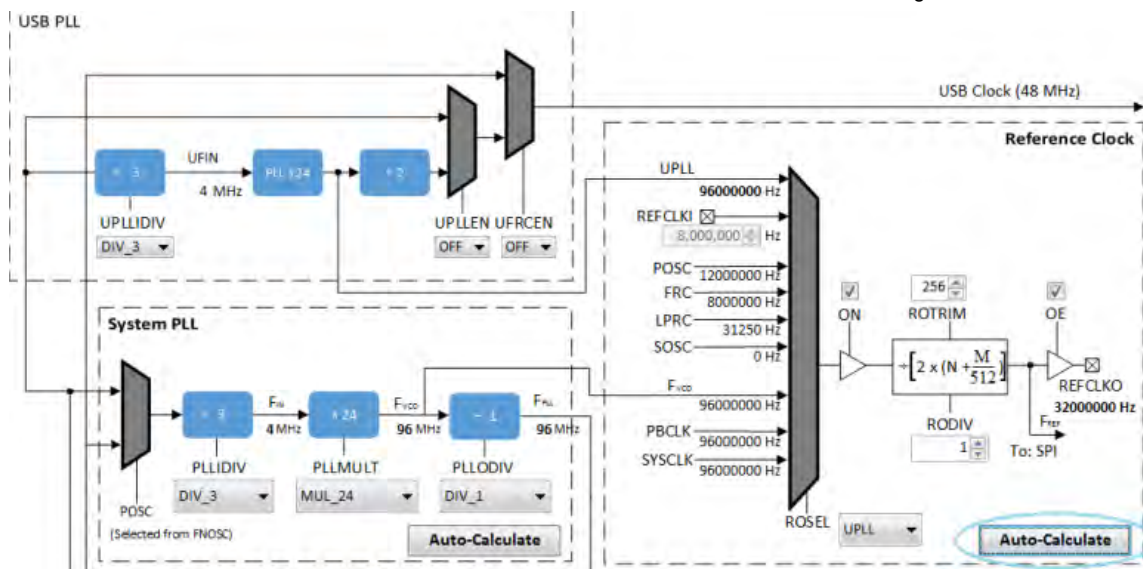
Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.
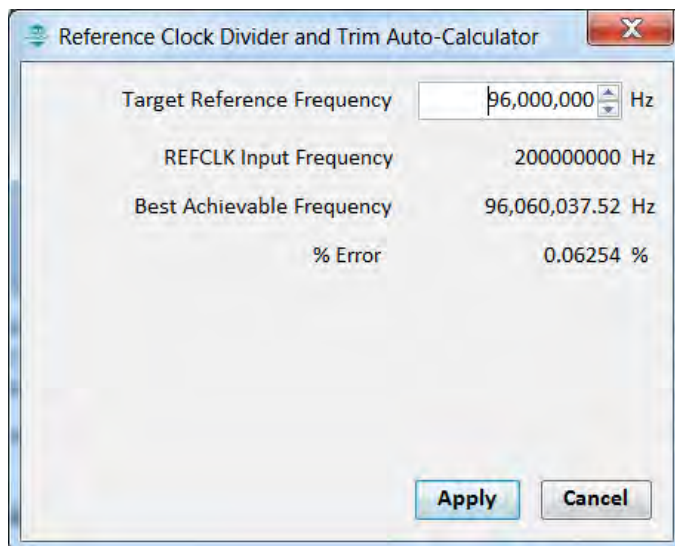
Enter the desired system clock frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the Reference Clock Divider (RODIV) and Trim (ROTRIM) combination, as well as the percentage discrepancy from the desired value, if any. The REFCLK Input Frequency is determined based on selection at ROSEL.

If the I2S driver is selected as part of the configuration, the Reference Clock Divider and Trim Auto-Calculator dialog opens automatically reconfigured with the option to use the target I2S input frequency as the target reference frequency.

Clicking the **Apply** button will cause the MHC Clock Configurator to update the Reference Clock divider and trim to establish the closest achievable frequency.

Configuring the USB PLL

Provides information on configuring the USB PLL for PIC32MX family devices.

Description

Part of enabling the USB peripheral is to enable the USB PLL. The USB PLL requires 4 MHz input clock frequency for accurate operation. With POSC being a variable value, it is important to configure the correct USB PLL Input Divider (UPLLIDIV) value. The MHC Clock Configurator will provide visual warning if the value can lead to inaccuracy in USB operation.
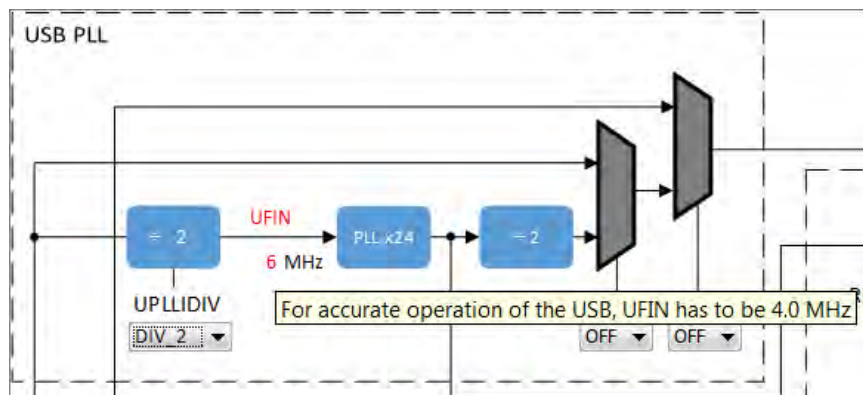
Using the SPLL Divider Auto-Calculate Feature

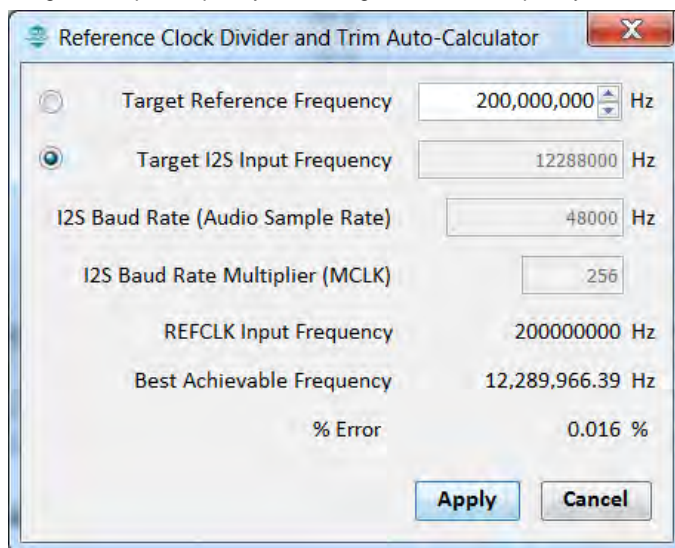Provides information on using the SPLL Divider Auto-Calculate feature for PIC32MX family devices.

Description

The MHC Clock Configurator is equipped with the ability to help the user establish closest possible match to a user-desired target system clock frequency. The Auto-Calculation feature is designed to determine the divider and multiplier values in the SPLL-based on a user requested system clock frequency.

The feature can be accessed via the Auto-Calculate button in the System PLL section of the Clock Configurator.
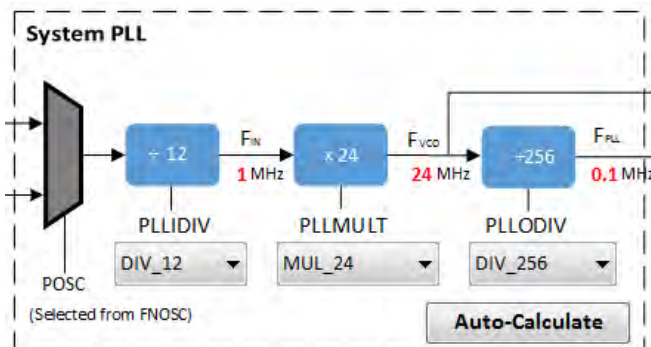
Clicking the **Auto-Calculate** button opens the Auto-Calculate dialog.
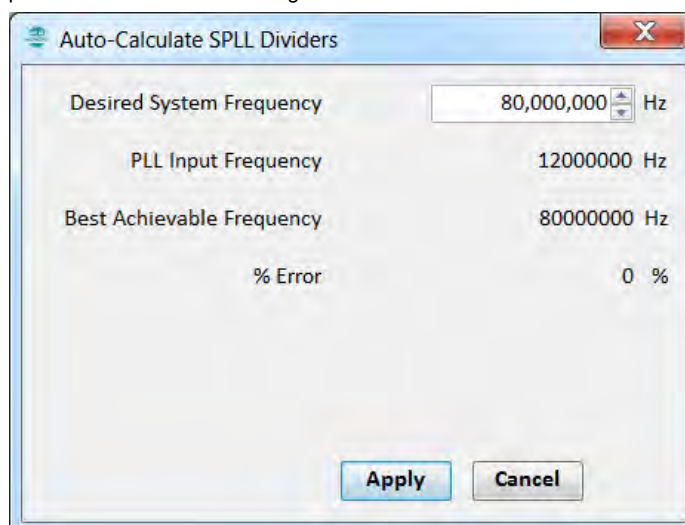
Enter the desired system clock frequency (remember to press the <Enter> key), and the dialog window will display the best achievable frequency that can be provided by the SPLL divider/multiplier combination, as well as the percentage discrepancy from the desired value, if any. The PLL Input Frequency is determined based on selection at FNOSC (FRCPLL or PRIPLL).

Clicking the **Apply** button will cause the MHC Clock Configurator to update the SPLL dividers and multiplier to establish the closest achievable frequency.

MPLAB Harmony Graphical Pin Manager

Provides information on the MPLAB Harmony Graphical Pin Manager tool that resides within MHC.

Description

This graphical management tool exists for the purpose of enabling users to configure the pins of Microchip devices in a fast and intelligent manner. The tool consists of a graphical representation of the state of the component and table that provides the means to configure the pins of the device. Users intending to use this tool should be familiar with the MPLAB Harmony configuration tree.

The user configures a device using the following process:

- Launch the tool (if not already running)
- Add modules by enabling desired functionality in the configuration tree (e.g., USART or SPI)
- Using the pin table to "Lock" cells representing function and pin pairings
- Using the pin flag management dialog to change pin register values
- Generating resultant code through the **Generate** button

Once generation is complete, the resultant code for configuring the device pins will be automatically added to the user's project.

Launching the Tool

Describes how to launch the pin manager tool.

Description

The pin manager tool automatically launches when MHC starts.
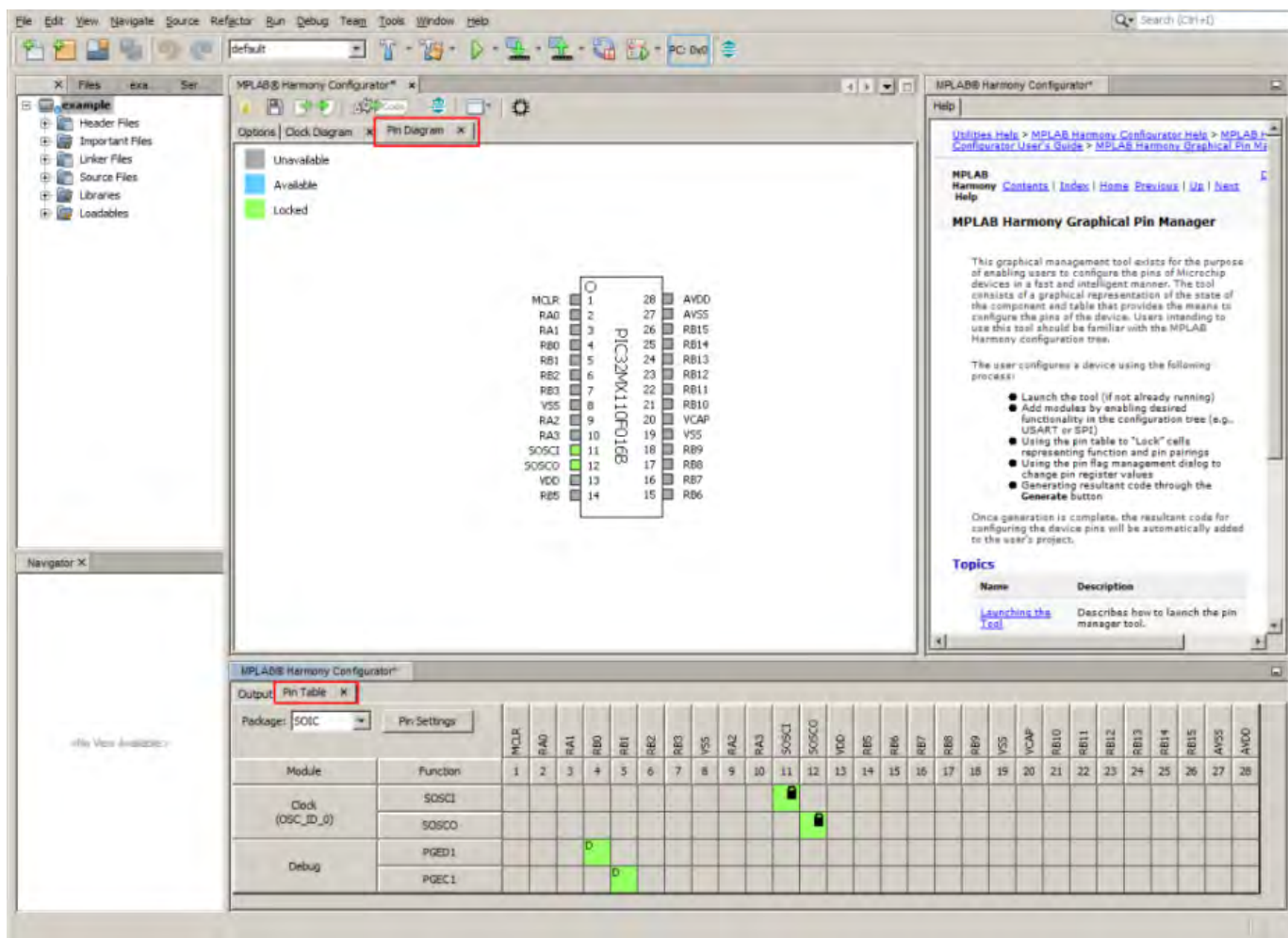
The pin manager tool can be launched from the main window toolbar application launcher or from the option tree.

Clock Configuration

Graphics Composer
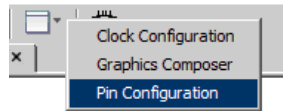
Pin Configuration

The pin manager tool can also be launched from the configuration tree.

System Services
  Clock
  Command
  Common
  Console
  Debug
  Device Control
  DMA
  File System
  Interrupts
  Message
  Ports
      ☑ Use Ports System Service?
      Launch Pin Manager   Execute
      **** Use Pin Manager to configure PPS, CN, Pull-Up, Pull-Down, Open Drain, Digital & Analog functionality ****

Tool Tabs

The pin manager tool has two tabs:

- Pin Diagram (see the red section in the following figure)
- Pin Table (see the blue section in the following figure)
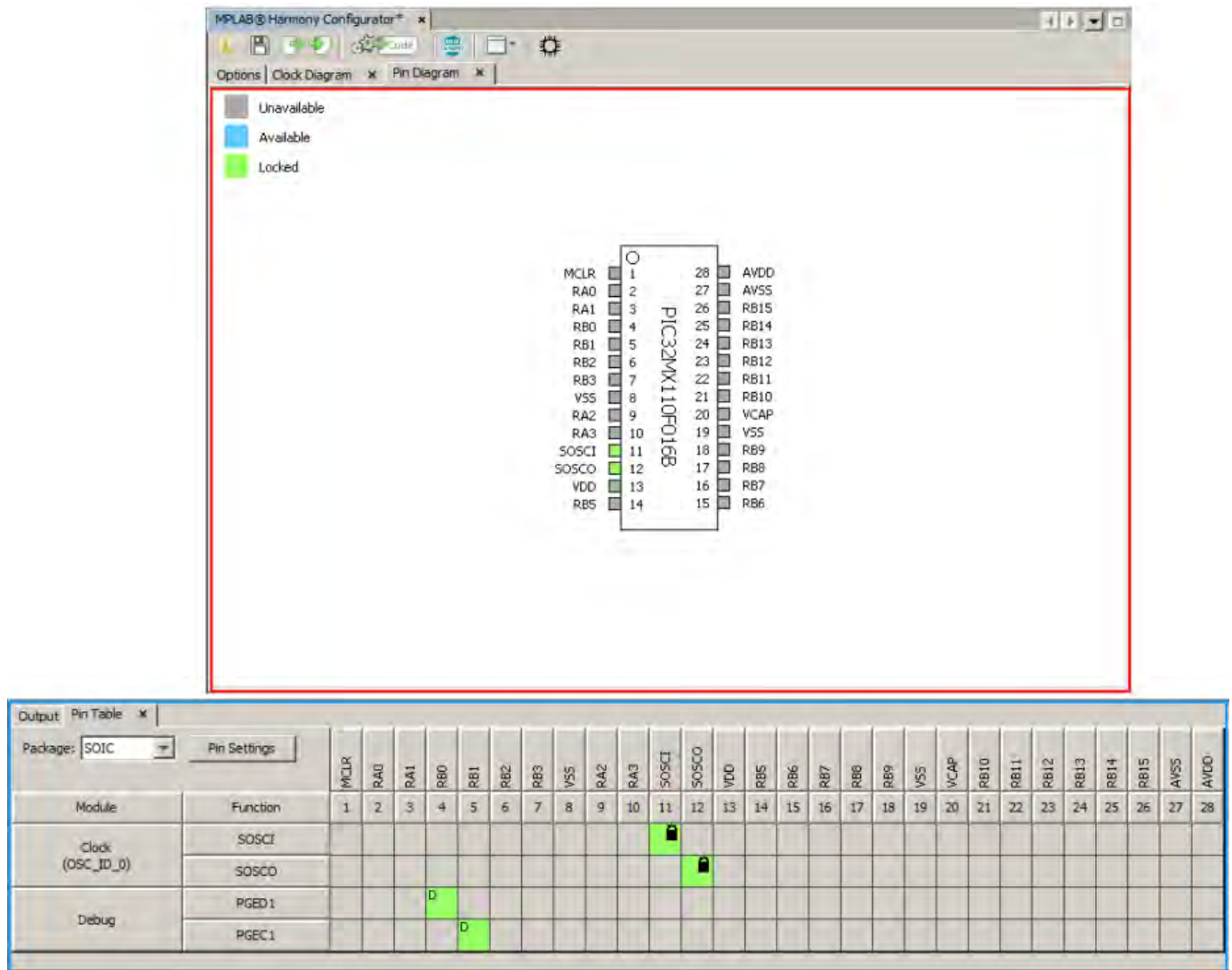
Pin Diagram Tab

Describes the pin diagram features.

Description

This diagram is a graphical representation of the selected component to be configured. The diagram contains the following:

Pin Names

These are the base names of each pin. These names will change based on the selected function for this pin.
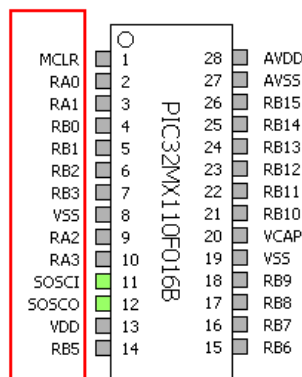
Pin States

This is a graphical indication of the state of the pin.

Pin States Legend:

Color	Icon	Description
Blue	☐	This pin can be locked to an available function in the table.
Gray	☐	This pin is currently unavailable based on the state of the pin table.
Green	☐	This pin has been locked to a function.
Red	☐	This pin has been automatically locked to a pin based on function priority.
Yellow	☐	This pin is currently highlighted by the cursor.
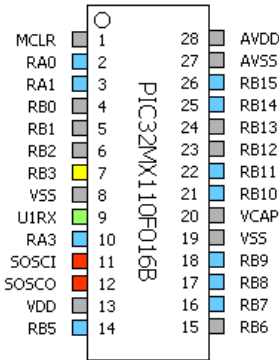
Pin Numbers

The number for each pin.
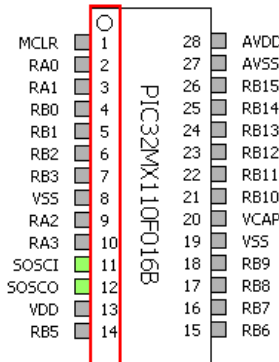
Component Name

The name of this component.
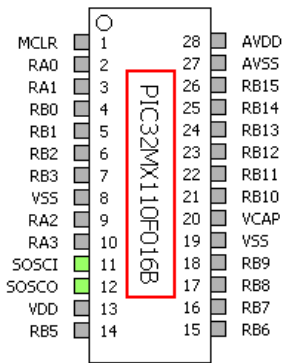
Pin Table Tab

Describes the pin table features.

Description

The pin table allows the user to graphically configure the pins for the given component. The table contains the following areas of interest:

Package Selector

This menu contains the available packages for the selected component.

📝 **Note:**   Changing this value will reset the state of the pins to default.

Observe the changes in the diagram and table when the QFN package is selected for this device.
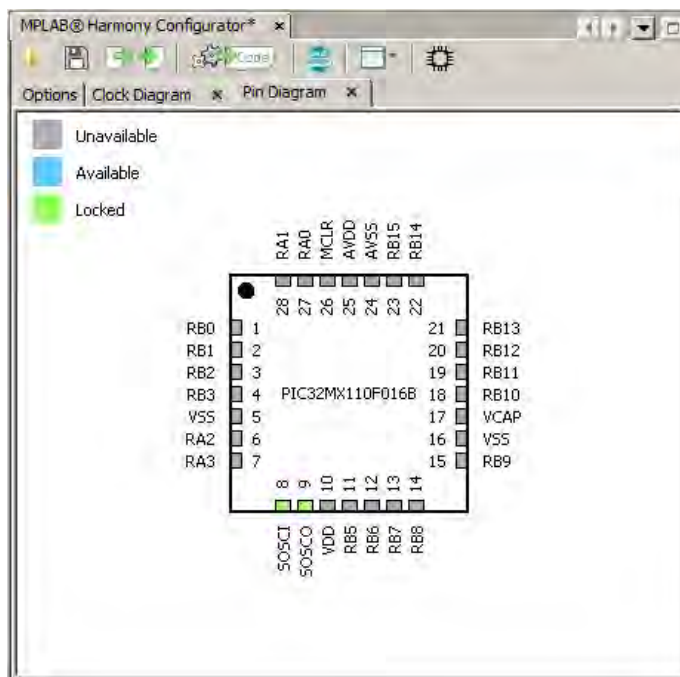
Pin Settings Button

This button shows the pin settings configuration menu. This dialog allows for the configuration of pin direction, drain, mode, latch, change notification, and pull-up and pull-down options.

📝 **Note:**   The direction and mode options are dependent on the function that is assigned to the pin. Board Support Package functions may lock other options as well.

The pin settings dialog can also be launched from the main toolbar when the pin diagram is visible.

MPLAB® Harmony Configurator* ×

Options | Clock Diagram × | Pin Diagram ×

Pin Setting Configuration
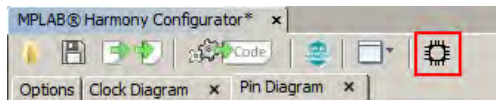
Pin	Name	Voltage Tolerance	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)	Change Notification (CNEN)	Pull Up (CNPU)	Pull Down (CNPD)
1	MCLR	5V		In	n/a	☐	Digital	☐	☐	☐
2	RA0			In	n/a	☐	Analog	☐	☐	☐
3	RA1			In	n/a	☐	Analog	☐	☐	☐
4	RB0			In	n/a	☐	Analog	☐	☐	☐
5	RB1			In	n/a	☐	Analog	☐	☐	☐
6	RB2			In	n/a	☐	Analog	☐	☐	☐
7	RB3			In	n/a	☐	Analog	☐	☐	☐
8	VSS	5V		In	n/a	☐	Digital	☐	☐	☐
9	RA2			In	n/a	☐	Analog	☐	☐	☐
10	RA3			In	n/a	☐	Analog	☐	☐	☐
11	RB4		SOSCI	n/a	n/a	☐	Analog	☐	☐	☐
12	RA4		SOSCO	n/a	n/a	☐	Analog	☐	☐	☐
13	VDD	5V		In	n/a	☐	Digital	☐	☐	☐
14	RB5	5V		In	n/a	☐	Digital	☐	☐	☐
15	RB6	5V		In	n/a	☐	Digital	☐	☐	☐
16	RB7	5V		In	n/a	☐	Digital	☐	☐	☐
17	RB8	5V		In	n/a	☐	Digital	☐	☐	☐
18	RB9	5V		In	n/a	☐	Digital	☐	☐	☐
19	VSS	5V		In	n/a	☐	Digital	☐	☐	☐
20	VCAP	5V		In	n/a	☐	Digital	☐	☐	☐
21	RB10	5V		In	n/a	☐	Digital	☐	☐	☐
22	RB11	5V		In	n/a	☐	Digital	☐	☐	☐

Pin Names

This row indicates the currently selected function for each pin. If no function is selected, the default pin name is shown instead.

Output | Pin Table ×

Package: SOIC   Pin Settings

Module	Function	MCLR 1	RA0 2	RA1 3	RB0 4	RB1 5	RB2 6	RB3 7	VSS 8	RA2 9	RA3 10	SOSCI 11	SOSCO 12	VDD 13	RB5 14	RB6 15	RB7 16	RB8 17	RB9 18	VSS 19	VCAP 20	RB10 21	RB11 22	RB12 23	RB13 24	RB14 25	RB15 26	AVSS 27	AVDD 28
Clock (OSC_ID_0)	SOSCI											🔒																	
	SOSCO												🔒																
Debug	PGED1				D																								
	PGEC1					D																							

Pin Numbers

This row indicates the number of each pin in the table.

Output | Pin Table ×

Package: SOIC   Pin Settings

Module	Function	MCLR 1	RA0 2	RA1 3	RB0 4	RB1 5	RB2 6	RB3 7	VSS 8	RA2 9	RA3 10	SOSCI 11	SOSCO 12	VDD 13	RB5 14	RB6 15	RB7 16	RB8 17	RB9 18	VSS 19	VCAP 20	RB10 21	RB11 22	RB12 23	RB13 24	RB14 25	RB15 26	AVSS 27	AVDD 28
Clock (OSC_ID_0)	SOSCI											🔒																	
	SOSCO												🔒																
Debug	PGED1				D																								
	PGEC1					D																							

Table Modules

This column contains the modules, or groups of functions, for the current configuration. These modules are controlled by the MHC configuration tree.

Table Functions

This column displays the functions that belong to each module.

Table Grid

This area contains the grid cells. This area is for making connections between pins and functions.

Table Grid Cell Legend:

Icon	Description
	The cell is currently unavailable and cannot be selected.
	The cell is available for selection.
	The cell has been locked by the user.
D	The cell is a special debug indicator. This cell does not actually lock to a pin but is a visual debug reminder. This indicator means that the pin this cell resides on will be appropriated for debugging purposes based on the currently selected debug options.
	This cell has been automatically locked based on the available choices. This selection takes function priority into account. This lock cannot be changed by the user.
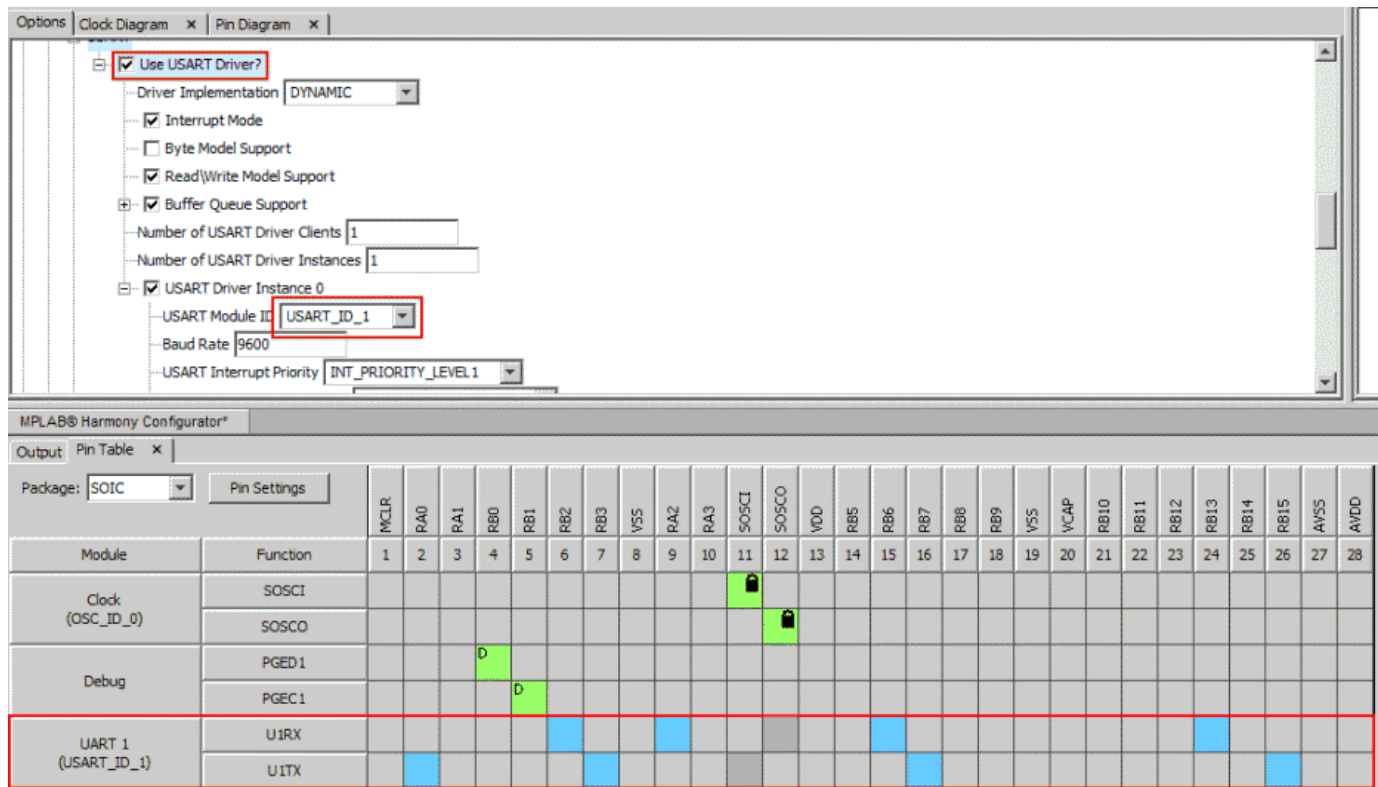
Module Management

Describes the module management features.

Description

The Pin Manager table displays modules based on selections made in the configuration tree.

Observe that by enabling the USART driver instance that the USART1 module appears in the pin table.

Now increase the number of USART driver instances to 2. Once the second USART instance is set to USART_ID_2, the table will display the second USART module.

Options | Clock Diagram x | Pin Diagram x |

```
⊟ USART
   ⊟ ☑ Use USART Driver?
        Driver Implementation  [DYNAMIC        ▼]
        ☑ Interrupt Mode
        ☐ Byte Model Support
        ☑ Read\Write Model Support
   ⊞ ☑ Buffer Queue Support
        Number of USART Driver Clients [1          ]
        Number of USART Driver Instances [2         ]
   ⊞ ☑ USART Driver Instance 0
   ⊟ ☑ USART Driver Instance 1
        USART Module ID  [USART_ID_2   ▼]
        Baud Rate [9600          ]
        USART Interrupt Priority  [INT_PRIORITY_LEVEL1      ▼]
        USART Interrupt Sub-priority  [INT_SUBPRIORITY_LEVEL0 ▼]
        Operation Mode  [DRV_USART_OPERATION_MODE_NORMAL  ▼]
        Operation Mode Data (hexadecimal) [0x00       ]
        ☐ Wake On Start
```

MPLAB® Harmony Configurator*

Output | Pin Table x |

Package: [SOIC ▼]   [Pin Settings]

Module	Function	MCLR	RA0	RA1	RB0	RB1	RB2	RB3	VSS	RA2	RA3	SOSCI	SOSCO	VDD	RB5	RB6	RB7	RB8	RB9	VSS	VCAP	RB10	RB11	RB12	RB13	RB14	RB15
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
Clock (OSC_ID_0)	SOSCI											🔒															
	SOSCO												🔒														
Debug	PGED1				D																						
	PGEC1					D																					
UART 1 (USART_ID_1)	U1RX						▦			▦		▦				▦									▦		
	U1TX		▦					▦		▦					▦				▦								▦
UART 2 (USART_ID_2)	U2RX			▦		▦									▦		▦					▦		▦			
	U2TX				▦				▦							▦			▦		▦				▦		

The U1RX, U1TX, U2RX, and U2TX functions are Peripheral Pin Select functions and can be assigned to multiple pins. Blue cells indicate a potential pin-to-function lock. Observe that left-clicking the blue cell corresponding to pin 9 and U1RX locks that cell to that pin/function pair. U1RX is now assigned to pin 9. Observe also that the name above pin 9 has changed to indicate the locked function, as well as the name of pin 9 in the pin diagram.

With pin 9 locked, the other options for pin 9 and U1RX are now marked unavailable.

The green cell can be left-clicked again to unlock the pin and function.

Conflict Resolution

Describes conflict resolution features.

Description

The Pin Manager uses automatic conflict resolution to determine the proper function when multiple options are available.

Consider the available functions for pin 12: SOSCO/RPA4/T1CK/CTED9/PMA1/RA4. Observe that the SOSCO function was given automatic priority over RPA4 (U1RX).

Output | Pin Table ✕

Package: SOIC ▾   Pin Settings

Module	Function	MCLR 1	RA0 2	RA1 3	RB0 4	RB1 5	RB2 6	RB3 7	VSS 8	RA2 9	RA3 10	SOSCI 11	SOSCO 12	VDD 13	RB5 14	RB6 15	RB7 16	RB8 17	RB9 18	VSS 19	VCAP 20	RB10 21	RB11 22	RB12 23	RB13 24	RB14 25	RB15 26	AVSS 27	AVDD 28
Clock (OSC_ID_0)	SOSCI											🔒																	
	SOSCO												🔒																
Debug	PGED1				D																								
	PGEC1					D																							
UART 1 (USART_ID_1)	U1RX				▪			▪					▪			▪			▪						▪				
	U1TX		▪				▪						▪					▪								▪			

The output window displays a detailed message of this event.

Module	Function	MCLR 1	CTED1 2	CTED2 3	RB0 4	CTED12 5	CTED13 6	RB3 7	VSS 8	RA2 9	RA3 10	SOSCI 11	SOSCO 12	VDD 13	RB5 14	RB6 15	CTED3 16	CTED10 17	CTED4 18	VSS 19	VCAP 20	CTED11 21	RB11 22	RB12 23	CTPLS 24	CTED5 25	CTED6 26	AVSS 27	AVDD 28
CTMU (CTMU_ID_0)	CTED1		🔒																										
	CTED2			🔒																									
	CTED3																🔒												
	CTED4																		🔒										
	CTED5																									🔒			
	CTED6																										🔒		
	CTED9												⬜																
	CTED10																	🔒											
	CTED11																					🔒							
	CTED12					🔒																							
	CTED13						🔒																						
	CTPLS																								🔒				
Clock (OSC_ID_0)	SOSCI											🔒																	
	SOSCO												🔒																
Debug	PGED1				D																								
	PGEC1						D																						
UART 1 (USART_ID_1)	U1RX					▪		▪					⬜			▪										▪			
	U1TX			▪				▪					▪					▪									▪		

Observe also that with the addition of another lower priority function that the selection does not change. The higher priority function SOSCO (red) is still automatically selected while lower priority functions RPA4 (PPS) and OC1 are disabled.

Module	Function	1 MCLR	2 CTED1	3 CTED2	4 RB0	5 CTED12	6 CTED13	7 RB3	8 VSS	9 RA2	10 RA3	11 SOSCI	12 SOSCO	13 VDD	14 RB5	15 RB6	16 CTED3	17 CTED10	18 CTED4	19 VSS	20 VCAP	21 CTED11	22 RB11	23 RB12	24 CTPLS	25 CTED5	26 CTED6	27 AVSS	28 AVDD
CTMU (CTMU_ID_0)	CTED1		🔒																										
	CTED2			🔒																									
	CTED3																🔒												
	CTED4																		🔒										
	CTED5																									🔒			
	CTED6																										🔒		
	CTED9												[green box]																
	CTED10																	🔒											
	CTED11																					🔒							
	CTED12					🔒																							
	CTED13						🔒																						
	CTPLS																								🔒				
Clock (OSC_ID_0)	SOSCI											🔒																	
	SOSCO												🔒 (red)																
Debug	PGED1				D																								
	PGEC1					D																							
UART 1 (USART_ID_1)	U1RX				▢			▢					[cyan box]		▢										▢				
	U1TX		▢				▢					▢				▢											▢		

If the highest priority is a Peripheral Pin Select function (red highlight) a choice is given to the user. The next lowest priority function is automatically selected (blue highlight), but this can be overridden by user action.

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
CTMU (CTMU_ID_0)	CTED1		🔒																										
	CTED2			🔒																									
	CTED3																🔒												
	CTED4																		🔒										
	CTED5																									🔒			
	CTED6																										🔒		
	CTED9												🔒 (blue)																
	CTED10																	🔒											
	CTED11																					🔒							
	CTED12					🔒																							
	CTED13						🔒																						
	CTPLS																								🔒				
Debug	PGED1				D																								
	PGEC1					D																							
UART 1 (USART_ID_1)	U1RX				▢			▢					[red box]		▢										▢				
	U1TX		▢				▢					▢			▢												▢		

If the Peripheral Pin Select function (red highlight) is manually selected then the automatic choice (blue highlight) is overridden. A conflict is still reported. If the Peripheral Pin Select function is unlocked then the lower priority function will be automatically locked again.

Pin Table Features

Describes pin table features.

Description

The Pin Table can be reconfigured to show as little or as much information as the user desires. For example, individual pin rows can be hidden or

isolated depending on how much information is desired. This is accomplished by right-clicking on a pin number and selecting a desired option from the context menu.

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Clock (OSC_ID_0)	SOSCI											▣																	
	SOSCO												▣																
Debug	PGED1				D																								
	PGEC1					D																							

To remove pin 18 from the table, right-click the pin 18 number box. Select **Hide** from the context menu.

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
	CTED1		▣																Hide										
	CTED2			▣															Isolate										
	CTED3																▩		Flags										
	CTED4															▩			View ▶										
	CTED5																									▣			
CTMU (CTMU_ID_0)	CTED6																										▣		
	CTED9													▩															
	CTED10																	▩											
	CTED11																		▩										
	CTED12					▣																							

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20	21	22	23	24	25	26	27	28
	CTED1		▣																									
	CTED2			▣																								
	CTED3																▩											
	CTED4																											
	CTED5																								▣			
CTMU (CTMU_ID_0)	CTED6																									▣		
	CTED9													▩														
	CTED10																	▩										
	CTED11																					▩						
	CTED12					▣																						

Observe that pin 18 has been removed from the table. To restore the column, right-click in the table and select **Show > All** or navigate the available sub-menus and select pin 18.

Module	Function	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	19	20	21	22	23	24	25	26	27	28
	CTED1		▣																									
	CTED2			▣																								
	CTED3														View ▶													
	CTED4														Show ▶	All												
	CTED5															Pins ▶	18								▣			
CTMU (CTMU_ID_0)	CTED6																									▣		
	CTED9													▩														
	CTED10																	▩										
	CTED11																					▩						
	CTED12					▣																						

The table can also be reduced to show only desired pins and functions by using the "Isolate" command. To show only pin 18, again right-click on the pin 18 number box and select **Isolate**.

This functionality also exists for pin modules, functions, and ports.

The table can also be modified by right-clicking the pin boxes in the pin diagram.

The table can also be reconfigured to display pins according to their respective ports. To do this, right-click the table, navigate to the View sub-menu, and select **Ports**. The top row is the original pin number, the middle row shows the port grouping, and the bottom row is the pin's number inside the port grouping. Ports can also be hidden and isolated in the same manner as pins, modules, and functions. This is accomplished by right-clicking on the port name box.

		2	3	9	10	12	4	5	6	7	11	14	15	16	17	18	21	22	23	24	25	26
				A							B											
Module	Function	0	1	2	3	4	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Clock (OSC_ID_0)	SOSCI										🔒											
	SOSCO					🔒																
Debug	PGED1						D															
	PGEC1							D														
UART 1 (USART_ID_1)	U1RX												🟩									
	U1TX									🟩												
UART 2 (USART_ID_2)	U2RX							🟩														
	U2TX															🟩						

Change Notification and Non-PPS Devices

Describes handling change notification for non-PPS devices.

Description

For non PPS parts, change notifications behave differently. They must be explicitly enabled in the configuration tree.
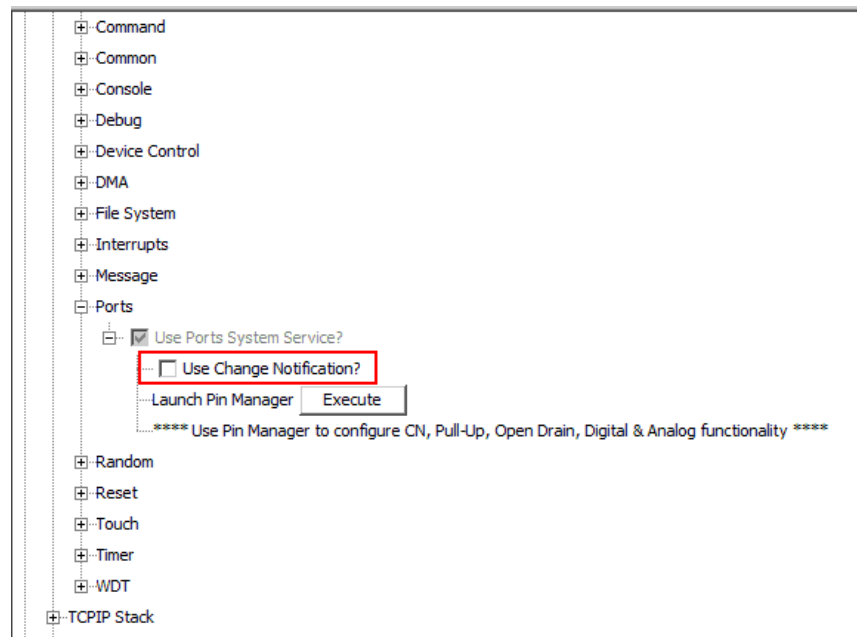
When enabled, the Change Notification module appears in the table. Change notification cells behave similarly to Peripheral Pin Select functions. They will be overridden by higher priority functions, but will provide a user choice if they are the highest priority.

The pin flag dialog also behaves differently for Non-PPS parts. The "Change Notification", "Pull Up", and "Pull Down" options are disabled.

Exporting Pin Mapping

Provides information on exporting pin mappings.

Description

The MPLAB Harmony Graphical Pin Manager provides the ability to export the pin mapping of the current configuration into Excel in `.xls` format for the purpose of printing out the pin mapping. Refer to Importing and Exporting Data for the steps to export the pin mapping.

Importing and Exporting Data

Provides information on importing and exporting data to/from the MHC.

Description

The MPLAB Harmony Configurator provides several options for importing and export various types of data to and from the application. The import and export icons can be found in the main window toolbar.
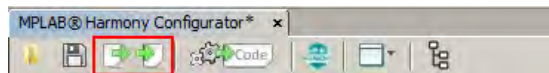
The Import dialog shows the various data sources that can be imported into MPLAB Harmony Configurator. To import, select an item from the list and click **Import**.
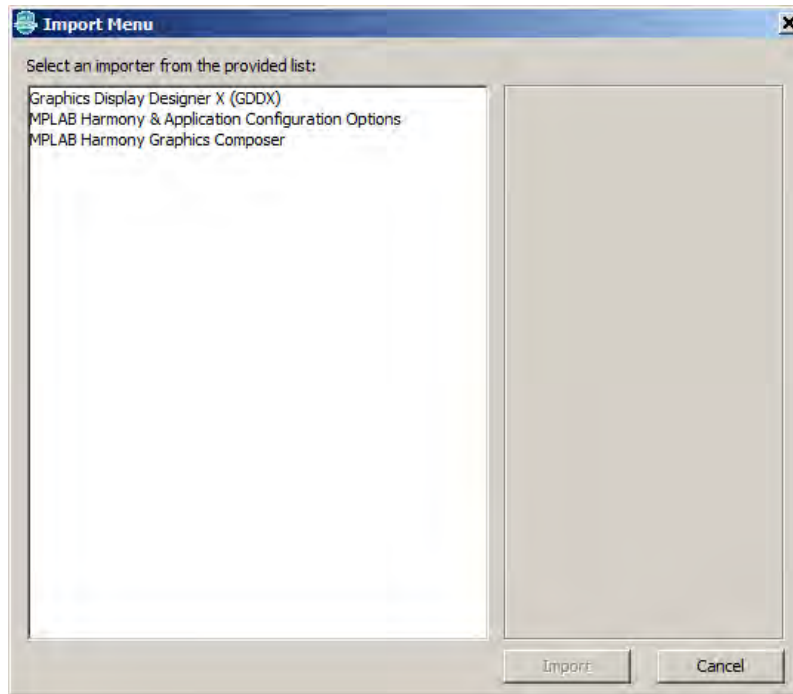
The Export dialog shows the various data sources that can be exported from MPLAB Harmony Configurator. To export, select an item from the list and click **Export**.
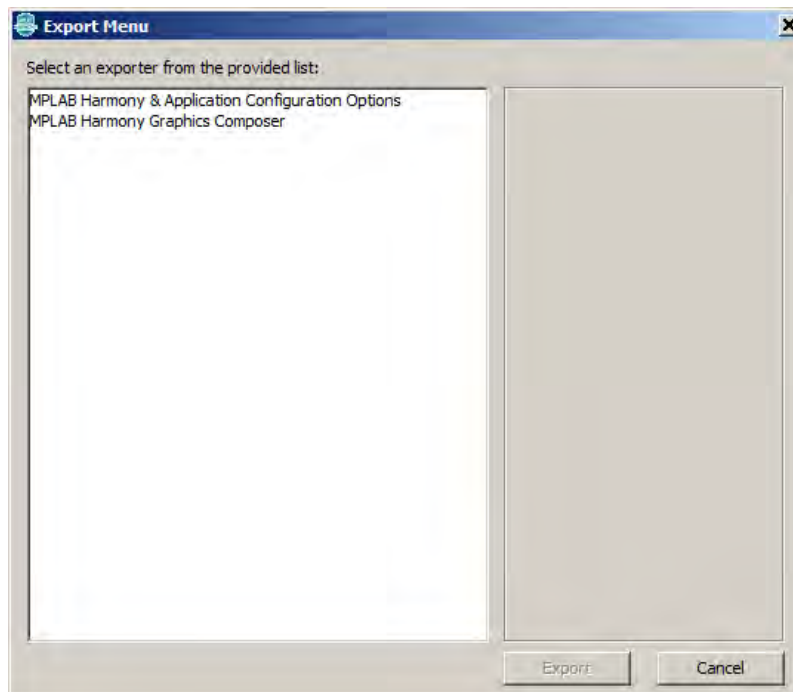
Importing and Exporting MPLAB Harmony Configurator Configuration Options

By selecting **MPLAB Harmony & Application Configuration Options** from either the Import or Export dialog, the user has the ability to create or import .mhc files with only user-selected options.

The following figure provides an example of the option export dialog.
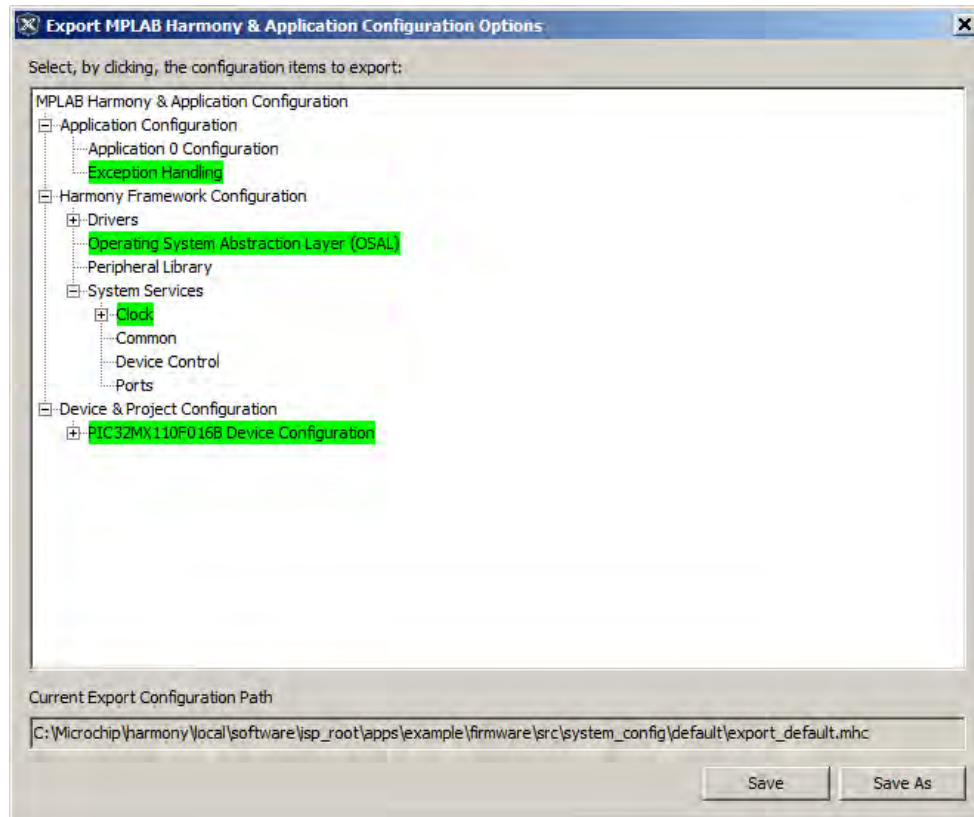
To use this feature, left-click any desired option to toggle its state. Green-highlighted options will be exported. Then, use the **Save** and **Save As** buttons as desired to write the file.

To import, select the option import from the Import dialog and select the previously exported file. Observe that only the exported options are visible in the import window. The user can again select and highlight items in green to select them for import. When all desired settings have been highlighted, click **Import**.
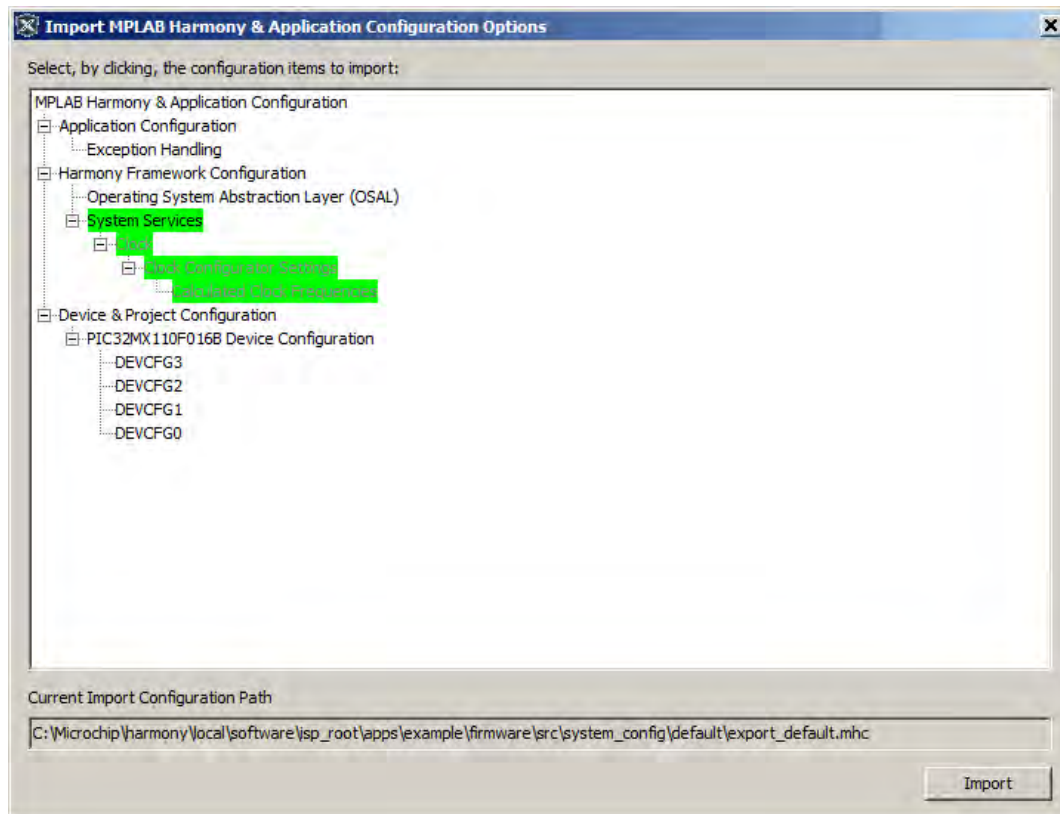
MPLAB Harmony Configurator Developer's Guide

This section describes the basic operation of the MPLAB Harmony Configurator (MHC), the details of the hconfig, template, and `.mhc` files it utilizes, and explains how to add support for new libraries that are compatible with MPLAB Harmony into the MHC.

CONFIG_DEVICE

This hconfig symbol can be used to provide the device ID based on the device selected in MPLAB X IDE. This feature is useful if hconfig/FTL logic that is unique to a device variant needs to be added.

The following example shows the FTL code to perform a check for a specific device:

```
<#if CONFIG_DEVICE == "PIC32MZ2028ECM144">
  … perform device-specific code …
<#/if>
```

Introduction

This topic provides an introduction and overview of the MPLAB Harmony Configurator (MHC).

Description

When installed into MPLAB X IDE, the MHC plug-in provides a "New MPLAB Harmony" project wizard and a graphical user interface for configuration of MPLAB Harmony projects. When used, it generates (or updates) a project outline, including the C-language main function and system configuration files and stores the project configuration selections for later retrieval, modification, and sharing. To do this, the MHC utilizes a completely data driven method for defining the configuration options presented to the user and a template driven method for generating the source code, as illustrated in the following diagram.
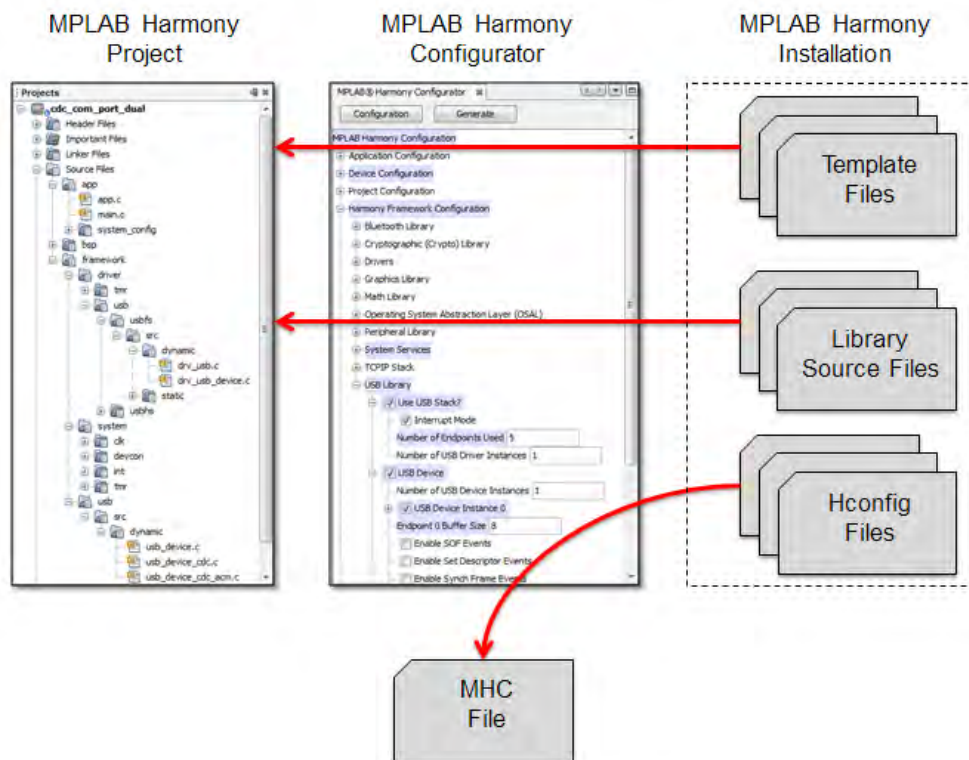
Libraries are primarily provided in the MPLAB Harmony installation in source form. Each library provides an hconfig file and a set of template (`.ftl`) files. The hconfig files are text files that use an extended version of the Linux Kconfig grammar to define the configuration options available for the associated library and to identify source files, dependencies, and help content. When launched from within MPLAB X IDE, the MHC reads the hconfig files and presents the libraries and options to the user for selection and configuration in a graphical tree-based format similar to the Linux Xconfig utility.

After the user makes the desired library and configuration option selections and clicks **Generate**, the MHC stores the selections in another text file named for the current project configuration (as defined by the IDE) with an `.mhc` extension. Then, it processes the basic MPLAB Harmony template files, along with the templates for the selected libraries, using the Java FreeMarker engine to replace the markup text in the template files with the selections made by the user. It then generates the configuration-specific C-language source files for the current configuration of the current main project in the MPLAB X IDE. It also inserts the appropriate source (and/or binary) files for the selected libraries into the MPLAB X IDE project.

After the MHC generates the configuration, the resultant project will build and run, but it may not do anything useful until the user implements the

desired application code.

Adding New Libraries

This section provides information on adding a new library to MPLAB Harmony.

Description

The process of adding a new library that is supported by MHC to a MPLAB Harmony installation consists of the following basic steps.

1. Develop a new MPLAB Harmony compatible library module.
2. Develop the hconfig file to define the library's configuration options and insert it into the MPLAB Harmony hconfig hierarchy.
3. Develop the FreeMarker templates to generate the necessary configuration-specific source code.
4. Insert the new FreeMarker templates into the MPLAB Harmony top-level templates.
5. Install the library source (and other supporting) files into the appropriate locations in the MPLAB Harmony installation tree.
6. Insert the library's documentation into the MPLAB Harmony documentation index.

These steps are described in detail in the following sections.

Developing a Library That is Compatible With MPLAB Harmony

This provides information on compatibility.

Description

MPLAB Harmony libraries need to be modular and inter-operable so that they can be configured for one or more of the different environments supported by MPLAB Harmony. To develop a library that is compatible with MPLAB Harmony, it must meet the design and implementation guidelines as described in the MPLAB Harmony Compatibility Guide. Please refer to this section for the modularity, flexibility, testing, and documentation guidelines that are required and recommended for MPLAB Harmony. Please ensure that any library added to the MPLAB Harmony framework meets these guidelines.

Developing a New hconfig File

This topic lists and describes the steps necessary when developing a new hconfig file.

Description

The configuration options for a library are wholly defined within the hconfig file(s) associated with that library. This section describes how to create an hconfig file for a new driver module. The process is as follows:

- Step 1: Create the File and Insert it into the hconfig Hierarchy
- Step 2: Create a Menu Item for the Module in the Driver Framework Tree
- Step 3: Creating Configuration Options
- Step 4: Use Dependencies
- Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another
- Step 6: Sourcing hconfig Files
- Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement
- Step 8: Add Help Links to Configuration Options
- Step 9: Create Multiple Module Instances

## Step 1: Create the File and Insert it into the hconfig Hierarchy

This topic describes how to create the hconfig file and insert it into the hconfig hierarchy.

Description

Our module example will be a MPLAB Harmony driver named "hconfig_example", and will be inserted into the `<$HARMONY_VERSION_PATH>/framework/driver` directory.
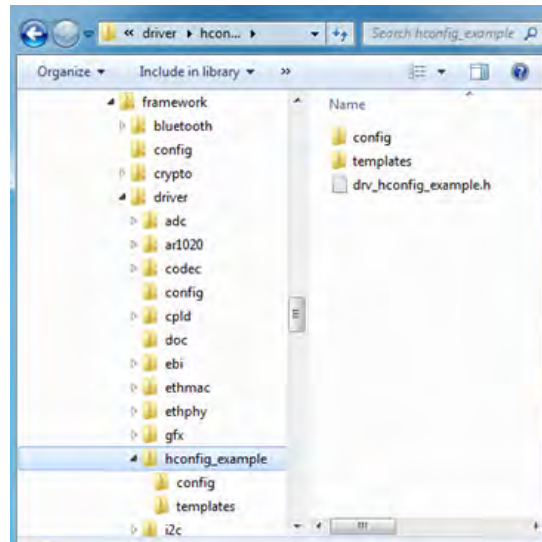
Let's create an hconfig file for the hconfig_example driver, and put it in the `config` folder. For now, all it will do is create a menu entry and a single Boolean config item "Use Hconfig Example?". Note that by default, the driver is not selected.
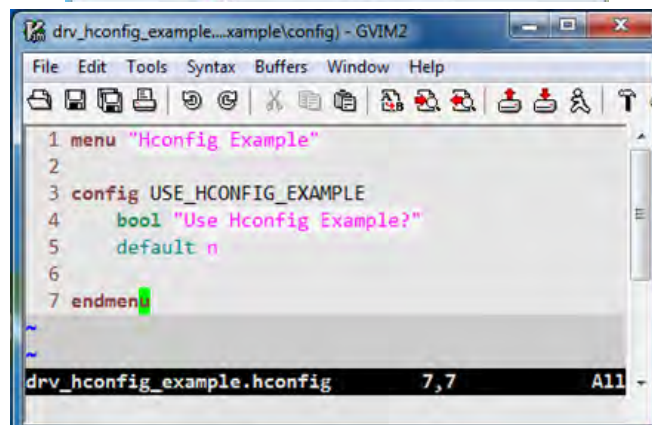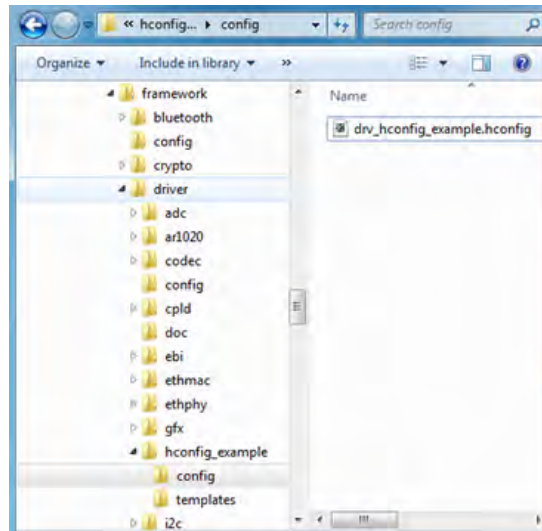
Now we need to insert our hconfig file into the hconfig tree hierarchy so it will be invoked when we run MHC.

Driver hconfig files are sourced from the `<$HARMONY_VERSION_PATH>/framework/driver/config/driver.hconfig` file.

```
 1 menu "Drivers"
 2
 3 config DRIVER
 4     bool
 5
 6 source "$HARMONY_VERSION_PATH/framework/driver/ar1020/config/drv_ar1020.hconfig"
 7 source "$HARMONY_VERSION_PATH/framework/driver/ebi/config/drv_ebi.hconfig"
 8 #source "$HARMONY_VERSION_PATH/framework/driver/ethmac/config/drv_ethmac.hconfig"
 9 #source "$HARMONY_VERSION_PATH/framework/driver/ethphy/config/drv_ethphy.hconfig"
10 source "$HARMONY_VERSION_PATH/framework/driver/gfx/config/drv_gfx.hconfig"
11 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example.hconfig"
12 source "$HARMONY_VERSION_PATH/framework/driver/i2s/config/drv_i2s.hconfig"
13 source "$HARMONY_VERSION_PATH/framework/driver/ic/config/drv_ic.hconfig"
14 source "$HARMONY_VERSION_PATH/framework/driver/nvm/config/drv_nvm.hconfig"
15 source "$HARMONY_VERSION_PATH/framework/driver/oc/config/drv_oc.hconfig"
16 source "$HARMONY_VERSION_PATH/framework/driver/pmp/config/drv_pmp.hconfig"
17 source "$HARMONY_VERSION_PATH/framework/driver/rtcc/config/drv_rtcc.hconfig"
18 source "$HARMONY_VERSION_PATH/framework/driver/sdcard/config/drv_sdcard.hconfig"
19 source "$HARMONY_VERSION_PATH/framework/driver/spi/config/drv_spi.hconfig"
20 source "$HARMONY_VERSION_PATH/framework/driver/tmr/config/drv_tmr.hconfig"
21 source "$HARMONY_VERSION_PATH/framework/driver/usart/config/drv_usart.hconfig"
22 #source "$HARMONY_VERSION_PATH/framework/driver/usb/config/drv_usb.hconfig"
23 #source "$HARMONY_VERSION_PATH/framework/driver/wifi/config/drv_wifi.hconfig"
24 endmenu
25
26 ifblock DRIVER
27 file DRIVER_H "$HARMONY_VERSION_PATH/framework/driver/driver.h"
28 endif
```

Added (line 11)

Note that all hconfig files are included recursively by the top-level hconfig file in the application's firmware directory. The entire hconfig tree is parsed when MHC is invoked and when a configuration change is made, so the relative placement of configuration options only affects the menu structure. There is no functional dependency.

Step 2: Create a Menu Item for the Module in the Driver Framework Tree

This topic describes creating a menu item for the MPLAB Harmony module in the Driver framework tree.

Description

Let's create a demonstration application and see if our driver config appears.
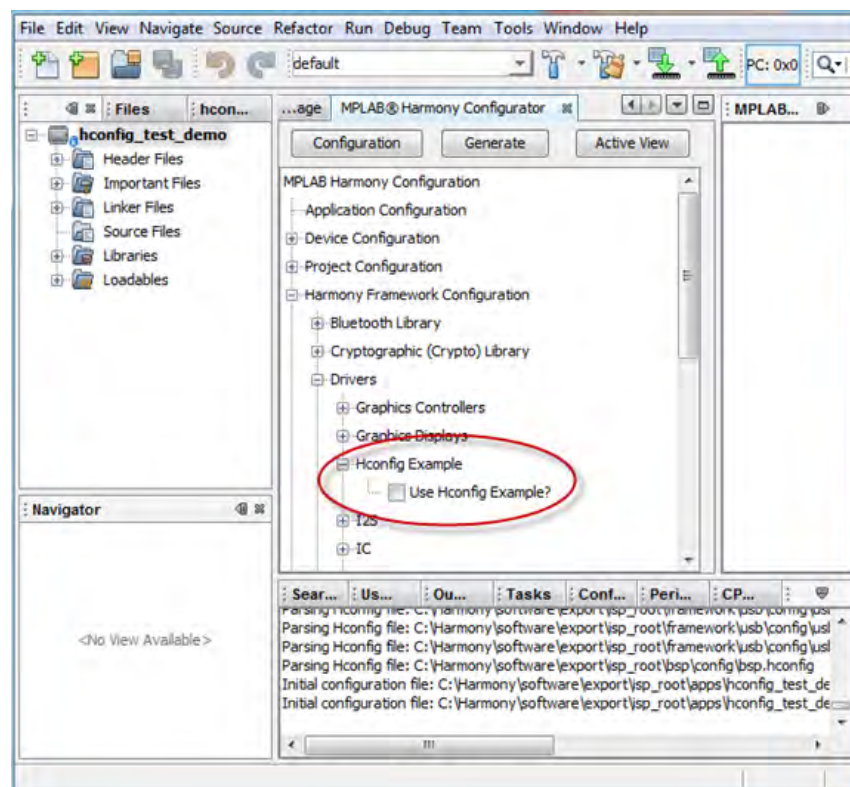
Step 3: Creating Configuration Options

This topic describes adding menu configuration options

Description

Now let's add some config options.

- A config option selected from a drop-down menu
- A Boolean config value
- An integer config whose default value depends on the first config option

```
1 menu "Hconfig Example"
2
3 config USE_HCONFIG_EXAMPLE
4     bool "Use Hconfig Example?"
5     default n
6
7 # 1. A config option selected from a dropdown menu.
8 enum CFG1_VAL
9     "cfg1_val_0"
10    || "cfg1_val_1"
11    || "cfg1_val_2"
12
13 config CFG1
14     depends on USE_HCONFIG_EXAMPLE
15     string "Config 1"
16     range CFG1_VAL
17     default "cfg1_val_0"
18
19 # 2. A boolean config value
20 config CFG2
21     depends on USE_HCONFIG_EXAMPLE
22     bool "Config 2"
23     default y
24
25 # 3. An integer config whose default value depends on CFG1.
26 config CFG3
27     depends on USE_HCONFIG_EXAMPLE
28     int "Config 3"
29     range 1 3
30     default 1 if CFG1 = "cfg1_val_0"
31     default 2 if CFG1 = "cfg1_val_1"
32     default 3 if CFG1 = "cfg1_val_2"
33
34 endmenu
```
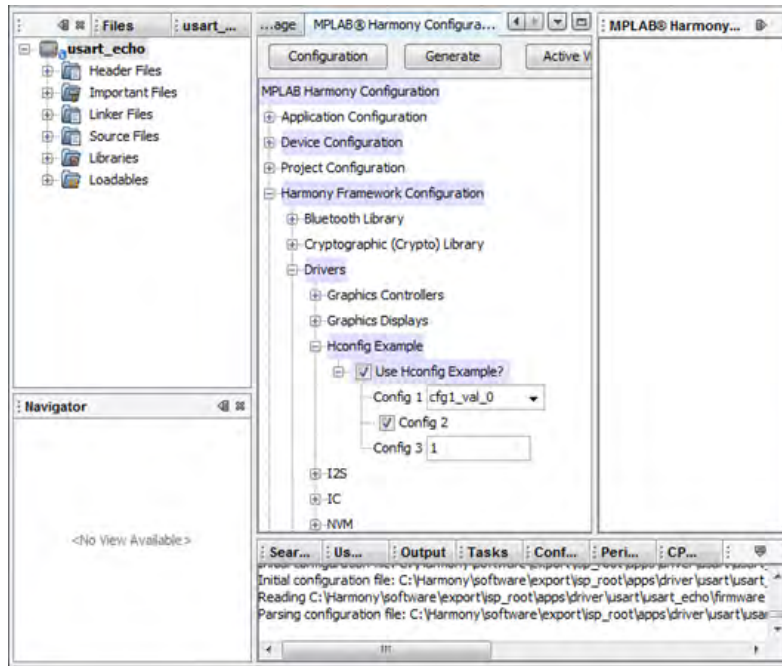
Step 4: Use Dependencies

This topic describes use dependencies.

Description

Note that all config options have a dependency on USE_HCONFIG_EXAMPLE. This means that they will not be visible in the MHC menu unless USE_HCONFIG_EXAMPLE is true. Also note the range on CFG3. An attempt to set CFG3 to a value outside the listed range will be flagged as an error in MHC.

The default value of "Config 3" is set according to Config 1. The first true default in a config block becomes the default value of the config option, and any subsequent default statements are ignored. Therefore, if we add a default with no `if` clause ahead of the other defaults, it will become the default of the config option regardless of whether or not any of the others are true.

A config option may contain multiple dependencies. Both dependencies and `if` statements can contain logical `AND` and `OR`.

```
33 config CFG4
34     depends on USE_HCONFIG_EXAMPLE && CFG2
35     bool "Config 4"
36     default n
37
38 config CFG5
39     depends on USE_HCONFIG_EXAMPLE && CFG2
40     depends on CFG4
41     bool "Config 5"
42     default y if (CFG1 = "cfg1_val_0") && CFG4
43
```

Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another

This topic describes using the "choice" and "select" statements to enable a module to be used by another module.

Description

You can make config options mutually exclusive with the "choice" statement. This is useful for modules that can be configured to operate in different modes. A choice block requires a prompt, which is displayed in the hconfig tree. Choice blocks can optionally have a default option and dependencies. If no default is provided, the choice block will be flagged in red until one of the config options is checked.

```
45 choice
46     prompt "Driver Mode"
47     depends on USE_HCONFIG_EXAMPLE
48     default CFG7
49 config CFG6
50     bool "Config 6"
51
52 config CFG7
53     bool "Config 7"
54
55 config CFG8
56     bool "Config 8"
57 endchoice
```
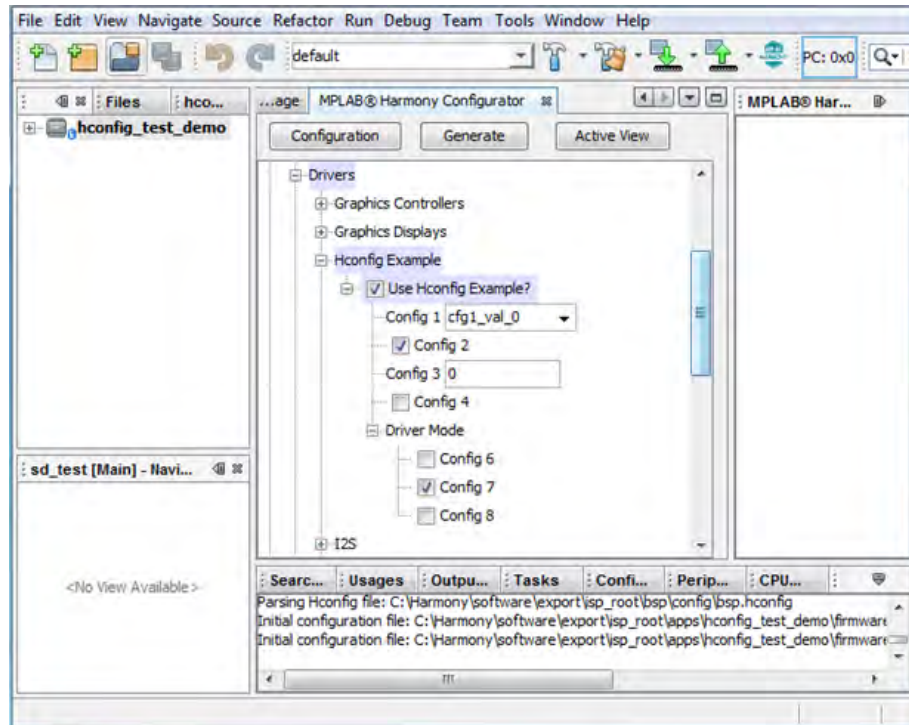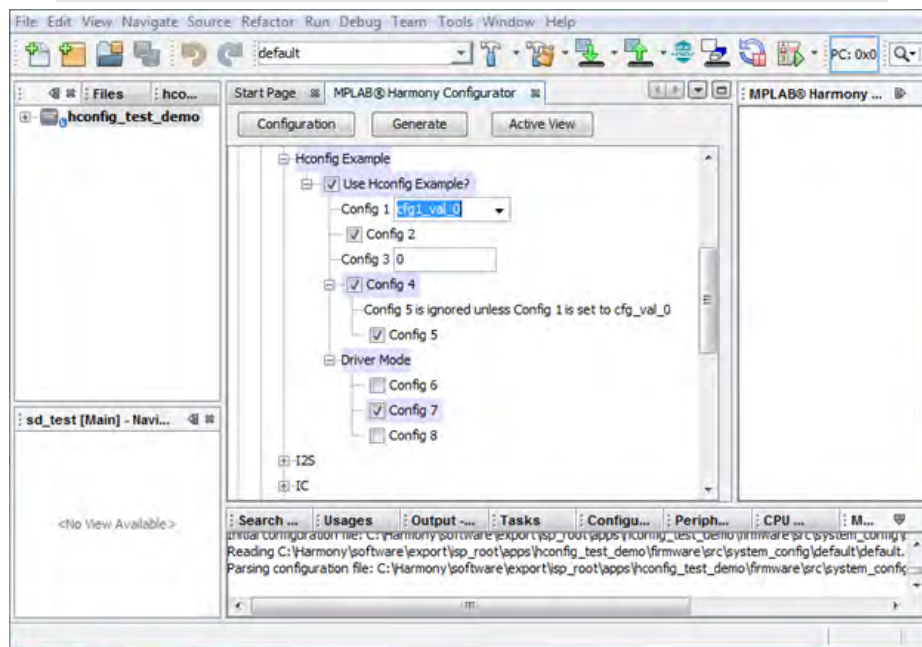
Comments can be displayed in the menu with the "comment" statement. Comments can also have dependencies.

```
39 comment "Config 5 is ignored unless Config 1 is set to cfg_val_0"
40     depends on CFG4
41 config CFG5
42     depends on USE_HCONFIG_EXAMPLE && CFG2
43     depends on CFG4
44     bool "Config 5"
45     default y if (CFG1 = "cfg1_val_0") && CFG4
46
```


The "select" statement is used to select or enable a config option based on another config option. This is often used to enable a module that may be required by multiple modules. An example is the Interrupt System Service, which is used by many drivers and system services.

The select statement must be part of a config block. It should only be used to select non-visible config options. The reason for this is that once a config option is selected, it cannot be unselected. Even if the config option is not checked in the menu, it will still be selected in hconfig, and included in the generated code.

```
60 config CFG9_NEEDED
61     bool
62
63 config CFG9
64     depends on CFG9_NEEDED
65     bool "Config 9"
66     default y if CFG9_NEEDED
67     default n
68
69 config CFG10
70     bool "Config 10"
71     default n
72     select CFG9_NEEDED
```
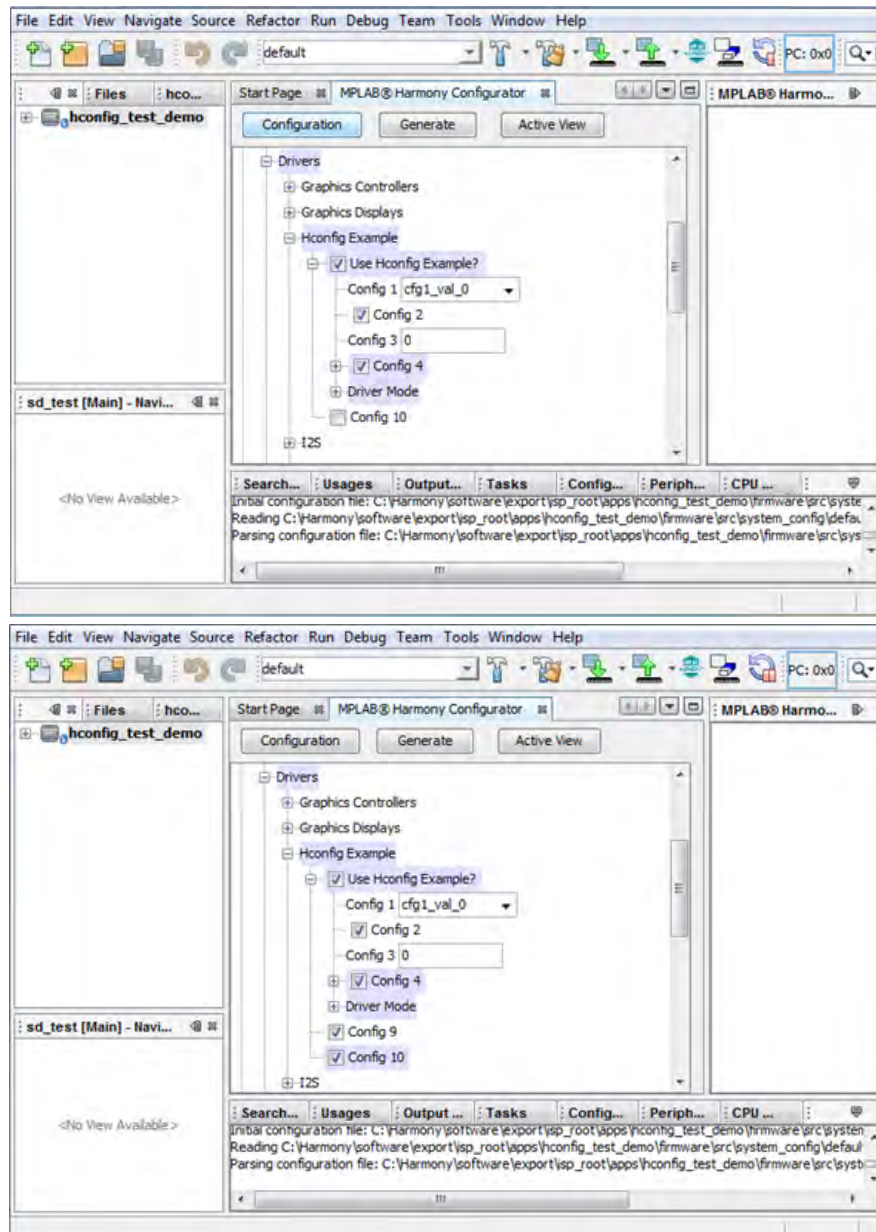

Step 6: Sourcing hconfig Files

This topic describes how to source an hconfig file from another hconfig file.

Description

An hconfig file can source other hconfig files. This is useful for grouping related config options or handling multiple module instances. The sourced file may optionally contain a menu/endmenu block.

Enclosing a source statement within an "ifblock" will apply the dependency to all config options within the sourced file. In the example shown below, all config options in the `drv_hconfig_example_0.hconfig` file are dependent on CFG10. All Ifblock statements must be terminated with `endif`.
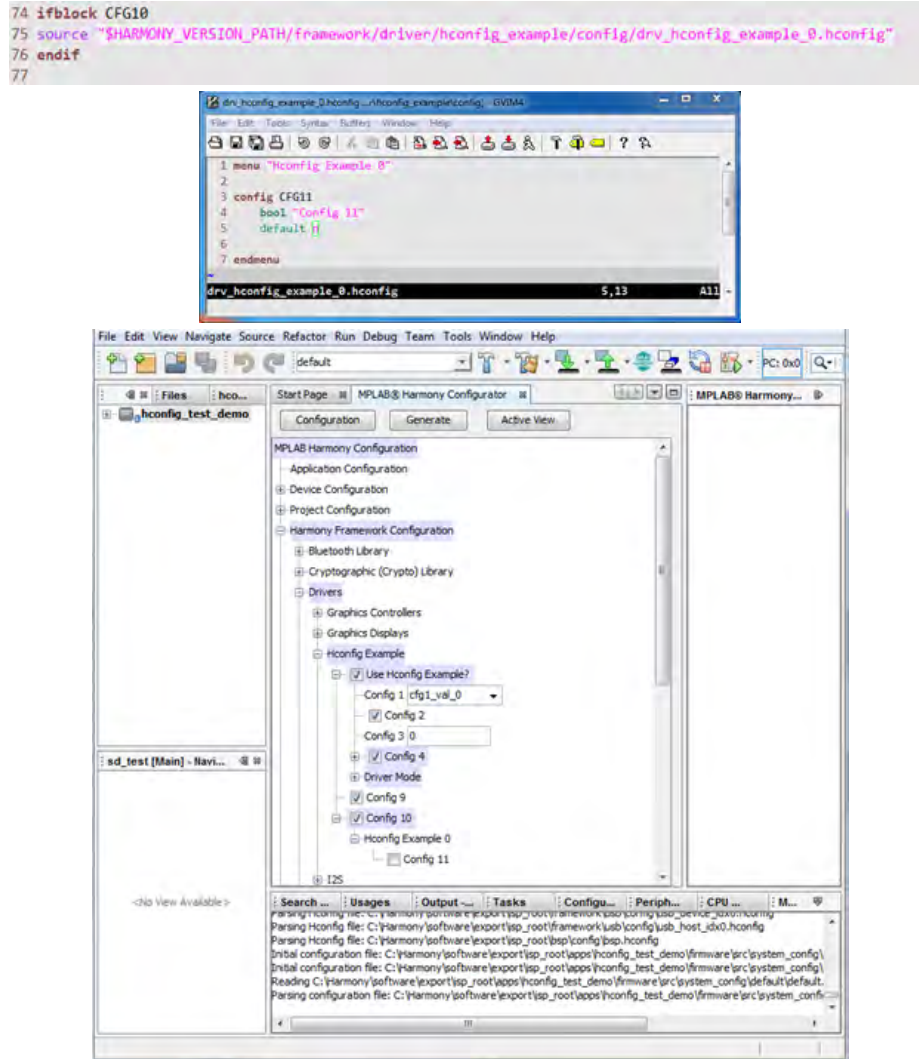
Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement

This topic describes adding source files using the "file" statement.

Description

MHC adds source files to the MPLAB X IDE project with the "file" statement. The full path to the file on disk must be provided, as well as the virtual directory in MPLAB X IDE. The "file" statement does not copy files, it just adds existing files to the MPLAB X IDE project. The files are added when the user clicks **Generate** within MHC.
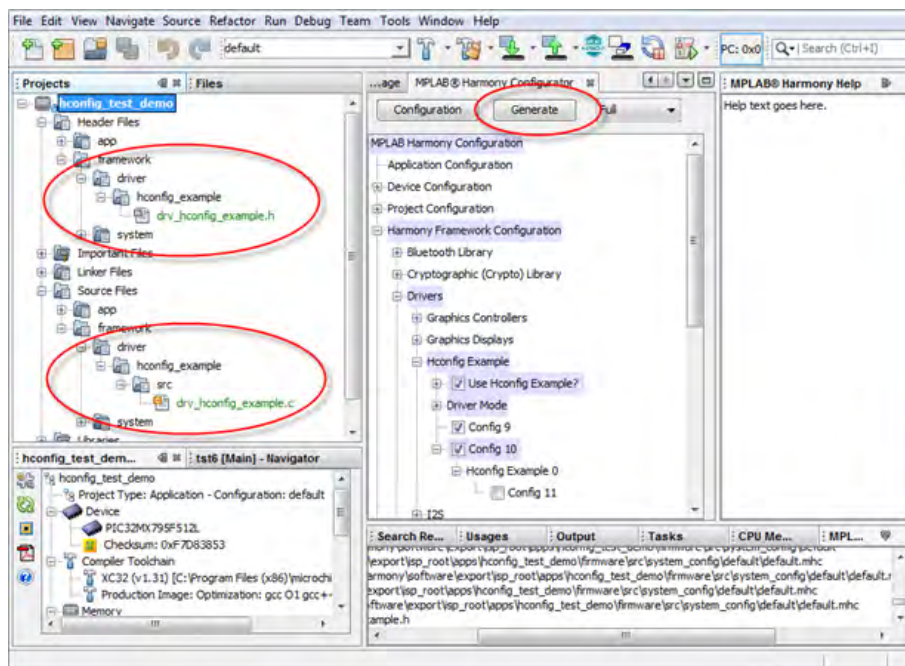
Step 8: Add Help Links to Configuration Options

This topic provides an example for adding Help links.

Description

Each configuration option may have Help text associated with it. In MHC, the Help text is a hyperlink into the MPLAB Harmony documentation. If no link exists, the text itself is displayed in the help window.

```
69  config CFG10
70      bool "Config 10"
71      default n
72      select CFG9_NEEDED
73      ---help---
74      Help text goes here.
75      ---endhelp---
76
```

Step 9: Create Multiple Module Instances

This topic describes the creation of multiple instances.

Description

Several modules support multiple instances, requiring separate configuration options for each instance. In this case, the configuration options of different instances are identical, but may be set to different values. This is handled in MHC by a combination of the "instances" keyword, and a FreeMarker template that is processed once for each instance of the module. As an example, we will create three instances of our hconfig demonstration driver, each containing two configuration options.

The instance template is sourced like a normal hconfig file, but with the keyword "instances" preceded by the maximum number of instances supported. A configuration option is added to allow the user to select the number of instances actually configured and instantiated.

```
77  config DRV_HCONFIG_INSTANCES_NUMBER
78      depends on USE_HCONFIG_EXAMPLE
79      int "Number of Hconfig Driver Instances?"
80      default 1
81      range 1 3
82
83  source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl" 3 instances
84
```
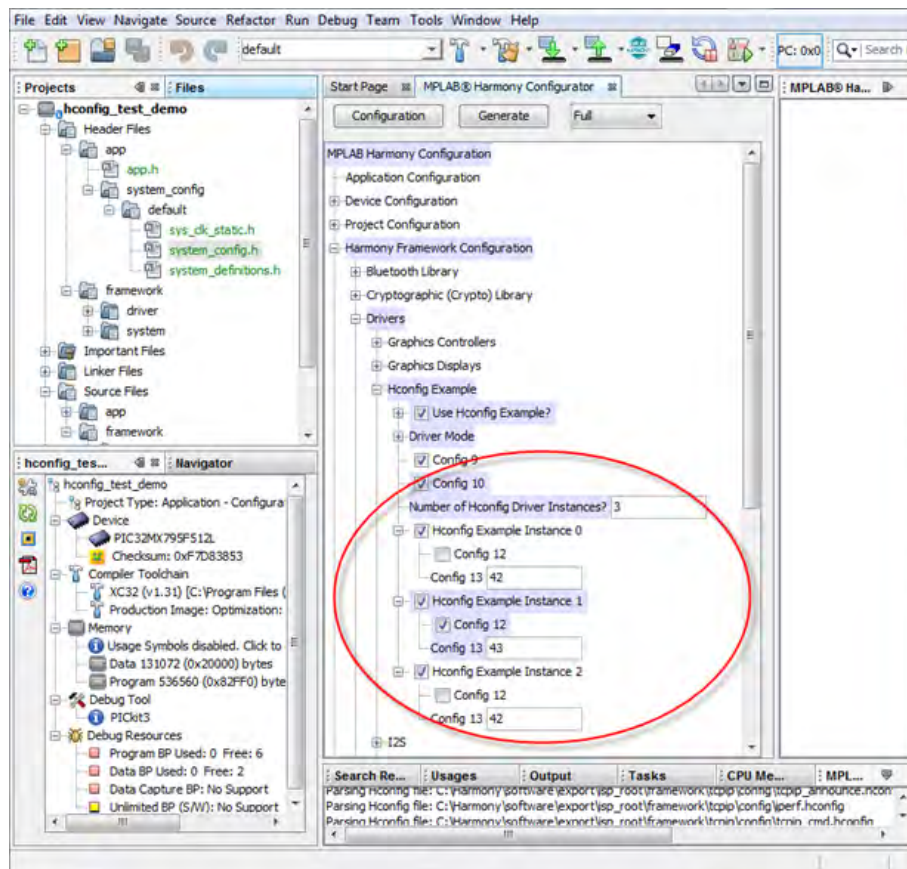
The FreeMarker template is a marked-up hconfig file that is processed through FreeMarker once for each instance. Each time it is processed, the ${INSTANCE} variable is set to the instance number.

```
1 config DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE+1}
2     bool
3 <#if INSTANCE != 0>
4     default n if DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE} = n
5 </#if>
6     default n if DRV_HCONFIG_INSTANCES_NUMBER = ${INSTANCE+1}
7     default y
8
9 config DRV_HCONFIG_INST_IDX${INSTANCE}
10     depends on USE_HCONFIG_EXAMPLE
11 <#if INSTANCE != 0>
12                 && DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE}
13 </#if>
14     bool "Hconfig Example Instance ${INSTANCE}"
15     default y
16
17 ifblock DRV_HCONFIG_CFG12_IDX${INSTANCE}
18
19 config DRV_HCONFIG_IDX${INSTANCE}
20     bool "Config 12"
21     default n
22
23 config DRV_HCONFIG_CFG13_IDX${INSTANCE}
24     int "Config 13"
25     default 42
26
27 endif
```

When MHC is run, the user is prompted for the number of instances. Configuration options for each instance are displayed in the menu.


The FreeMarker templates that are used to generate code must also be updated for multiple instances.

```
44 <#if CONFIG_USE_HCONFIG_EXAMPLE == true>
45
46 // ********************************************************************
47 /* MPLAB Harmony Hconfig Example Driver Configuration Options
48 */
49
50 #define CFG1  ${CONFIG_CFG1}
51 <#if CONFIG_CFG2 == true>
52 #define CFG2   true
53 <#else>
54 #define CFG2   false
55 </#if>
56 #define CFG3  ${CONFIG_CFG3}
57
58 <#-- Instance 0 -->
59 <#if CONFIG_DRV_HCONFIG_INST_IDX0 == true>
60 <#if CONFIG_DRV_HCONFIG_CFG12_IDX0 == true>
61 #define DRV_HCONFIG_CFG12_IDX0   true
62 <#else>
63 #define DRV_HCONFIG_CFG12_IDX0    false
64 </#if>
65 #define DRV_HCONFIG_CFG13_IDX0    ${CONFIG_DRV_HCONFIG_CFG13_IDX0}
66 </#if>
67
68 <#-- Instance 1 -->
69 <#if CONFIG_DRV_HCONFIG_INST_IDX1 == true>
70 <#if CONFIG_DRV_HCONFIG_CFG12_IDX1 == true>
71 #define DRV_HCONFIG_CFG12_IDX1   true
72 <#else>
73 #define DRV_HCONFIG_CFG12_IDX1    false
74 </#if>
75 #define DRV_HCONFIG_CFG13_IDX1    ${CONFIG_DRV_HCONFIG_CFG13_IDX1}
76 </#if>
77
78 <#-- Instance 2 -->
79 <#if CONFIG_DRV_HCONFIG_INST_IDX2 == true>
80 <#if CONFIG_DRV_HCONFIG_CFG12_IDX2 == true>
81 #define DRV_HCONFIG_CFG12_IDX2   true
82 <#else>
83 #define DRV_HCONFIG_CFG12_IDX2    false
84 </#if>
85 #define DRV_HCONFIG_CFG13_IDX2    ${CONFIG_DRV_HCONFIG_CFG13_IDX2}
86 </#if>
87
88 </#if>
```

When the code is generated, code is generated for each instance.
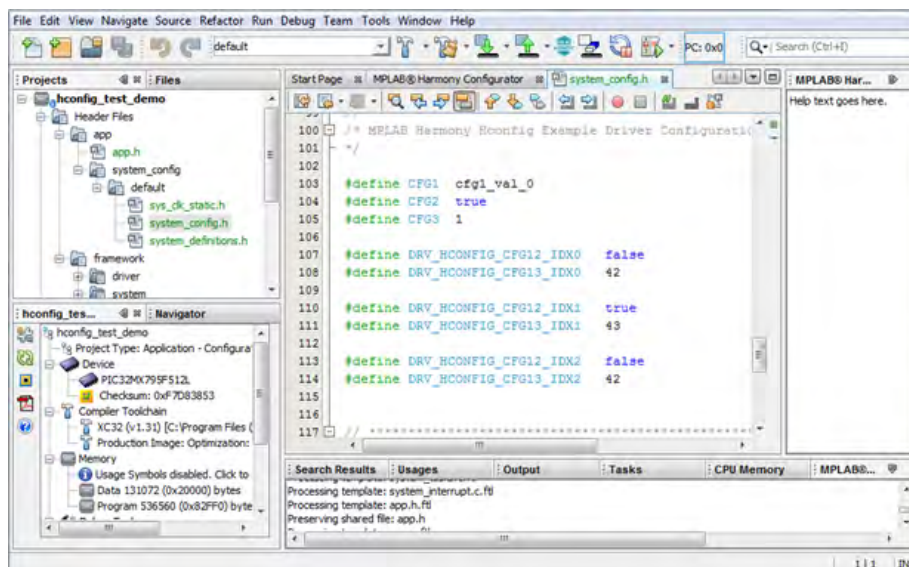
Using the Set Statement

Demonstrates how to use the `set` statement to configure dependencies.

Description

Often one MPLAB Harmony library uses (depends upon) another and has specific requirements on how that library must be configured. To illustrate this, the following Hconfig code MHC Options menu items to allow selection and configuration of a library (library C) that might be shared by other libraries.

Library C Selection and Configuration Menu Definition
```
# Library C Configuration
config USE_LIBRARY_C
    bool "Use Library C?"
    default n

menu "Configure Library C"
    depends on USE_LIBRARY_C

config LIBRARY_C_ITEM_1
    depends on USE_LIBRARY_C
    int "Library C, Item 1: Enter an integer"
    default 0

endmenu  # Configure Library C
```

If another library (library A) requires the use of library C and requires library C's configuration item (LIBRARY_C_ITEM_1) to have a specific value (42), the following Hconfig code will define an MHC options menu to satisfy this requirement.

Library A Selection and Configuration Menu Definition
```
# Library A Configuration
config USE_LIBRARY_A
    bool "Use Library A?"
    default n
    set USE_LIBRARY_C to y if USE_LIBRARY_A = y
    set LIBRARY_C_ITEM_1 to 42 if USE_LIBRARY_A = y

comment "Sets Library C, Item 1 to 42"
    depends on USE_LIBRARY_A

menu "Configure Library A"
    depends on USE_LIBRARY_A

config LIBRARY_A_ITEM_1
    depends on USE_LIBRARY_A
    int "Library A, Item 1: Enter an integer"
    default 0

endmenu  # Configure Library A
```

However, if a second library (library B) also depends on library C, it is possible that the default configuration settings for library C that it requires may be different. This is shown in the following Hconfig code that defines library B's selection and configuration menu, and uses the `set` statement to set library C's item 1 to a value of 86.

Library B Selection and Configuration Menu Definition
```
# Library B Configuration
config USE_LIBRARY_B
    bool "Use Library B?"
    default n
    set USE_LIBRARY_C to y if USE_LIBRARY_B = y
    set LIBRARY_C_ITEM_1 to 86 if USE_LIBRARY_B = y

comment "Sets Library C, Item 1 to 86"
    depends on USE_LIBRARY_B

menu "Configure Library B"
    depends on USE_LIBRARY_B

config LIBRARY_B_ITEM_1
    depends on USE_LIBRARY_B
    int "Library B, Item 1: Enter an integer"
```

```
        default 0
```

```
endmenu  # Configure Library B
```

When such a conflict occurs, the MHC notifies the user, who is then required to enter a value to resolve the conflict (if possible) or disable one of the dependent libraries.

The following sequence of images illustrates the behavior of the MHC when the previous Hconfig code is used. Before any of these libraries have been selected, the MHC Options menu shows their Use Library options.
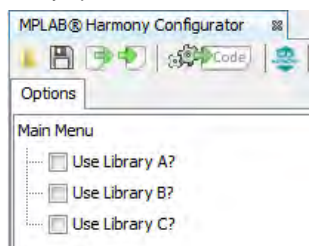

If library A is used (but not library B), the MHC automatically sets the value of library C's configuration item to 42.
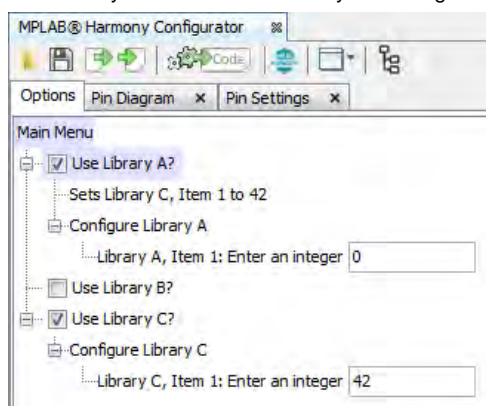

If library B is used (but not library A), the MHC automatically sets the value of library C's configuration item to 86.
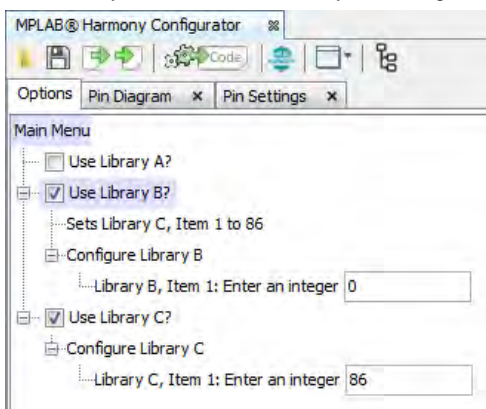
However, if both library A and B are used, the MHC highlights the conflict in library C in red and requires the user to enter a value to resolve the conflict.
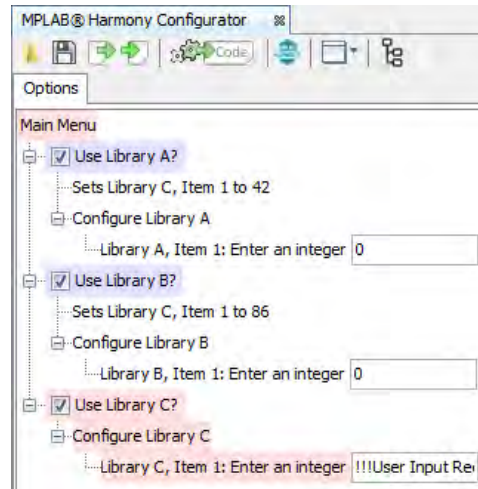
If the user then enters a value for library C's item 1, the MHC recognizes that the user has set that item's value and assumes that the conflict has been resolved.
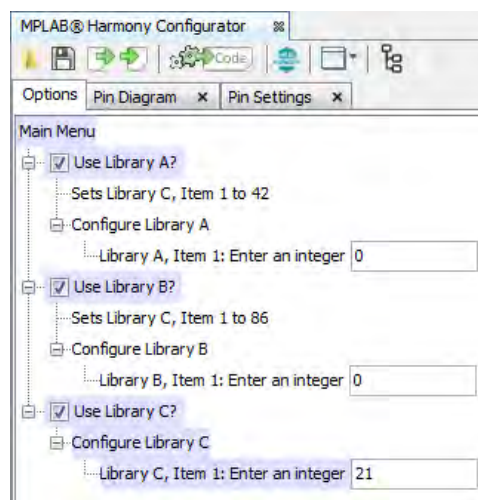
It is important to understand that the MHC does not validate that the chosen value satisfies the requirements of both libraries A and B. It is up to the user to understand the requirements and select an appropriate configuration value.

It is also a good practice to provide a comment in the dependent library's configuration menu when it sets dependencies so that the user knows it has done so.

hconfig Development Guidelines

This topic describes the conventions and guidelines to be used when creating hconfig files.

Description

The following conventions need to be followed when developing MPLAB Harmony hconfig files:

- HAVE_<peripheral> configuration options are used to indicate whether or not a specific peripheral is supported on the device. These options are non-visible, Boolean, and primarily located in `framework.hconfig`. They are set to 'y' using the "select" keyword in the processor-specific peripheral hconfig files. The processor-specific peripheral hconfig files are generated automatically from processor-specific PLIB header files.
- All hconfig files shall be placed in a "config" folder in the MPLAB Harmony framework tree. The hconfig files shall "source" other hconfig files lower in the framework hierarchy. For example the framework hconfig file sources an hconfig file for each folder in the framework directory. The driver hconfig file sources an hconfig file for every driver in the framework/driver directory, and so on.
- The keyword "select" shall not be used with visible config options. Once something is selected using the "select" keyword, it is always selected, regardless of whether or not it is checked in the MHC menu.
- When sourcing an hconfig file within an ifblock, the file is always sourced, and the ifblock dependencies are applied to all items within the sourced file
- Adding the keyword "exclusive" to an enum definition prevents the same element from being assigned to more than one config option
- There can be only one "mainmenu". The top-level hconfig file containing the mainmenu is generated by MHC and placed in the application firmware directory. The template for the top-level hconfig file is located in `utilities/mhc/config`.
- It is often useful to have modules enable each other. The mechanism for this is to use the "select" keyword within one module to select a non-visible config option within another module. The non-visible config option is then used as a dependency for the first module. By convention, the non-visible option is named `USE_<module>_NEEDED`. For example, the Timer System Service requires a Timer Driver instance. The Timer

Driver hconfig contains:
```
config USE_DRV_TMR_NEEDED
     bool
config USE_DRV_TMR
     depends on HAVE_TMR
     bool "Use Timer Driver?"
     default y if USE_DRV_TMR_NEEDED
     default n
```
and the timer system service hconfig contains:
```
config USE_SYS_TMR
     bool "Use Timer System Service?"
     select USE_DRV_TMR_NEEDED
     default y if USE_SYS_TMR_NEEDED
     default n
```
Selecting the Timer System Service automatically selects the Timer Driver, by selecting USE_DRV_TMR_NEEDED, which (if selected) sets the USE_DRV_TMR default to 'y'.

- When multiple default values are given to a config option, the first one that evaluates to *true* becomes the config option value
- By convention, the selection of a module is made in the menu with the menu text "Use <module>?" (e.g., "Use Timer Driver?")
- Modules should default to not-used unless selected by another module
- Visible config options should follow MPLAB Harmony naming conventions
- When selecting module features, menu entries should include a feature rather than exclude, and enable rather than disable. Default for all visible config options should be excluded or disabled, unless needed by another module or enabled by dependencies.
- All visible config options must have an associated help tag, and must be documented in the Help documentation
- Visible config options and comments must capitalize each word in menu text
- Integer config options should have a range whenever possible

Developing MPLAB Harmony FreeMarker Templates

This topic provides information on developing FreeMarker templates.

Description

MHC uses FreeMarker to generate code from template files. The template files use the configuration settings generated from hconfig files to generate code specific to the configuration. A complete description of the FreeMarker language is beyond the scope of this document. Please refer to the online FreeMarker manual, which available at: http://freemarker.org/docs/. This section will illustrate how MHC uses it with a simple example.

The configuration options generated by MHC are written to a <configuration>.mhc file in the project's firmware/src/system_config/<configuration> directory. In our example, the project name is hconfig_test_demo, and the configuration is "default". By default, a number of files are generated by MHC and placed in the application's firmware/src directory. The configuration-specific files are in the firmware/src/ system_config/<configuration> directory. The configuration options are written to the system_config.h file. For this example, we will first show how to create a FreeMarker template for our system_config.h, and insert it into MHC.

For this example, we will use a simple version of drv_hconfig_example.hconfig, with just three config options, CFG1, CFG2, and CFG3.

First, we need to create the template for system_config.h. By convention, this file will be named system_config.h.ftl, and be placed in the <module>/templates directory.
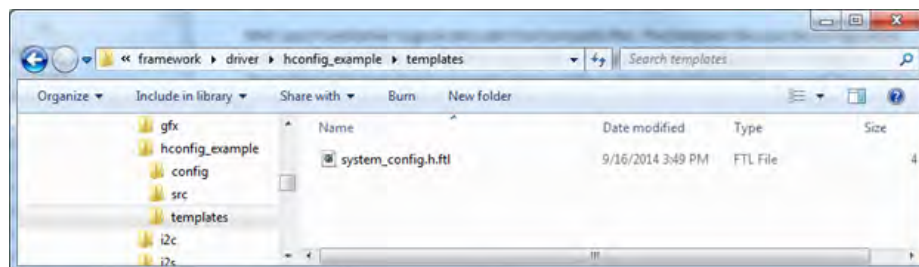
Second, we need to implement the source code template, as shown by the following example.

```
42  ******************************************************************
43  -->
44  <#if CONFIG_USE_HCONFIG_EXAMPLE == true>
45  
46  // *****************************************************************
47  /* MPLAB Harmony Hconfig Example Driver Configuration Options
48  */
49
50  #define CFG1  ${CONFIG_CFG1}
51  <#if CONFIG_CFG2 == true>
52  #define CFG2  true
53  <#else>
54  #define CFG2  false
55  </#if>
56  #define CFG3  ${CONFIG_CFG3}
57
58  </#if>
59  <#--
60  /******************************************************************
```

The source code template will include FreeMarker "markup" statements (defined between the `<#` and `>` escape tags and will use FreeMarker variables (defined between the `${` and `}` escape tags. The FreeMarker statements are interpreted semantically by the FreeMarker engine and the variables are textually replaced using values defined using the MHC by the user and stored in the `.mhc` file.

📝 **Note:**   Symbols defined in hconfig files must be prefixed with `CONFIG_` to use them in FreeMarker templates.

The resulting customized source code is generated directly into the configuration-specific folder of the current project.

Device Configuration

This topic describes the CONFIG_DEVICE hconfig symbol.

Description

CONFIG_DEVICE

This hconfig symbol can be used to provide the device ID based on the device selected in MPLAB X IDE. This feature is useful if hconfig/FTL logic that is unique to a device variant needs to be added.

The following example shows the FTL code to perform a check for a specific device:
```
<#if CONFIG_DEVICE == "PIC32MZ2028ECM144">
  … perform device-specific code …
<#/if>
```

Insert the New FreeMarker Templates into the MPLAB Harmony Top-level Templates

This topic describes how to insert a new FreeMarker template into the MPLAB Harmony top-level templates.

Description

To insert the template into MHC, we need to do one of two things:
- Use the "template" keyword in hconfig, or
- Include this template into another template

Since the `system_config.h` file draws config options from many templates, we will include our template in the top-level `system_config.ftl`. It is located in the `$HARMONY_VERSION_PATH/utilities/mhc/templates/app/system_config` directory.
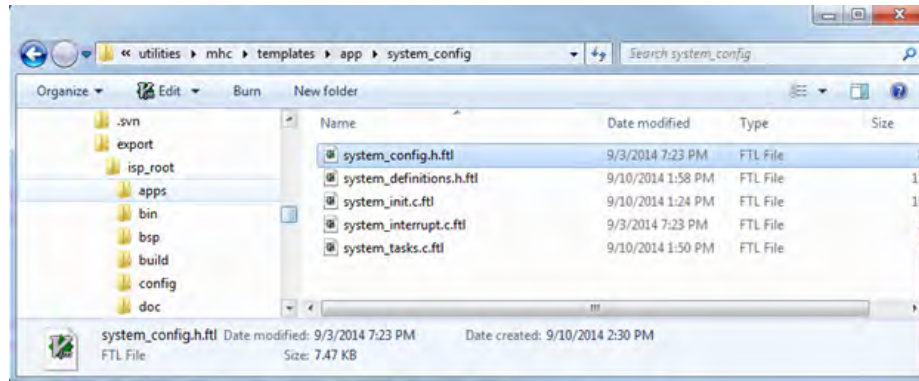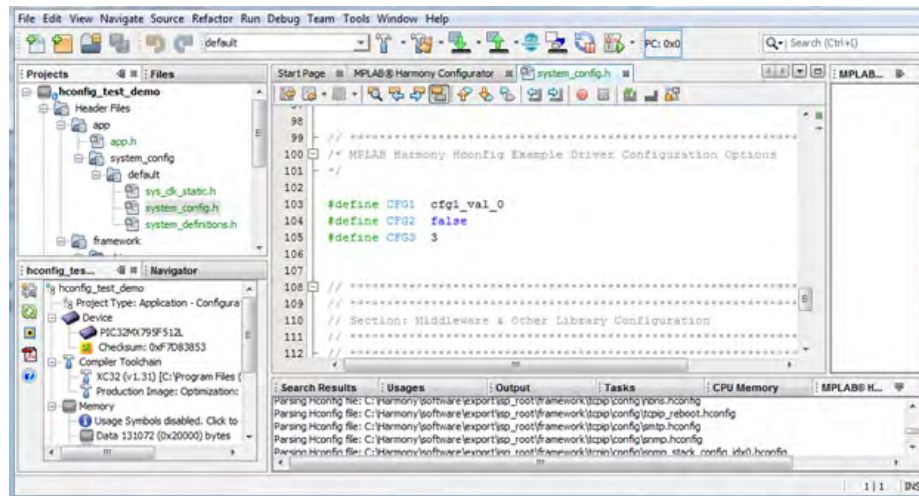
This top-level template simply includes all of the module-specific templates that contribute to the `system_config.h` file. The included files are logically organized within the top-level template. For the following example, we will add our template in the driver configuration section.

When we generate the code, we see our config options are now in `system_config.h`.

Code generation for the rest of the system files follows the same process. A library typically needs to insert code into the following template system configuration template files:

`system_config.h.ftl`	Configuration item definitions
`system_definitions.h.ftl`	Configuration data types, object handles, and include statements
`system_init.c.ftl`	Init data structure definition and call to initialize function
`system_interrupt.c.ftl`	Raw ISR and call to tasks function, if interrupt driven
`system_tasks.c.ftl`	Call to tasks function, if polled

It will be necessary to modify each of the above templates to include the module-specific templates for any new libraries. It is also necessary to carefully review each top-level template to determine the appropriate location at which to include the module-specific templates and then test the code that is generated to ensure that it does not contain any FreeMarker engine error messages and that it functions as expected.

Installing a New Library into MPLAB Harmony

This topic provides information on inserting a new library into MPLAB Harmony.

Description

Within the MPLAB Harmony installation, you will find a `third_party` top-level folder, as shown in the follow figure. Within that folder, third-party code is organized by its purpose. If an appropriate sub-folder exists, create a directory named for your company or your product within that folder and copy your source installation files and folders into it. Your source tree should include the necessary hconfig and FreeMarker templates (as described previously) and Help content (described in the next section) to support your library in the MHC.
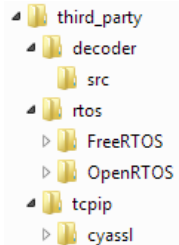
By default, MPLAB Harmony installs into a version-specific folder (`C:\microchip\harmony\<version>` on a Windows personal computer or `~/microchip/harmony/<version>` on a Mac or Linux computer). Therefore, when you install a newer version of MPLAB Harmony, it is very likely that you will need to reinstall your library. If your library is not part of the MPLAB Harmony installation, providing an installer that automates the process of copying your installation files into the new MPLAB Harmony installation folder and inserting the hconfig, FreeMarker templates, and help files into the new hierarchies will be a necessity.

Inserting New Library Help into the MPLAB Harmony Documentation Index

This topic provides information on inserting Help created for a new library into the existing MPLAB Harmony Help.

Description

The MHC displays Help information for each option when it is selected (i.e., clicked) by the user in the configuration window. To do this, the MHC reads the first word (token of contiguous characters with no whitespace) in the Help (or "---help---") section in the associated hconfig file. This word is assumed to be an index entry in the `install-dir>/doc/html/help_harmony_html_alias.h` header file in the selected MPLAB Harmony installation. If the MHC finds this entry in the alias file, it opens the associated HTML file in the Help window pane. If it does not find this entry in the alias header file, it displays the actual text provided in the Help section of the hconfig file. Therefore, there are two ways to support Help documentation in the MHC.

HTML Browser Used by MHC

This topic provides information on the HTML browser used by the MHC to display Help content.

Description

The HTML browser used by MHC is the GUI widget, HTMLEditorKit, which is provided by Java 7's standard library.

This browser accepts HTML Version 3.2 or older; therefore, any HTML to be added the user must be compatible with this version. Any HTML that is constructed to use features newer than V3.2, may not be rendered as expected. It is important to know that the `<applet>` tag is not supported, but some support is provided for the `<object>` tag.

For more information on HTMLEditorKit, visit the Oracle website: http://docs.oracle.com/javase/7/docs/api/javax/swing/text/html/HTMLEditorKit.html

Help Documentation Methods

This topic provides information on the two methods that can be used to create Help content.

Description

Two methods exist for creating Help content:

• Raw text in the "---help---" section of the configuration entry in the hconfig file, or
• HTML Help, identified by an entry in the MPLAB Harmony Help HTML alias header file

To utilize the first method of providing help content for a library, simply include the appropriate help content for each configuration item in text form in the associated help section for that item in the library's hconfig file.

To utilize the second method, define the appropriate HTML help content in an HTML file. Copy that file into the `<install-dir>/doc/html` folder. Then, append the appropriate Help link (following the conventions described in the following sections) to the end of the HTML alias header file. The order of the entries in the alias header file is not important as it is read, sorted, and searched in it's entirety, by the MHC. However, every alias identifier in the file must be unique, as described in the following section.
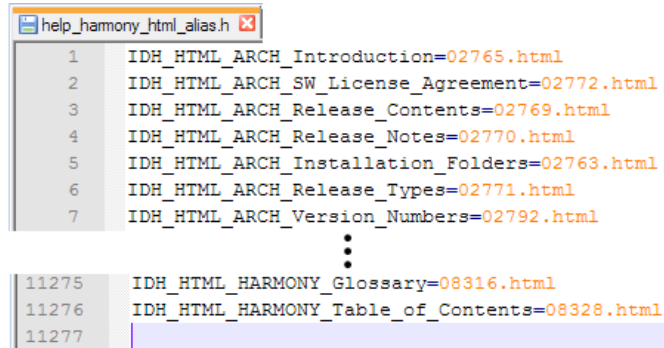
HTML Alias Header File

This topic provides information on the structure and conventions to be followed when adding HTML references to the MHC HTML alias header file.

Description

The HTML alias header file, `help_harmony_html_alias.h`, is located in the following folder within the MPLAB Harmony installation: `<install-dir>/doc/html/`

An example of this file is shown in the following figure:

```
help_harmony_html_alias.h
    1      IDH_HTML_ARCH_Introduction=02765.html
    2      IDH_HTML_ARCH_SW_License_Agreement=02772.html
    3      IDH_HTML_ARCH_Release_Contents=02769.html
    4      IDH_HTML_ARCH_Release_Notes=02770.html
    5      IDH_HTML_ARCH_Installation_Folders=02763.html
    6      IDH_HTML_ARCH_Release_Types=02771.html
    7      IDH_HTML_ARCH_Version_Numbers=02792.html
                               •
                               •
11275      IDH_HTML_HARMONY_Glossary=08316.html
11276      IDH_HTML_HARMONY_Table_of_Contents=08328.html
11277      |
```

To add your own HTML file references to this list, use the following conventions:

`IDH_HTML_`**`<NAME>_<ID>_<TopicTitle>`**`=`**`<NAME>`**`<file>.html`

Where:

- **`<NAME>`** is an abbreviated company name. For example, IBC, which stands for a company named: Itty Bitty Computer
- **`<ID>`** is the tool identifier. For example, GRC for Graphics Resource Converter.
- **`<TopicTitle>`** is a unique topic identifier. For example, Release_Notes.
- **`<file>`** is the file name (after the company name prefix) of the HTML file for the particular topic

For example, to add a new section named New Tool with a title of New Tool Help to the existing HTML Help, the recommended entry in the alias header file would be:

`IDH_HTML_IBC_TOOL_New_Tool=IBC_new_tool_help.html`

Notes:
1. The content of your HTML files must be compatible with HTML Version 3.2 or older. The HTML browser used by MHC cannot process HTML tags that are newer than V3.2.
2. You must ensure that any entries added to the existing alias header file are unique from all other entries.
3. When choosing the TopicTitle, use underscores in place of spaces, hyphens, etc.
4. To avoid conflicts with the HTML file numbering used by the MPLAB Harmony Help, it is suggested to use names such as, `IBC_Release_Notes.html`.
5. Add new entries to the end of the file.
6. The following HTML file names are already used by the MPLAB Harmony Help and cannot be reused:
    - `contents.html`
    - `frames.html`
    - `ftxtsearch.html`
    - `header.html`
    - `idx.html`
    - `index.html`

hconfig Files

This topic provides information and the location of hconfig files.

Description

The hconfig file tree represents a hierarchy of configuration options presented, with associated Help documentation, by the MHC so that the user can select and configure the desired build options.
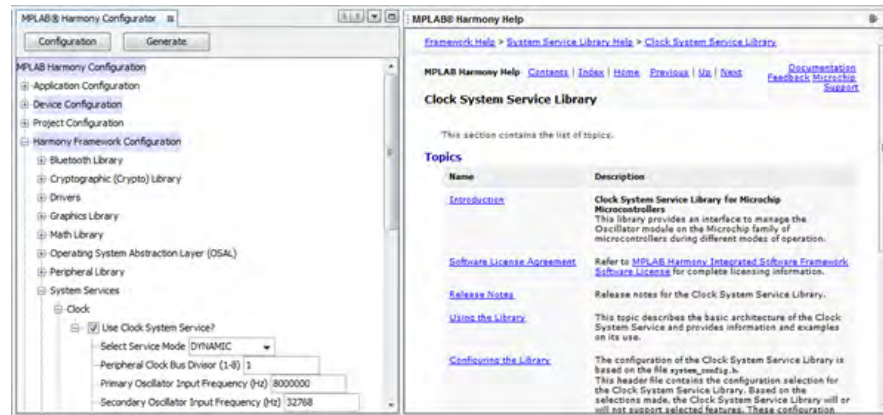
Within the MPLAB Harmony installation, hconfig files are kept in the `config` folder at each level in the installation hierarchy that requires them, with one exception. The root of the hconfig file tree is an application-specific file (`<application-name>.hconfig`) that is generated in the project's `firmware` folder. It is not a predefined file. This generated hconfig root file enables the creation of application-specific options if desired (see **Note**). The root file defines the "MPLAB Harmony configuration" main menu item and then includes (AKA "sources") the installation's top-level hconfig file (`<install-dir>/config/harmony.hconfig`) for the installed libraries and templates (as illustrated in the following figure). The top-level hconfig file then includes (sources) the next level of hconfig files in the hierarchy, each of which includes the next level, and so on, down to the individual library hconfig files, which form the "leaves" of the hconfig tree.
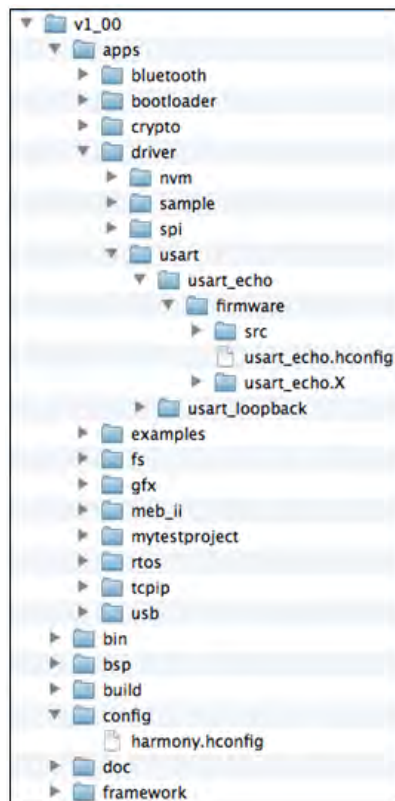
⭐ **Important!**   The MHC does not currently provide a graphical method of creating application-specific configuration options. It is therefore necessary to manually edit the application-specific hconfig file to create application-specific configuration options that will appear in the MHC tree.

Kconfig Language Specification

This topic provides information on obtaining the Kconfig Language Specification.

Description

The MHC hconfig grammar is based on the Linux Kconfig language specification, with a number of MHC-specific extensions. Please reference the following link for documentation of the core Kconfig language specification. The hconfig extensions are documented in the next section.

https://www.kernel.org/doc/Documentation/kbuild/kconfig-language.txt

hconfig Language Extensions (Kconfig+)

This sections provides information on the extensions that have been added to the Kconfig grammar to form the hconfig grammar.

"enum"

This topic describes the "enum" extension.

Description

Syntax: `"enum" <enum set name> [exclusive] <string> [|| <string>]...`

The enum entry specifies a named set of possible input values for string symbols. The enum set name can be used within a string range attribute. The optional 'exclusive' attribute indicates that each config symbol that references the enumeration must use a unique enum string value. Symbols that have been used are grayed out in the combo box drop down list, although they can still be selected. Multiple uses of an exclusive enum value will be flagged as an error.

The keyword "enum" both starts and ends a menu entry.

Example:
```
enum PLIB_MODULE_ID exclusive
"PLIB_ID_0"
|| "PLIB_ID_1"
|| "PLIB_ID_2"
```

"range"

This topic describes the "range" extension.

Description

Syntax: `"range" <enum set name> ["if" <expr>]`

The string range attribute specifies the set of possible values for a string symbol. The user can only input one of the enumerated values of the enum set names. Any default value must be included in these enumerated values.

Example:
```
config PLIB_MODULE
string "PLIB Module"
range PLIB_MODULE_ID
default "PLIB_ID_0"
```

"template"

This topic describes the "template" extension.

Description

Syntax: `"template" <template name> <template file path> to <project logical path> ["if" <expr>]`

The template entry specifies a file to be processed as a FreeMarker template file and copied to a specific location within the project logical path structure.

Example:
```
template SYSTEM_CONFIG_H
"$HARMONY_VERSION_PATH/utilities/mhc/templates/app/system_config/system_config.h.ftl" to
                    "$PROJECT_HEADER_FILES/system_config/$CONFIGURATION/system_config.h"
```

"file"

This topic describes the "file" extension.

Description

Syntax: `"file" <file name> <file path> [to <project logical path>] ["if" <expr>]`

The file entry specifies a file name to be added into the project structure. The path to the file is normally added to the project source search paths. However, if the `[to <project file path>]` is specified, the file is physically copied into the project logical path structure.

Example:
```
file DRV_USART_H "$HARMONY_VERSION_PATH/framework/driver/usart/drv_usart.h" to
                "$PROJECT_HEADER_FILES/framework/driver/usart/drv_usart.h
```

"library"

This topic describes the "library" extension.

Description

Syntax: `"library" <library name> <library file path> ["if" <expr>]`

The library entry specifies a library to be added to the Project linker directives. The path to the library is added to the Project library search paths.

Example:
```
library DEVICE_PERIPHERALS_A "$HARMONY_VERSION_PATH/bin/framework/peripheral/$DEVICE_peripherals.a"
```

"execute"

This topic describes the "execute" extension.

Description

Syntax: `"execute" <exec name> <plugin name> "if" <expr>`

Whenever the "if expression" changes value from *false* to *true*, MHC immediately executes an asynchronous plug-in. The "if expression" must transition from *true* to *false* to *true* again to force another execution of the plug-in.

Example:
```
execute GDDX_PLUGIN GDDX if USER_EXECUTES_GDDX
```

"persistent"

This topic describes the "persistent" extension.

Description

The persistent attribute indicates that the symbol cannot be modified by the user.

Syntax: `"persistent" ["if" <expr>]`

Example:
```
config PERS
bool "Make persistent"
default y

config SOME_INT
int "Enter an int for $PROJECT_NAME in $DEVICE"
default 0
persistent if PERS
```

hconfig Environment Variables

This topic provides information on the hconfig environment variables.

Description

Within the hconfig language, environment variables may be used to reference more global MPLAB X IDE project information. These environment variables, which begin with a "dollar sign" ($), are by convention uppercase, and function much like C preprocessor variables.

The hconfig environment variables include:

Variable Name	Description
$HARMONY_VERSION_PATH	Physical pathname to the MPLAB Harmony directory (i.e., `C:/microchip/harmony/<version>`).
$PROJECT_NAME	MPLAB X IDE main project name when the Generate option was selected.
$PROJECT_FIRMWARE_DIRECTORY	Physical path to the project's `firmware` directory.
$PROJECT_BSP_DIRECTORY	Physical path to the project's `bsp` directory.
$PROJECT_HEADER_FILES	Logical path to the project header files.
$PROJECT_SOURCE_FILES	Logical path to the project source files.
$CONFIGURATION	MPLAB X IDE project configuration name.
$DEVICE	MPLAB X IDE project device name.

$OS_NAME	Name of the operating system on the computer running MPLAB X IDE.

hconfig Configuration Variables

This topic provides information on the hconfig configuration variables.

Description

The hconfig configuration variables include:

Variable Name	Description
DEVICE	Supplies the device variant ID in string format.

Complete hconfig Grammar Definition

This topic provides a complete listing of the hconfig grammar definition.

Description

```
Model:
  ( Statements += Statement)*;

Statement:
    CommonStatement
  | MainmenuStmt
  | MenuStmt
  | ChoiceStmt
;

CommonStatement:
      IfStmt
    | CommentStmt
    | ConfigStmt
    | MenuconfigStmt
    | SourceStmt
    | EnumStmt
    | TemplateStmt
    | FileStmt
    | LibraryStmt
    | ExecuteStmt
    | CompilerStmt
  | AssemblerStmt
;

TemplateStmt:
    'template' name= ID templateFilePath=STRING 'to' templateLogicalPath=STRING ( 'if' (Expr = Expr) )?
;

FileStmt:
    'file' name=ID filePath=STRING ('to' fileLogicalPath=STRING)? ( 'if' (Expr = Expr) )?
;

LibraryStmt:
    'library' name=ID libraryPath=STRING ( 'if' (Expr = Expr) )?
;

ExecuteStmt:
    'execute' name=ID
    (OptionList += Option*)
    // Only valid execute options are Prompt | Dependency | Default | HelpText=KCONFIG_HELP
;

CompilerStmt:
  'compiler' name=ID  which=('C' | 'CPP')?  type=('define' | 'undefine' | 'includepath') str=STRING ( 'if'
(Expr = Expr ) )?
```

```
    ;

    AssemblerStmt:
        'assembler' name=ID  type=('define' | 'undefine' | 'includepath') str=STRING ( 'if' (Expr = Expr ) )?
    ;
    IfStmt:
        'ifblock' ifexpr=Expr
        (statements += Statement*)
        'endif'
    ;

    MainmenuStmt:
        'mainmenu' value=STRING
    ;

    MenuStmt:
        'menu' value=STRING
        (VisibilityList += Visible*)
        (DependsList    += Dependency*)
        (Helptext = KCONFIG_HELP)?
        (MenuBlockList  += Statement*)
        'endmenu'
    ;

    Visible:
        Visible = 'visible' ( 'if' (visible_expr = Expr) )?
    ;

    Dependency:
        'depends on' depexpr = Expr
    ;

    MenuconfigStmt:
        'menuconfig' name= ID
        (OptionList += Option*)
    ;

    CommentStmt:
        'comment' value=STRING
        (DependsList += Dependency*)
        (Helptext = KCONFIG_HELP)?
    ;

    EnumStmt:
        'enum' name=ID (exclusive='exclusive')?
        (Firststring = STRING)
        (Orstrings += Orstring)*
        (Helptext = KCONFIG_HELP)?
    ;

    Orstring:
        '||' value=STRING
    ;

    ChoiceStmt:
        ChoiceStmt = 'choice' (name = ID)?
        (OptionList += ChoiceOption*)
        (Helptext = KCONFIG_HELP)?
        (statements += ConfigStmt*)
        'endchoice'
    ;

    ChoiceOption:
        Optional | Prompt | Dependency | Default
    ;

    Optional:
        Optional='optional'
    ;
```

```
Option:
    Type | Prompt | Range | Dependency | Select | Default | Persistent | MiscOption | HelpText=KCONFIG_HELP
;


SourceStmt:
    'source' path=STRING (numInstances=SIGNED_INT 'instances')?
;


ConfigStmt:
    'config' name= ID
    (OptionList += Option*)
;


Type:
    type=('bool'|'tristate'|'int'|'hex'|'string') tprompt=STRING? ('if' ifexpr=Expr)? |
    type=('def_bool'|'def_tristate') defexpr=Expr ('if' ifexpr=Expr)?
;


Select:
    'select' name=ID ('if' ifexpr = Expr)?
;


Set:
    'set' name=ID 'to' value=Expr ('if' ifexpr = Expr)?
;


Default:
    'default' (value=Expr) ('if' ifexpr = Expr)?
;


Persistent:
    persistent='persistent' ('if' ifexpr = Expr)?
;


Prompt:
    'prompt' value=STRING ('if' ifexpr = Expr)?
;


Range:
    'range' rangeexpr=RangeExpr ('if' ifexpr = Expr)?
;


MiscOption:
    'option' (MiscOption='modules' | MiscOption='allnoconfig_y' | MiscOption='env' '=' string=STRING |
MiscOption='defconfig_list')
;


RangeExpr returns KconfigExpr:
     RangeLiteral ({RangeExpr.left=current} right=RangeLiteral)?
;


RangeLiteral:
    (conf = ID | signed_int=SIGNED_INT | hex=HEX_TERMINAL)
;


Expr returns KconfigExpr:
    OrLiteral ({Expr.left=current} '&&' right=OrLiteral)*
;


OrLiteral returns KconfigExpr:
    EqLiteral ({OrLiteral.left=current} '||' right=EqLiteral)*
;


EqLiteral returns KconfigExpr:
    NeqLiteral ({EqLiteral.left=current} '=' right=NeqLiteral)?
;


NeqLiteral returns KconfigExpr:
```

```
         PrimaryLiteral ({NeqLiteral.left=current} '!=' right=PrimaryLiteral)?
;


PrimaryLiteral returns KconfigExpr:
    ConfigLiteral | NotLiteral | NotExpr | ParenExpr
;


NotExpr:
    '!' '(' NotExpr=Expr ')'
;


ParenExpr:
    '(' ParenExpr=Expr ')'
;


NotLiteral:
    '!' (NotLiteral = ID)
;


ConfigLiteral:
    conf = ID | signed_int = SIGNED_INT | hex = HEX_TERMINAL | string = STRING
;



terminal ID:
    ('1'..'9')('0'..'9')('0'..'9')('0'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    |

     ('1'..'9')('0'..'9')('0'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

    ('0'..'9')('0'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

     ('1'..'9')('0'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
     |

    ('1'..'9')
        (('A'..'Z')|('a'..'z')|'_')
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    |

    (('A'..'Z')|('a'..'z'))
        (('a'..'z')|('0'..'9')|('A'..'Z')|'_')*
    ;

terminal HEX_TERMINAL: '0x'('0'..'9'|'a'..'f'|'A'..'F')*;
terminal KCONFIG_HELP: ('---help---' | 'help') ('\r'? '\n') -> '---endhelp---' ('\r'? '\n');
terminal SL_COMMENT: '#' !('\n'|'\r')* ('\r'? '\n')?;
SIGNED_INT: ('-')? INT;
```

Please refer to Kconfig Language Specification and hconfig Language Extensions (Kconfig+) for semantic descriptions of the hconfig grammatical elements. For usage information, refer to Developing a New hconfig File.

MHC Files

This topic provides an example MHC file.

Description

The MHC stores the user's selections in an MHC file. An MHC file is created for each configuration, named using the configuration name provided

by the MPLAB-IDE, and located (by default) in the configuration-specific `system_config` folder within the `src` folder in the default MPLAB Harmony project.

Default MHC file name and location: `<my_project>/firmware/src/system_config/<my_config>/<my_config>.mhc`

The MHC file is analogous to the `.config` file in a Linux system configuration. It is created and maintained by the MHC and should not be edited by the user. It is parsed when the user clicks **Generate** within the MHC configuration window to provide the data set utilized by the FreeMarker engine when processing the MPLAB Harmony template (`.ftl`) files. This file captures all settings created by user selections in the MHC GUI and can be shared or copied to duplicate a complete set of configuration selections.

MHC prepends `CONFIG_` to each config option, and stores the value in the `.mhc` file. The format is:

`CONFIG_<config option>=<value>`

A common mistake when creating FreeMarker templates is to forget the leading `CONFIG_` when using config values to generate code.

The following example shows `.mhc` file entries for the example drivers that were used in Developing a New hconfig File:

```
93  #
94  # from $HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example.hconfig
95  #
96  CONFIG_USE_HCONFIG_EXAMPLE=y
97  CONFIG_CFG1="cfg1_val_0"
98  CONFIG_CFG2=y
99  CONFIG_CFG3=1
100 CONFIG_CFG4=n
101 CONFIG_CFG6=n
102 CONFIG_CFG7=y
103 CONFIG_CFG8=n
104 CONFIG_CFG9=y
105 CONFIG_CFG10=y
106 CONFIG_DRV_HCONFIG_INSTANCES_NUMBER=3
107 #
108 # from $HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl
109 #
110 CONFIG_DRV_HCONFIG_INST_IDX0=y
111 CONFIG_DRV_HCONFIG_CFG12_IDX0=n
112 CONFIG_DRV_HCONFIG_CFG13_IDX0=42
113 CONFIG_DRV_HCONFIG_INST_IDX1=y
114 CONFIG_DRV_HCONFIG_CFG12_IDX1=y
115 CONFIG_DRV_HCONFIG_CFG13_IDX1=43
116 CONFIG_DRV_HCONFIG_INST_IDX2=y
117 CONFIG_DRV_HCONFIG_CFG12_IDX2=n
118 CONFIG_DRV_HCONFIG_CFG13_IDX2=42
```

📝 **Note:**   The `.mhc` file does not contain config-option definitions for modules that are not selected for use. However, keep in mind that a module may be selected for use by default or as a result of the selection of another module that requires it.

MHC Configuration File

This topic describes the purpose of the `configuration.xml` file.

Description

The file, `configuration.xml`, is used by the MHC to store configuration-specific information. The `configuration.xml` file is created by the MHC for all managed configurations. This file resides in the configuration's `system_config` folder.

The information that this file currently contains includes:

• The configuration's MPLAB Harmony path
• The configuration user preferences
• A list of automatically added files (untracked)
• A list of automatically added templates (tracked)
• A list of automatically added libraries

The tracked attribute means that the generated file is being tracked using checksums.

If this file is not present, the MHC will prompt the user for a MPLAB Harmony path. The file will then be recreated. Upon configuration regeneration, the MHC will compare existing files to the list of generated files. If a name match occurs, the user will be prompted to merge the two files.

⭐ **Important!**   This file is automatically generated by the MHC and should not be manually modified.

BSP XML Specification

This topic describes the format of the `bsp.xml` file, which is required for MHC Board Support Package (BSP) development.

Description

The `bsp.xml` file contains pin information pertinent to an individual Board Support Package or BSP. MHC uses this file to add the appropriate options to the Pin Manager table during configuration. When a BSP is properly organized and presented, MHC will find the appropriate file and dynamically load it when the BSP is selected in the HConfig tree.

This file must reside in the `xml` sub-folder within the desired BSP folder. The XML file must be named `bsp.xml`. An example path for the BSP that supports the PIC32 Bluetooth Audio Development Kit would be: `<install-dir>/bsp/bt_audio_dk/xml/bsp.xml`.

File Example

The following example shows what this `bsp.xml` file might contain:

```
<?xml version="1.0"?>
<bsp name="bt_audio_dk">
    <function name="SWITCH_1" pin="RA0" mode="digital" pullup="true"/>
    <function name="SWITCH_2" pin="RA1" mode="digital" pullup="true"/>
    <function name="SWITCH_3" pin="RA10" mode="digital" drain="true" pullup="true"/>
</bsp>
```

The root node is named 'bsp' and contains a name attribute unique to this package. The root node contains any number of child nodes defining functions that this BSP will add to the Pin Manager table.

The function node must have these required attributes:

• name – a custom name assigned to this function
• pin – the pin name to which this function is attached

The function node may also have these attributes:

• direction – 'in' or 'out', default = 'in'
• latch – 'high' or 'low', default = 'low'
• drain – 'true' or 'false', default = 'false'
• mode – 'digital' or 'analog', default = 'analog
• cn – 'true' or 'false', default = 'false'
• pullup – 'true' or 'false', default = 'false'
• pulldown – 'true' or 'false', default = 'false'

When a BSP is added in HConfig, these defined values will be pushed to the corresponding pin. If it is removed, the pin will return to its default state. The Pin Manager does not prevent the user from changing these values in the Pin Manager after they have been read from the XML file.

📝 **Note:**   To ensure that alterations to `bsp.xml` files are applied, developers must manually clear and reselect the corresponding BSP entry in HConfig. This will notify MHC to reapply the xml values.

Adding New BSPs

This section provides information adding a new BSP.

Updating the BSP hconfig File

This topic provides information on configuring the hconfig file for the purpose of adding a new BSP.

Description

Adding a new Board Support Package (BSP) is a three-step process, which includes:

• Updating `<install-dir>/bsp/config/bsp.hconfig` with the new BSP
• Creating a new `bsp` folder with the necessary BSP files
• Updating `<install-dir>/bsp/config/bsp.config` with the path to the new `bsp.hconfig` file within your new `bsp` directory

Step One: Within the `choice` statement, create and name the `bool` config for the new BSP and specify (*the first three items are required, the fourth is optional*):

• Upon which device family this BSP depends
• The dependency on USE_BSP
• The BSP_TRIGGER selection
• Optionally, which MPLAB Harmony components this BSP should select (i.e., enable)

For example, if a new BSP uses a PIC32MZ EF device, which needs to enable the Graphics Library, the hconfig code may appear like the following:

```
config BSP_MYBOARD
    depends on USE_BSP
    depends on DS60001320      # Microchip document number for devices that can use this BSP
    select BSP_TRIGGER
    select USE_GFX_STACK
```

```
        bool "BSP for my board"  # Ensure you are using the correct quotation marks to prevent errors
```

Step Two: Specify which files should be added to the MPLAB X IDE project when the new BSP is selected, as well as the include path. Outside of the `choice` statement, define a new `ifblock` statement, as follows:

The easiest way to create the files required is to copy an existing `bsp` folder structure from within `<install-dir>/bsp` and edit the files accordingly. There are four files that you will need to edit:

- Files 1 and 2 - `bsp_config.h` and `bsp_sys_init.c` are the files that will be included in your project. These files include the macros and defines for use within your code.
- File 3 - The XML file described below that has your pin descriptions (copy an existing XML for formatting)
- File 4- The `bsp.hconfig` file within your `bsp` directory that contains the following information:
  - Path to XML file containing the pin description for the new BSP (see BSP XML Specification for more information)
  - Path to `bsp_config.h` file
  - Path to `bsp_sys_init.c` file
  - Compiler include path

For example, if the new BSP uses a PIC32MZ EF device that needs to enable the Graphics Library, the hconfig code may appear similar to the following:

```
ifblock BSP_MYBOARD
  file BSP_my_board_xml "$HARMONY_VERSION_PATH/bsp/my_board/xml/bsp.xml" to "$BSP_CONFIGURATION_XML"
  file BSP_ my_board _H "$HARMONY_VERSION_PATH/bsp/my_board/bsp_config.h" to
"$PROJECT_HEADER_FILES/bsp/my_board/bsp_config.h"
  file BSP_my_board_C "$HARMONY_VERSION_PATH/bsp/my_board/bsp_sys_init.c" to
"$PROJECT_SOURCE_FILES/bsp/my_board/bsp_sys_init.c"
  compiler BSP_COMPILER_INCLUDE_my_board includepath "$HARMONY_VERSION_PATH/bsp/my_board "
endif
```

Step Three: Add a pointer to your new BSP in the `<install-dir>/bsp/config/bsp.hconfig` file. Your new line will be at the end of the file and should appear similar to the **bold** line in the following example:

```
...
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+meb2+wvga/config/bsp.hconfig"
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+s1d_pictail+vga/config/bsp.hconfig"
source "$HARMONY_VERSION_PATH/bsp/pic32mz_ef_sk+s1d_pictail+wqvga/config/bsp.hconfig"

source "$HARMONY_VERSION_PATH/bsp/my_board/config/bsp.hconfig"
endmenu
```

MPLAB Harmony Configurator Plug-ins

Describes the clock screen system plug-in interface.

Description

MPLAB Harmony Configurator provides a plug-in interface into the clock screen system.

System Requirements

The system requirements for a MPLAB Harmony Configurator clock screen plug-in are:

- NetBeans v8.0 or later
- Java 7
- MPLAB X IDE v3.06 or later
- MPLAB Harmony Configurator v1.06 or later
- A Java JAR file containing a class that inherits from the abstract class "com.microchip.mplab.modules.mhc.clock.ClockModel".

NetBeans Project Setup

Java Dependencies

Your project will have a dependency on the library `com.microchip.mplab.modules.mhc.jar`. Once MPLAB X IDE and the MPLAB Harmony Configurator have been installed, this file can be found in the following Windows location:
`C:\Users\($YOUR_USER_NAME)\AppData\Roaming\mplab_ide\dev\($MPLABX_VERSION)\modules`.

Please reference the example clock screen plug-in project for more detailed programming interface information.

The project can be found in the MPLAB Harmony framework within the `<install-dir>\utilities\mhc\plugins\clock\`plugin_example folder.

Plug-in Installation:

There are two steps required to install your plug-in into MHC.

1. *Plug-in file*. NetBeans will produce a JAR file of your plug-in. This file must be copied to the MPLAB Harmony framework folder:
   `<install-dir>\utilities\mhc\plugins\clock`.
2. *HConfig* - MPLAB Harmony Configurator relies on the HConfig tree to tell it which clock plug-in file to load. Often this is processor-specific. To get your plug-in loaded, you must edit the MPLAB Harmony framework file: `<install-dir>\framework\config\framework.hconfig`.

- The string symbol `SYS_CLK_MANAGER_PLUGIN_SELECT` must be defined. This symbol value must have the following format: `($JAR_FILE_NAME):($CLOCK_MODEL_CLASS)`, where:
    - `($JAR_FILE_NAME)` - the name of your plug-in JAR file without the `.jar` file extension
    - `($CLOCK_MODEL_CLASS)` - the name of the class in your plug-in that inherits from the ClockModel base class
- For example, to load the MX1 clock screen the symbol would be set to: `mx1:MX1ClockModel`

Once these two items are complete, MHC will attempt to load your clock plug-in at start-up. If loading fails, an exception and stack trace will be printed.

Debugging Plug-ins

MPLAB X IDE Configuration

MPLAB X IDE can be configured to allow NetBeans to debug MPLAB Harmony Configurator plug-ins, as follows:

1. Open the following file in a text editor: `($MPLABX_ INSTALLATION_PATH)/($MPLABX_VERSION)/etc/mplab_ide.conf`.
2. Locate the configuration entry `default_options`. Add the following text to the line (without a line break):

`-J-Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=5858`

This instructs MPLAB X IDE to allow debugging over the socket 5858.

NetBeans Configuration

1. To attach to MPLAB X IDE, right-click the Debug Project drop-down menu and select **Attach Debugger**.
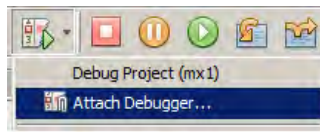
2. Configure the Attach dialog, as shown in the following figure.
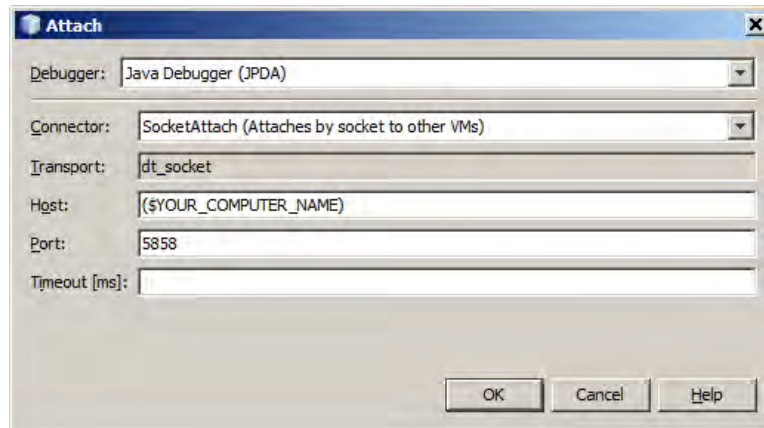
3. If MPLAB X IDE is running and you configured everything properly, the message "User program running" should appear in the lower-left corner of NetBeans.
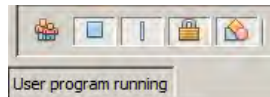
You can now set breakpoints in your plug-in code and debug as normal. If you receive the message *Connection Refused*, this indicates that something has been misconfigured.

Pin Manager Development

Provides details on pin manager development.

Description

The MPLAB Harmony Configurator Pin Manager system is a data driven state machine that provides the capability for users to configure the I/O pins for many different components. It also provides a data-driven mechanism for drawing basic representations of these components.

The following table provides common terms and their descriptions.

Term	Description
Pin Manager	A system for configuring component I/O pins.
Pin Diagram	A visual representation of a component.
Pin Table	A matrix-based system for assigning functions to pins.
Pin	A single I/O interface on a component.

Package	A physical pin layout for a component. Components may come in several packages.
Function	A processing capability that a component supports (e.g., UTX1).
Module	Designates a set of related functions (e.g., UART1).
Component	A discreet part number (e.g., PIC32MX110F016B).
Family	Designates a superset of components (e.g., PIC32MZEF).

File Parsing

The pin manager is responsible for parsing through a set of XML files for the purpose of building a component pin state. These data files are located in the MPLAB Harmony framework within `<install-dir>/utilities/mhc/pin_xml`.

These data files come in four types, Component, Pin, Diagram, and Family:

- **Component** – A unique file for every component supported by MPLAB Harmony Configurator. This data file links the component to a pin file.
- **Pin** – A file that describes the physical characteristics of a component, which includes:
  - Available Packages
  - Pin-to-Package Association – This is needed because a function may not map to the same pin numbers for every package
  - Package-to-Diagram association
  - Supported pin functions – The function groups do not change between packages; however, their associated pin may change
- **Diagram** – A file that describes how to render an image of the selected component package.
- **Family** – Provides several different functions:
  - PPS information (if available, which is taken directly from the product data sheet)
  - Module information:
    - Instructs the pin manager as to how to group available functions in the pin table
    - Provides the capability to specify display constraints, which is what allows the pin table to show UART1 when the UART driver is enabled in the option tree
    - Allows the capability to specify module and function characteristics.

XML File Hierarchy

The following diagram provides a visual illustration of the XML file hierarchy.
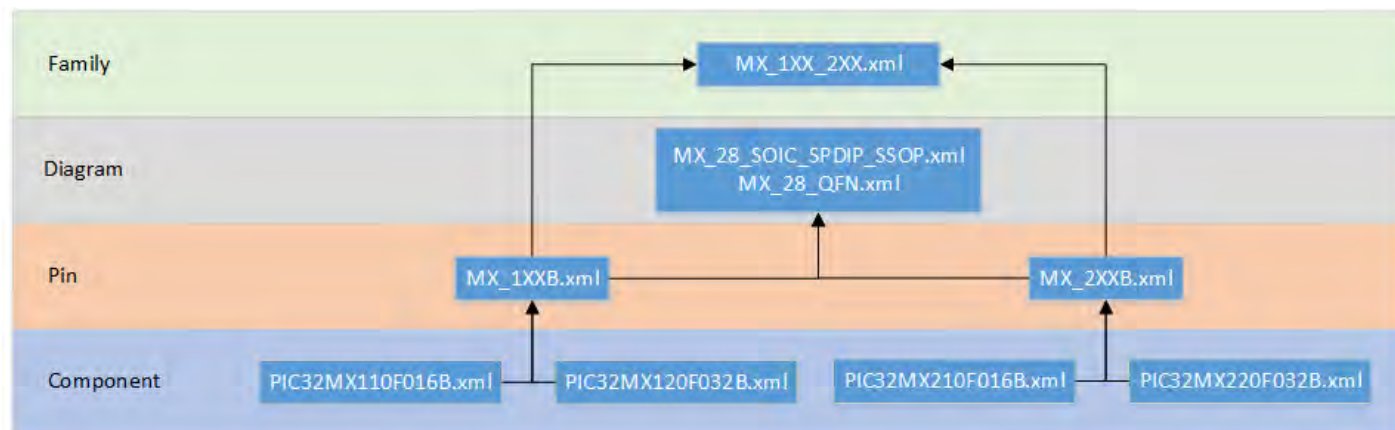
Detailed File Descriptions

Component

The component file only has one entry that maps the selected component to its pin map file.
```
<component device="PIC32MX110F016B" pins="MX_1XXB" />
```

Pin

The `pinfile` root node of the pin file maps the pin file to the family file:
```
<pinfile family="MX_1XX_2XX">
```

The `pinfile` node has two main child nodes: `packages` and `pins`

One or more `package` nodes will be listed inside the `packages` node.
```
<packages>
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="1" name="SOIC" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="2" name="SPDIP" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="3" name="SSOP" />
<package diagram="MX_28_QFN" id="4" name="QFN" />
```

```
</packages>
```
Package Node

The package node description is as follows:

- `diagram` – designates the diagram for a package
- `id` – a unique numerical identifier. This governs the order in which the package appears in the pin table package selector.
- `name` – the name of this package that will be shown in the pin table package selector

One or more `pin` nodes will be listed inside the `pins` node.

```
<pin name="RB5">
<modifiers>
<modifier value="5V" />
</modifiers>
<number package="1" pin="14" />
<number package="2" pin="14" />
<number package="3" pin="14" />
<number package="4" pin="11" />
<function name="PGED3" />
<function name="RPB5" />
<function name="PMD7" />
</pin>
```

Pin Node

The `pin` node description is as follows:

- `modifiers` – The `modifiers` node can have a list of modifier nodes attached to it. **Note:** Currently, only one modifier "5V" is specified. However, this value is no longer used by the pin manager and will be removed in a future version.
- `number` – provides a map between a pin name, a package, and a pin number within that package
- `function` – provides a list of functions supported by this pin

Diagram

A diagram file instructs the pin diagram rendering engine how to draw the particular package for a selected component.

```
<diagram min_x="380" min_y="380" >
<shape type="rect" width="160" height="160" stroke="2"/>
<shape type="string" line="000000" val="$DEVICE_NAME" orientation="right" size="11"/>
<shape type="circle" x="-75" y="-75" radius="5" stroke="1" fill="000000"/>
<layout type="row">
<row pins="1-7" x="-80" margin="5" direction="down"
pin_width="7" pin_height="10" pin_name_location="left" pin_name_size="10" pin_name_margin="10"
pin_number_location="right" pin_number_size="10" pin_number_margin="6" pin_number_orientation="up" />
<row pins="8-14" y="72" margin="5" direction="right"
pin_width="10" pin_height="7" pin_name_location="down" pin_name_size="10" pin_name_margin="10"
pin_number_location="up" pin_number_size="10" pin_number_margin="5" pin_number_orientation="left" />
<row pins="15-21" x="72" margin="5" direction="up"
pin_width="7" pin_height="10" pin_name_location="right" pin_name_size="10" pin_name_margin="10"
pin_number_location="left" pin_number_size="10" pin_number_margin="6" pin_number_orientation="up" />
<row pins="22-28" y="-80" margin="5" direction="left"
pin_width="10" pin_height="7" pin_name_location="up" pin_name_size="10" pin_name_margin="10"
pin_number_location="down" pin_number_size="10" pin_number_margin="5" pin_number_orientation="left" />
</layout>
</diagram>
```

The `root` node is `diagram` and has two attributes: `min_x` and `min_y`. These values describe the overall area of the diagram and are useful for controlling the blank space around the diagram.

Shape Nodes

Shape nodes (`shape`) instruct the rendering engine to draw basic shapes. The shape attributes are dependent on the required `type` attribute. The available `shape` types and their sub-attributes, are as follows:

line – A line:

- `x` – (attribute) the x1 position of the line
- `y` – (attribute) the y1 position of the line
- `x2` – (attribute) the x2 position of the line
- `y2` – (attribute) the y2 position of the line
- `stroke` – (attribute) the width of the line
- `line` – (attribute) the color of the line represented as a hex value RRGGBB

circle – A circle:

- `x` – (attribute) the x position of the circle's radius
- `y` – (attribute) the y position of the circle's radius

- `radius` – (attribute) the radius of the circle in pixels
- `stroke` – (attribute) the width of the circle line
- `line` – (attribute) the color of the circle represented as a hex value RRGGBB
- `fill` – (attribute) the color used to fill in the shape represented as a hex value RRGGBB

rect – A rectangle centered inside the diagram screen:

- `width` – (attribute) the width of the rectangle in pixels
- `height` – (attribute) the height of the rectangle in pixels
- `rounded` – (attribute) a Boolean value to indicate if the rectangle has round corners. Default is "false". Set to "true" to enable.
- `arc` – (attribute) indicates the radius of the rounded corners. Ignored if `rounded` is not "true".
- `stroke` – (attribute) the width of the rectangle lines
- `line` – (attribute) the color of the rectangle border represented as a hex value RRGGBB
- `fill` – (attribute) the color used to fill in the rectangle represented as a hex value RRGGBB

complex_rect – A complex rectangle, centered inside the diagram screen, that can have unique corner descriptions:

- `width` – (attribute) the width of the rectangle in pixels
- `height` – (attribute) the height of the rectangle in pixels
- `corners` – (node) a group node indicating the presence of "corner" attributes
  - `corner` – (node) a node describing a complex rect corner
    - `loc` – (attribute) the corner being described. Must be "topleft", "topright", "bottomleft", or "bottomright"
    - `type` – (attribute) the type of complex corner
      - `notch` – (value) a notched corner
      - `round` – (value) a rounded corner
    - `length` – (attribute) the length of the notch in pixels. Used only if type equals "notch"
    - `arc` – (attribute) the radius of the rounded corner in pixels. Used only if type equals "round"
- `stroke` – (attribute) the width of the rectangle lines
- `line` – (attribute) the color of the rectangle border represented as a hex value RRGGBB
- `fill` – (attribute) the color used to fill in the rectangle represented as a hex value RRGGBB

string – A text string:

- `x` – (attribute) the x position of the string
- `y` – (attribute) the y position of the string
- `val` – (attribute) the value of the string
  - The value `$DEVICE_NAME` is a special keyword that will print the selected component name
- `orientation` – (attribute) controls the direction that the string is printed
  - `up` – (value) print the string rotated counter-clockwise 90 degrees
  - `down` – (value) print the string rotated clockwise 90 degrees
- `size` – (attribute) the font size to use
- `stroke` – (attribute) the width of the text lines
- `line` – (attribute) the color of the text represented as a hex value RRGGBB

string_array – An array of strings drawn on separate lines top-down or bottom-up:

- `vals` – (attribute) a list of strings to print delimited by a comma ",". (e.g., A,B,C,D,E)
- `orientation` - (attribute) controls the direction that the string is printed
  - `up` – (value) print the string rotated counter-clockwise 90 degrees (default)
  - `down` – (value) print the string rotated clockwise 90 degrees
- `size` – (attribute) the font size to use
- `margin` – (attribute) the amount of space to pad between the strings

Layout Node

The `layout` node instructs the rendering engine on how to lay out the pins in the diagram. Pins are typically laid out in rows or grids. When rows are used, sub-segments of pins are assigned to individual rows, and rows are placed as necessary in the grid. The pin diagram will automatically make the cells for each pin interactive when the application is run.

A `layout` node is defined as such with the type attribute being set to either `row` or `grid`:

```
<layout type="row">
</layout>
```

Row Layout

The `row` layout is used to assign pins to individual rows in the diagram. These rows can be placed anywhere but are typically placed on the outline of the shape used to represent the component package. The pin cells in a `row` layout are rectangular.

Row Node

The `row` node provides the capability to specify a pin row. The `row` node has several required attributes:

`pins` – a numerical range specifying what component pins belong to this row. (e.g. "1", "1-7", or "A1-A7")

`margin` – the numerical amount of pixels to pad between each pin cell in this row

`direction` – the direction to draw this row. Valid values are:

- `up` – row is drawn from bottom to top
- `down` – row is drawn from top to bottom
- `left` – row is drawn from right to left
- `right` – row is drawn from left to right

`pin_width` – describes the width of a pin cell

`pin_height` – describes the height of a pin cell

`pin_name_location` – describes which side of the cell in which to draw the pin name text. Valid values are:

- `left` (default)
- `down`
- `right`
- `up`
- `none`

`pin_name_size` – describes the text size of the pin name

`pin_name_margin` – describes the distance to pad the pin name from the pin cell

`pin_number_size` – describes the size of the text used when drawing the pin number

`pin_number_location` – describes the location of the pin number relative to the pin cell

- `left`
- `down`
- `right` (default)
- `up`
- `inside`

`pin_number_orientation` – describes the orientation of the text representing the pin number. Valid values are:

- `left`
- `down`
- `right`
- `up` (default)

Grid Layout

The `grid` layout is used to display a table of pins in a grid-based layout. Pins are laid out in a uniform manner of rows and columns. Pins are displayed as circles instead of rectangles. Pin numbers are contained inside the circle and the pin name is displayed below the circle.

An example of a `grid` layout is as follows:
```
<layout type="grid" pin_margin="40" pin_name_margin="0" pin_rows="18" pin_cols="18" pin_radius="11" />
```

The attributes of a grid layout are as follows:

- `pin_margin` – this describes the spacing of the pin circles
- `pin_name_margin` – this describes the distance between the pin name and the pin circle
- `pin_rows` – the number of pins per row
- `pin_cols` – the number of pins per column
- `pin_radius` – the radius of the pin circles

Family

Family files provide the method by which the connections between the physical pin descriptions (pin files) and the MPLAB Harmony Pin Manager's user interface as well as the HConfig symbol tree.

A family file consists of root "family" node. The two main child nodes of a "family" node are "groups" and "modules".

Groups

A group describes the Peripheral Pin Select (PPS) capabilities of the family. This data is taken directly from the applicable family data sheet's PPS section. The number of XML groups should match the number of PPS groups specified by the data sheet.

Typical PPS descriptions of input and output groups in a product data sheet are shown in the following two figures:

PPS Input Pins

Peripheral Pin	[*pin name*]R Value to RPn Pin Selection
INT4	0000 = RPA0
	0001 = RPB3
T2CK	0010 = RPB4
	0011 = RPB15
	0100 = RPB7
IC4	0101 = RPC7[2]
	0110 = RPC0[1]
	0111 = RPC5[2]
$\overline{SS1}$	1000 = Reserved
	.
	.
REFCLKI	.
	1111 = Reserved

PPS Output Pins

RPn Port Pin	RPnR Value to Peripheral Selection
RPA0	0000 = No Connect
RPB3	0001 = U1TX
	0010 = $\overline{U2RTS}$
RPB4	0011 = SS1
RPB15	0100 = Reserved
	0101 = OC1
RPB7	0110 = Reserved
	0111 = C2OUT
RPC7	1000 = Reserved
RPC0	.
	.
RPC5	.
	1111 = Reserved

This data is described in XML format as follows:

```
<group id="1">
<pin name="RPA0" value="0"/>
<pin name="RPB3" value="1"/>
<pin name="RPB4" value="2"/>
<pin name="RPB15" value="3"/>
<pin name="RPB7" value="4"/>
<pin name="RPC7" value="5"/>
<pin name="RPC0" value="6"/>
<pin name="RPC5" value="7"/>
<function name="INT4" direction="in"/>
<function name="T2CK" direction="in"/>
<function name="IC4" direction="in"/>
<function name="SS1 (in)" direction="in"/>
<function name="SS1 (out)" direction="out" value="3"/>
<function name="REFCLKI" direction="in"/>
<function name="U1TX" direction="out" value="1"/>
<function name="U2RTS" direction="out" value="2"/>
<function name="OC1" direction="out" value="5"/>
<function name="C2OUT" direction="out" value="7"/>
</group>
```

id – (attribute) the number of this group. This corresponds to the group id in the data sheet.

pin – (node) describes a pin that is part of this PPS group:

- `value` – (attribute) the register value that is assigned in the input table

function – (node) lists pps functions that can be mapped to the listed pins:

- `name` – (attribute) the name of the function
- `direction` – (attribute) specifies if this function is input or output
- `value` – (attribute) register value for this function (output only)

📝 **Note:** Some pins may have the same name regardless of I/O direction. In this example this case is mitigated by adding a unique prefix (e.g., (in) or (out)). These prefixes may be stripped out during code generation.

Modules

A module allows a mechanism to group functions together under a common name. It also provides the capability to hook into the HConfig symbol tree. This allows the Pin Table to be dynamic and only show modules that have been enabled by the user based on data-defined constraints.

A module contains the superset of all functions for a particular family. It is often the case that a component does not support all of the functionality defined in its respective data sheet. The Pin Manager will discard any functions that are not found in the corresponding pin file. Modules that have no available functions will not be shown in the table.

An example of a module definition is as follows:

```
<module name="UART 1" desc="UART 1\n(USART_ID_1)">
<function name="U1RX">
<constraint type="enable">
<pair key="DRV_USART_USE_RX_PIN_IDX[0-5]" value="USART_ID_1"/>
</constraint>
</function>
</module>
```

XML Specification Descriptions

Detailed descriptions of the module XML specification are as follows:

`module` – (node):

- `name` – (attribute) a unique name for this module
- `desc` – (attribute) a nicely formatted name. This is what will be shown in the pin table. Line breaks are specified by the string "\n"
- `analog` – (attribute) Boolean value indicating that this module and all of its associated functions are not 5 volt tolerant and that they can be configured as analog-capable. Default is "false".
- `constraint` – (node) indicates that a constraint is placed on this module
  - `type` – (attribute) specifies the type of constraint
  - `enable` – (value) hides this module if the required constraints are not met. Multiple enable constraints may be used in conjunction:
    - `pair` – (node) a key-value pair:
      - `key` – (attribute) the HConfig symbol to test. In the event that multiple symbols in a particular numerical sequence need to be tested a range can be specified. For example, a module can be dependent on the Hconfig symbols `DRV_USART_USE_RX_PIN_IDX` indices 0 through 5. This can be quickly specified as: `DRV_USART_USE_RX_PIN_IDX[0-5]`
      - `value` – (attribute) a string to test the HConfig symbol against. In the case of a Boolean the value to use is either "y" or "n"

`function` – (node):

- `name` – (attribute) the unique name of this function
- `analog` – (attribute) Boolean value indicating that this function is not 5 volt tolerant and that it can be configured as analog-capable. Default is "false".
- `constraint` – (node) indicates that a constraint is placed on this function
  - `type` – (attribute) specifies the type of constraint:
    - `enable` – (value) hides this function if the required constraints are not met. Multiple enable constraints may be used in conjunction:
      - `pair` – (node) a key-value pair:
        - `key` – (attribute) the HConfig symbol to test. In the event that multiple symbols in a particular numerical sequence need to be tested a range can be specified. For example, a module can be dependent on the Hconfig symbols `DRV_USART_USE_RX_PIN_IDX` indices 0 through 5. This can be quickly specified as: `DRV_USART_USE_RX_PIN_IDX[0-5]`
        - `value` – (attribute) a string to test the HConfig symbol against. In the case of a Boolean, the value to use is either "y" or "n"
    - `debug` – (value) indicates that this function is a debug function. This places special modifiers on the function and it cannot be selected in the pin table.

Note: A special module is defined for use with Board Support Packages. It must have the name "BSP" to be properly identified by the pin manager. This is module is added to when a BSP is selected in the HConfig option tree.

An example of a BSP module is as follows:

```
<module name="BSP" desc="Board Support Package">
<constraint type="enable">
<pair key="USE_BSP" value="y"/>
</constraint>
</module>
```

MPLAB Harmony Display Manager User's Guide

Provides information on the MPLAB Harmony Display Manager plug-in.

Introduction

This section provides a guided user experience with a step-by-step procedure that can be used to configure the MPLAB Harmony framework and the MPLAB Harmony Display Manager plug-in tool to prototype a new display. The hardware platform used as an example will be the PIC32MZ EC Starter Kit plus the Multimedia Expansion Board II (MEB II).

For hardware support to connect your own display to the MEB II, please contact your local Microchip sales office.

Configuring a New Display

Provides the steps to create a New MPLAB Harmony project for the purpose of rendering a test graphics screen on the display.

Description

Use the follow process to create a new MPLAB Harmony project to render a test graphics screen on the display:

1. Create a new MPLAB Harmony project using the instructions provided in MPLAB Harmony Configurator User's Guide > Using MHC to Create a New Application > Step 1: Create the New Project and select the PIC32MZ2048EHM144 as the device.
2. The MPLAB Harmony Configurator (MHC) will be launched automatically.
3. Perform the following configuration changes in the MHC Tree:
   * BSP Configuration > select **Use BSP**
   * Selecting BSP to Use:
     * PIC32MZ EC Starter Kit w/Multimedia Expansion Board (MEB) II
     * Graphics Library > Use Harmony Graphics Library > set to **Enable**
4. Launch the MPLAB Harmony Display Manager (MHDM) using the plug-in drop-down menu.
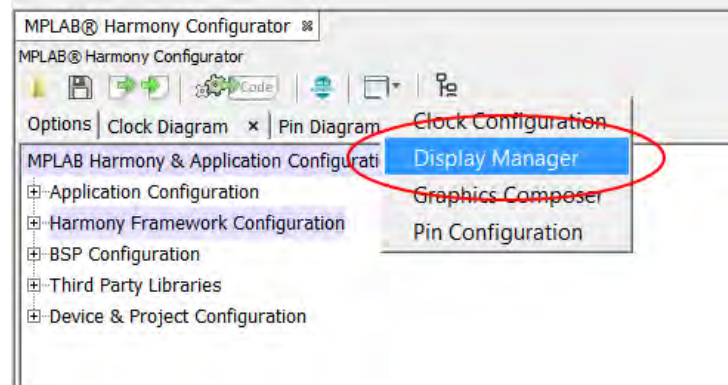
5. The Display Manager will launch and bring its tabs into focus.
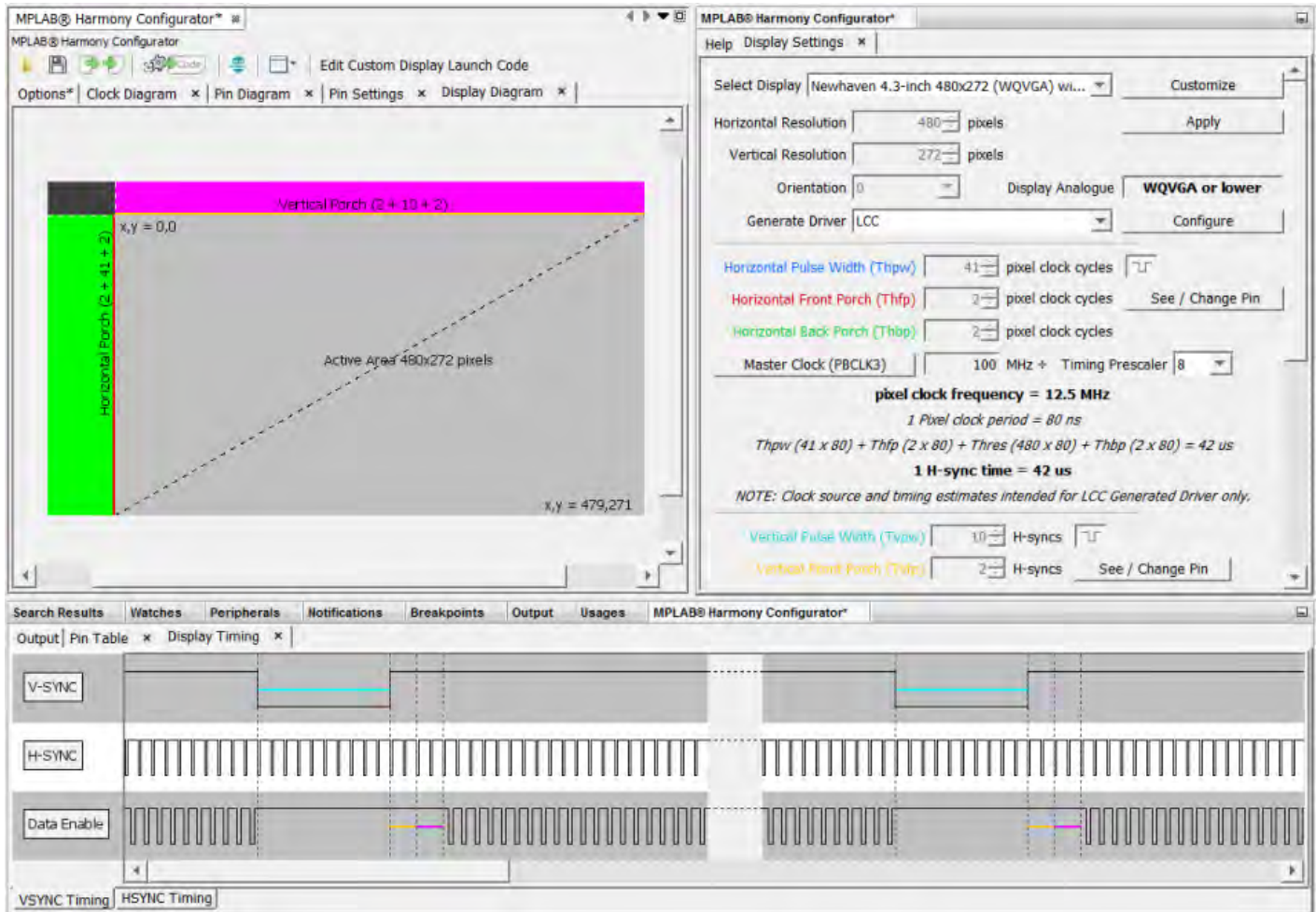
6. Based on the settings of the BSP, the Display Manager will default to the Newhaven 4.3-inch 480x272 WQVGA display. Select **Customize** in the Display Settings tab to enable the fields for entering custom display timing values.

📝 **Note:**   If you choose to use the Newhaven display. certain specification values from the manufacturer's data sheet are required during the configuration process in the MPLAB Harmony Display Manager. A PDF of this data sheet can be obtained from Newhaven Display International, Inc. at: http://www.newhavendisplay.com/specs/NHD-4.3-480272EF-ATXL-CTP.pdf.
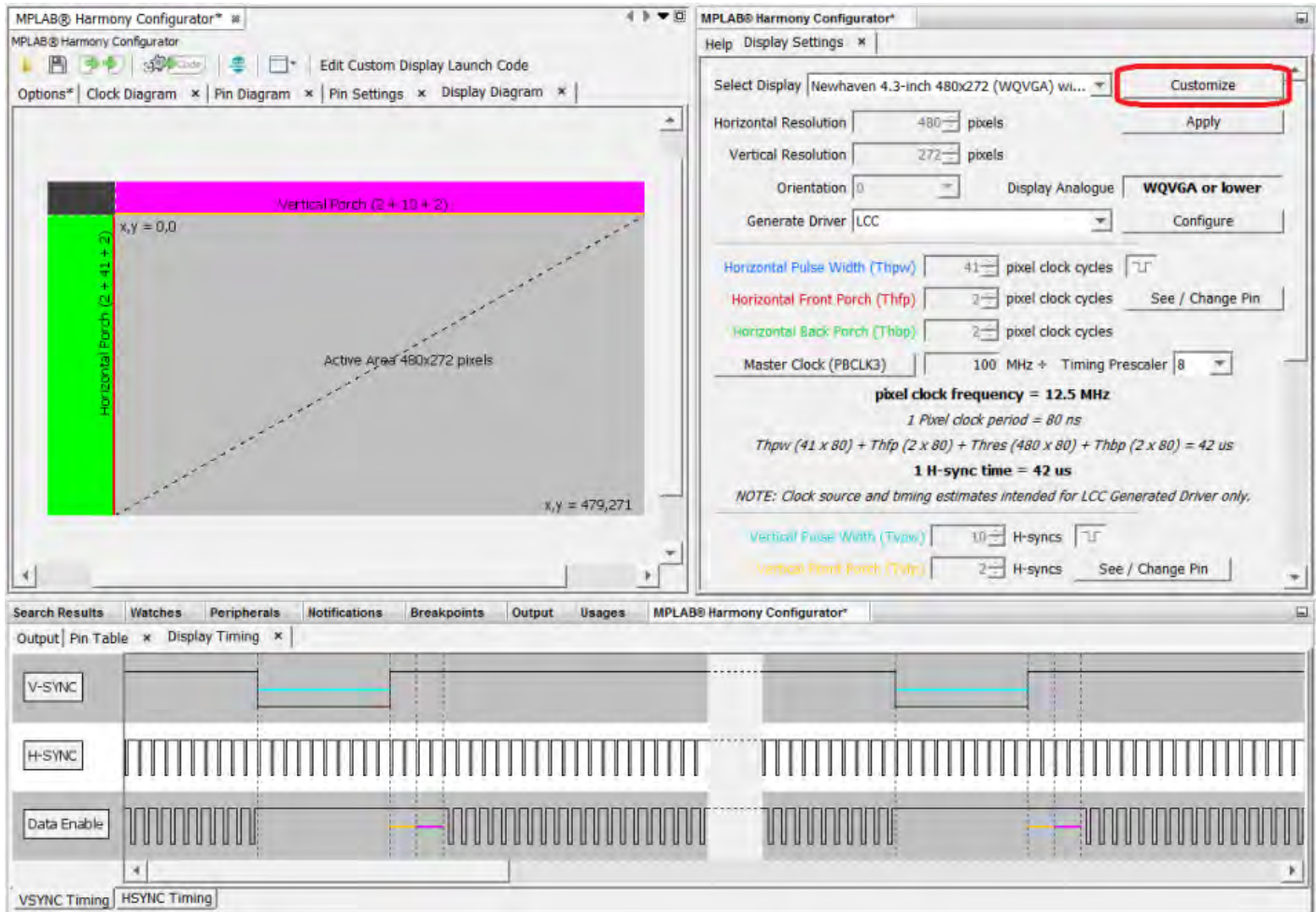
7. In the display data sheet, locate the following values and enter them into their respective fields in the Display Settings tab:
    - Horizontal Pixel Resolution
    - Vertical Pixel Resolution
    - Horizontal Pulse Width (Typically listed as T_{hpw} in the data sheet)
    - Horizontal Front Porch (Typically listed as T_{hfp} in the data sheet)
    - Horizontal Back Porch (Typically listed as T_{hbp} in the data sheet)
    - Vertical Pulse Width (Typically listed as T_{vpw} in the data sheet)
    - Vertical Front Porch (Typically listed as T_{vfp} in the data sheet)
    - Vertical Back Porch (Typically listed as T_{vbp} in the data sheet)

8. The display data sheet may include a diagram depicting the active area transposed over the hidden area. The Display Diagram tab is intended to simulate this diagram based on the values entered in step 7. You may want to visually compare the two.
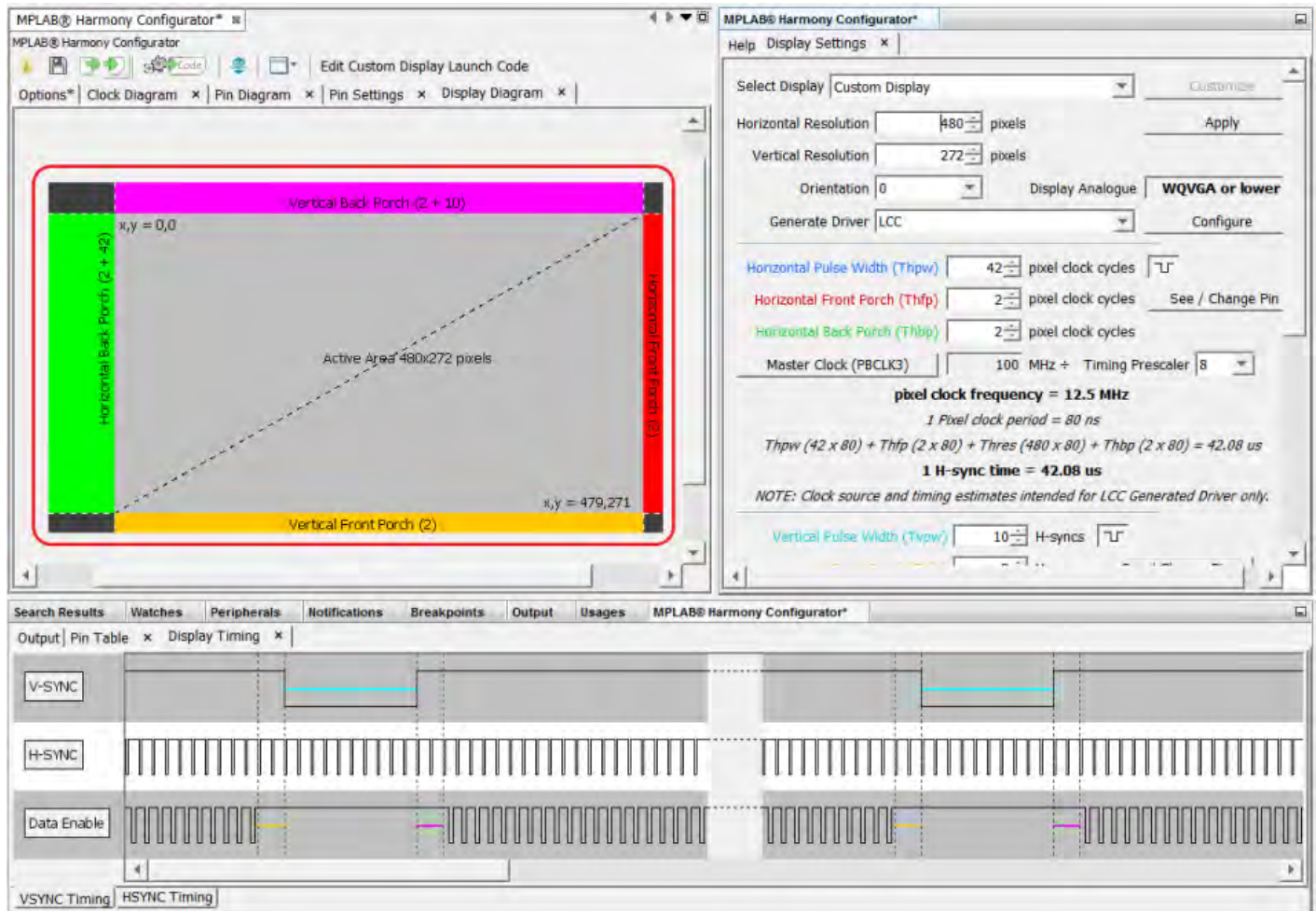
9. Typically, most display data sheets include waveform diagram depicting the timing interaction between the pixel clock (P-clock) signal, Vertical Sync (V-sync) signal, the Horizontal Sync (H-Sync) signal, and the Data Enable (DE) signal. Some displays may not require a Data Enable signal. The Display Timing tab shows a timing simulation of how the graphics controller will behave. You may want to compare the simulation with the waveform diagram in the display data sheet.
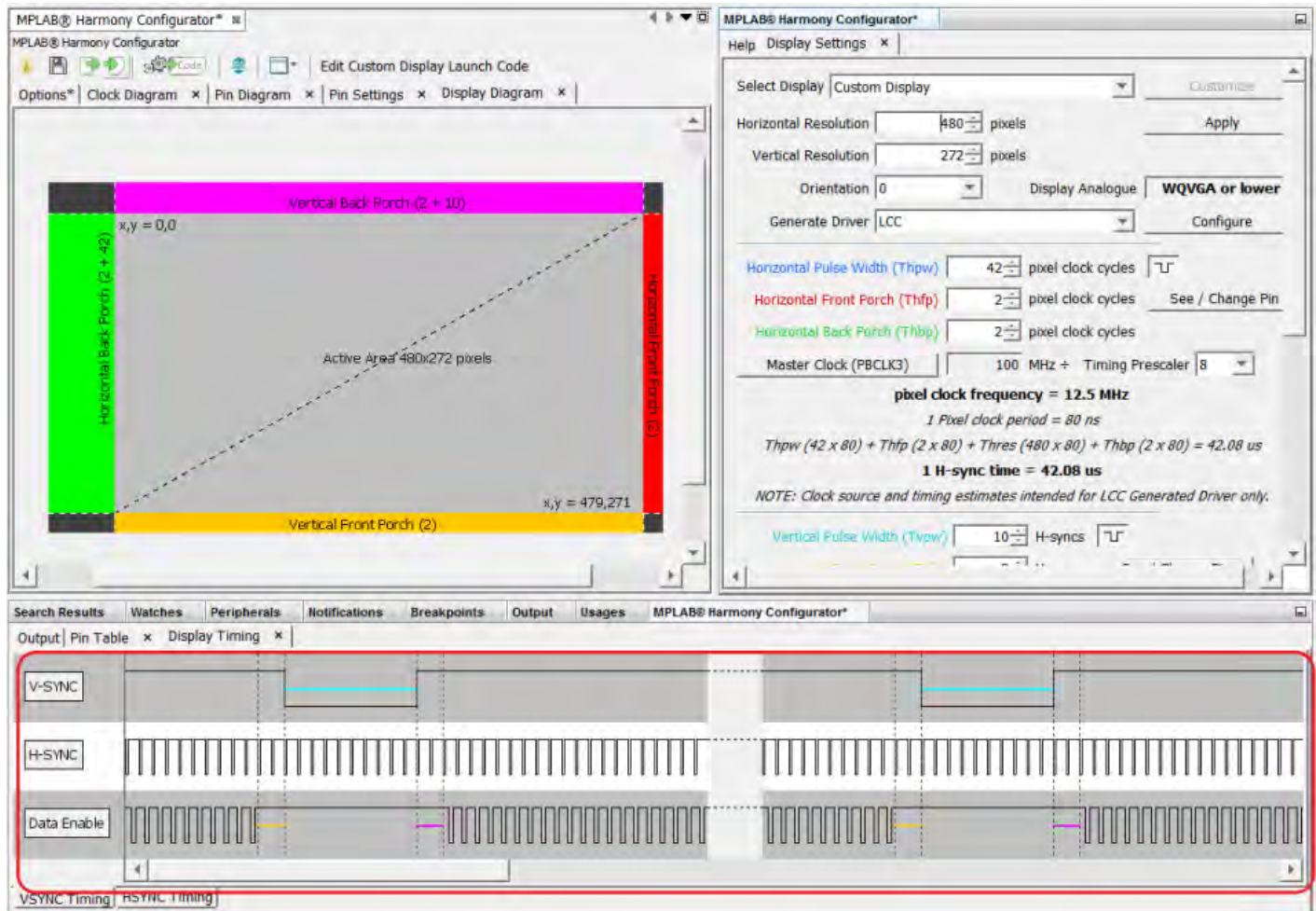
10. Based on the total number of pixels, the Display Manager will estimate a rough analogy of standard display resolution. The estimate is shown in the Display Analogue field in the Display Settings tab. If the display resolution is estimated by the Display Manager to be greater than the largest resolution supported, this field will show "Not Supported".

11. Since we are using the MEB II development board, the Low-Cost Controllerless (LCC) Graphics Controller will be used. The Display Manager has an option to provide a generated driver custom tailored for your display. The MEB II BSP should have been preselected to generate LCC . If not, select **LCC** from the Generate Driver drop-down in the Display Settings tab. This will expose more configuration features within the Display Manager specific to the LCC Controller Driver. For more information regarding the LCC technology, please refer to application note AN1387 *"Using PIC32 MCUs to Develop Low-Cost Controllerless (LCC) Graphics Solutions"*, which is available for download from the Microchip web site (www.microchip.com).

📝 **Note:**   The timing values entered in Step 7 still applies to the graphics controller even if you are not using the LCC Controller Driver.

12. Select **Configure** in the Display Settings tab to open the LCC Configuration user interface.
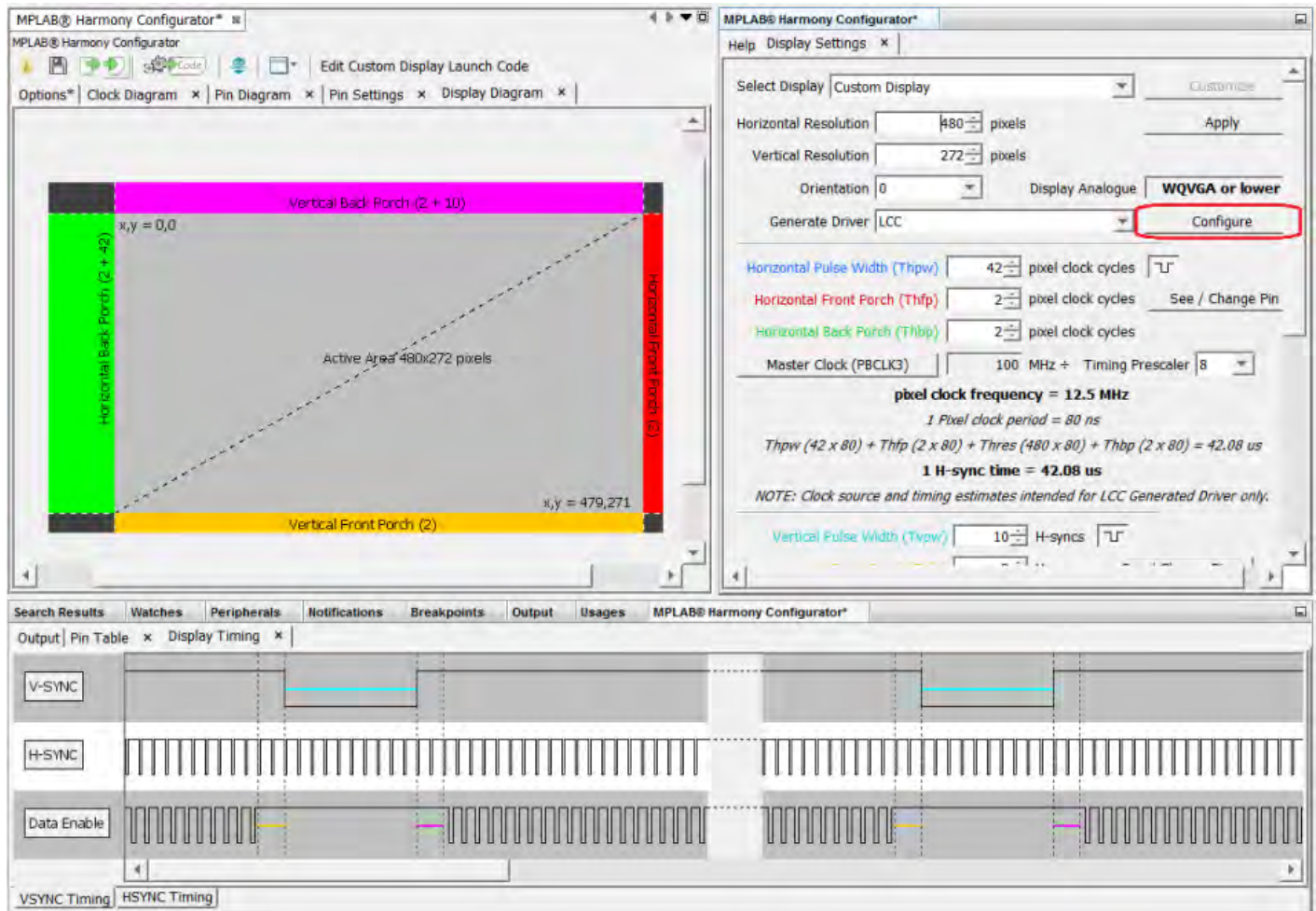
13. The LCC Driver Configuration Settings user interface contains several key settings for quickly setting up the LCC Controller Driver. For now, we will select **Conventional** under Refresh Strategy, as this tells the LCC Controller Driver to use the refresh algorithm most likely to allow the display to render.
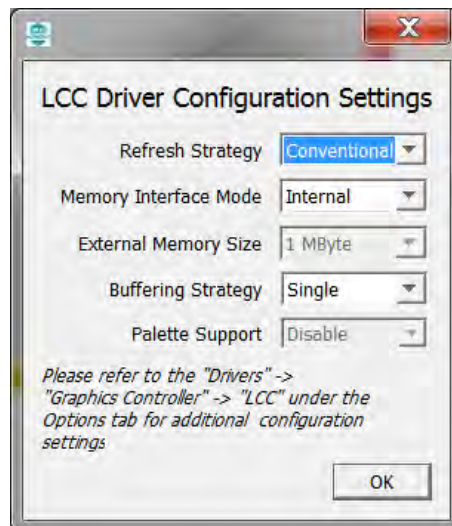
📝 **Note:**   The LCC Controller Driver contains more configuration settings under the MPLAB Harmony & Application Configuration tree in the Options tab of the MPLAB Harmony Configurator (MHC). The LCC Driver Configuration Settings user interface services the essential configuration settings. For further system fine-tuning, you may wish to inspect the configuration settings within the Options tab of the MHC.
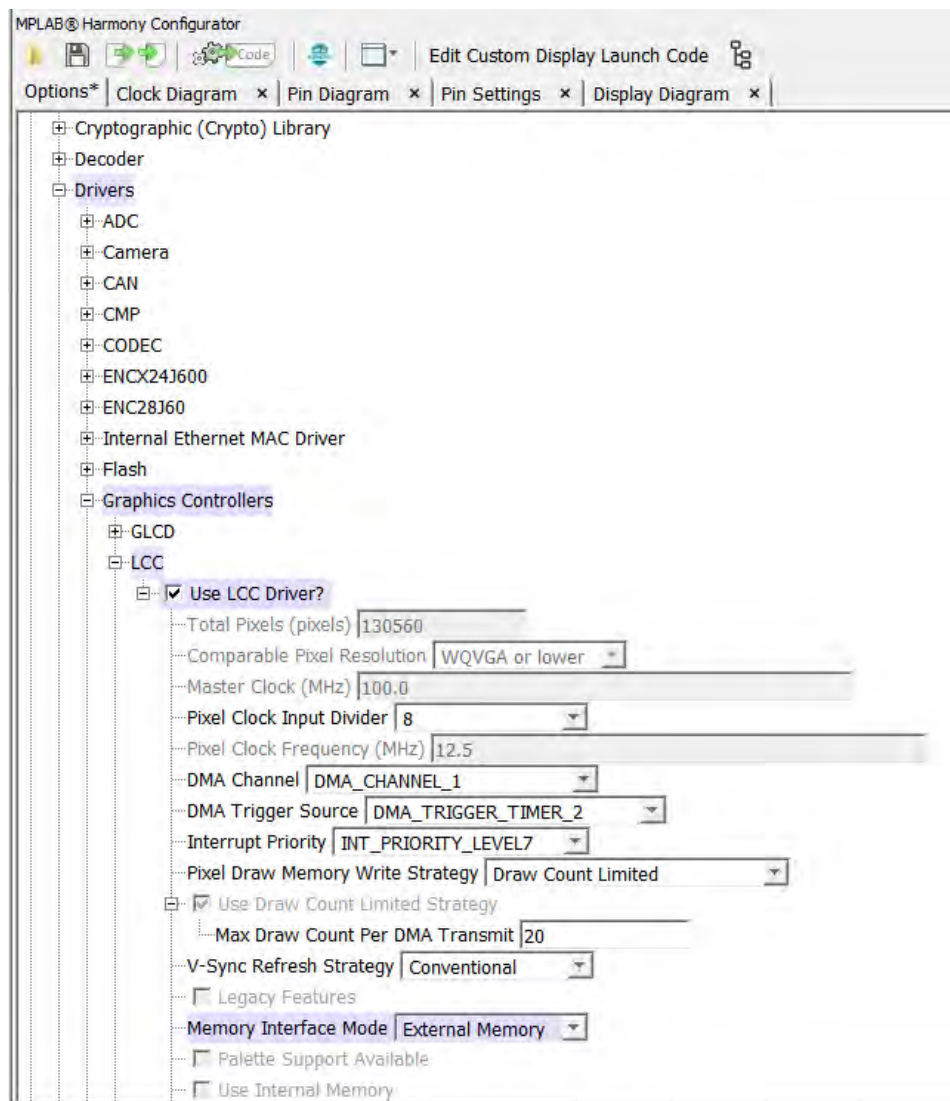
14. The next field to consider is the Memory Interface Mode field. This field directs the LCC Controller Driver, for its display frame buffer, whether to use the PIC32's internal memory or an external memory via the External Bus Interface (EBI).
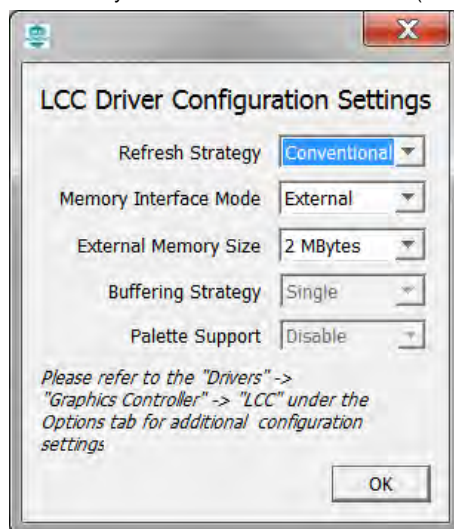
Depending on the display size, you may need to utilize the 2 MB SRAM on the MEB II to accommodate the necessary frame buffer (and in the case of double buffering, two times that amount). The following table can be used to guide you with the selection. Please note that information is provided based on the assumption of 512K memory being internally available on the PIC32MZ device.

Display Analogue	Buffering Strategy	Memory Interface Mode
WQVGA or lower	Single	Internal
WQVGA or lower	Double	External
HVGA	Single	Internal
HVGA	Double	External
VGA	Single	External
VGA	Double	External
WVGA	Single	External
WVGA	Double	External

The Display Manager will provide you with a warning pop-up if it detects the current configuration may not have enough memory to support the display resolution.

Another checking mechanism, in the case of frame buffer being too large for the supported memory, is a compile error such, as shown in the following figure.

Note: Make sure to select **2 Mbytes** when using the external memory, as it is the size of the SRAM connected to the EBI on the MEB II.

15. Depending on your display, the polarity of the V-Sync, H-Sync, and DE pulses can be inverted, respectively. You will want to do so to match the simulated waveform with the data sheet waveform.

Note: Polarity control of the pulses is only available with the LCC generated driver for the current release.

16. If the LCC generated driver is used, the Display Manager also provides an estimated refresh rate base on the master clock source and timing values provided. The estimated timing will be updated as the timing values are adjusted. The important number to note is the Display Refresh Rate. You may want to ensure this number is within the tolerance specification in the display data sheet. There are two ways to adjust the clock source:

- The first method is selecting **Master Clock** to open the Clock Configurator. Depending on your PIC32 device, the button will indicate which clock to adjust. In the case of the PIC32MZ, the clock source to adjust is PBCLK3.

- The second method is to adjust the LCC generated driver's internal prescaler setting. Admittedly, this is a coarser adjustment setting

📝 **Note:**   Typical estimated timing tends to be approximate 5 to 10% higher in frequency than the actual measured value, as the pixel clock for the LCC driver is driven by the clocking of each byte on the DMA channel, the DMA channel is preempted every time the Graphics Library is updating the frame buffer in a draw event. This pause is what slows down the entire refresh rate.

17. Next, we will enable the MPLAB Harmony Graphics Composer (MHGC) to create a test screen for the display. For details about the MHGC, please refer to the MPLAB Harmony Graphics Composer User's Guide.
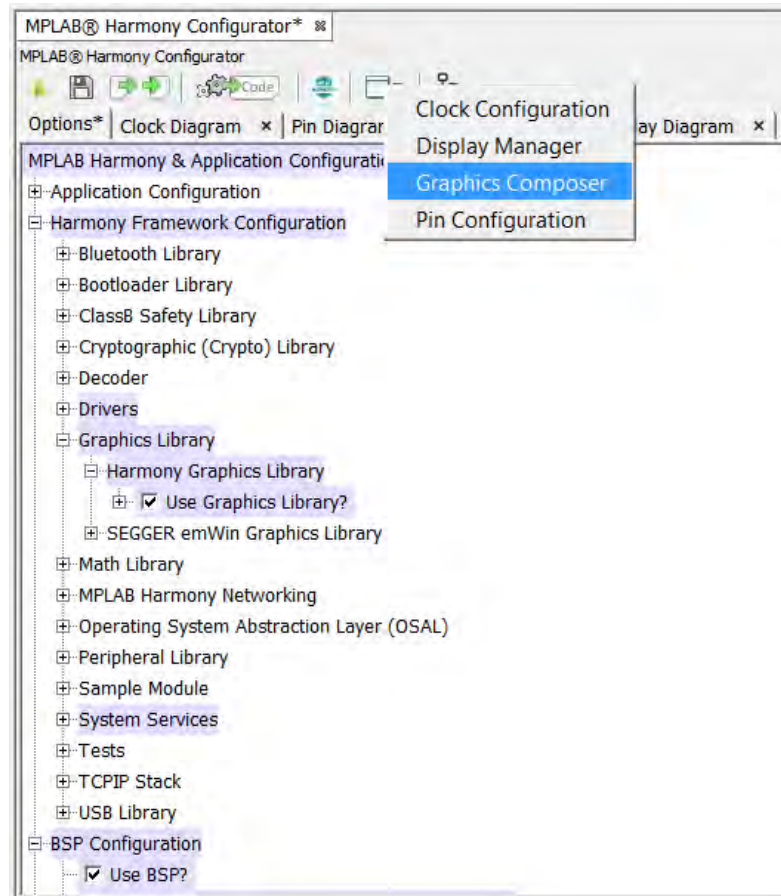
18. Using the MHGC, draw a test screen with a white background, and four line primitives along the border of the screen, as shown in the following figure. You can use the Snap button to get the lines right up against the edge. You may want to add a couple of primitive shapes with fill color so that you can easily see if the display is rendering the graphics properly. The test screen is designed to ensure the timing values entered in step 7 are adequate for the display.
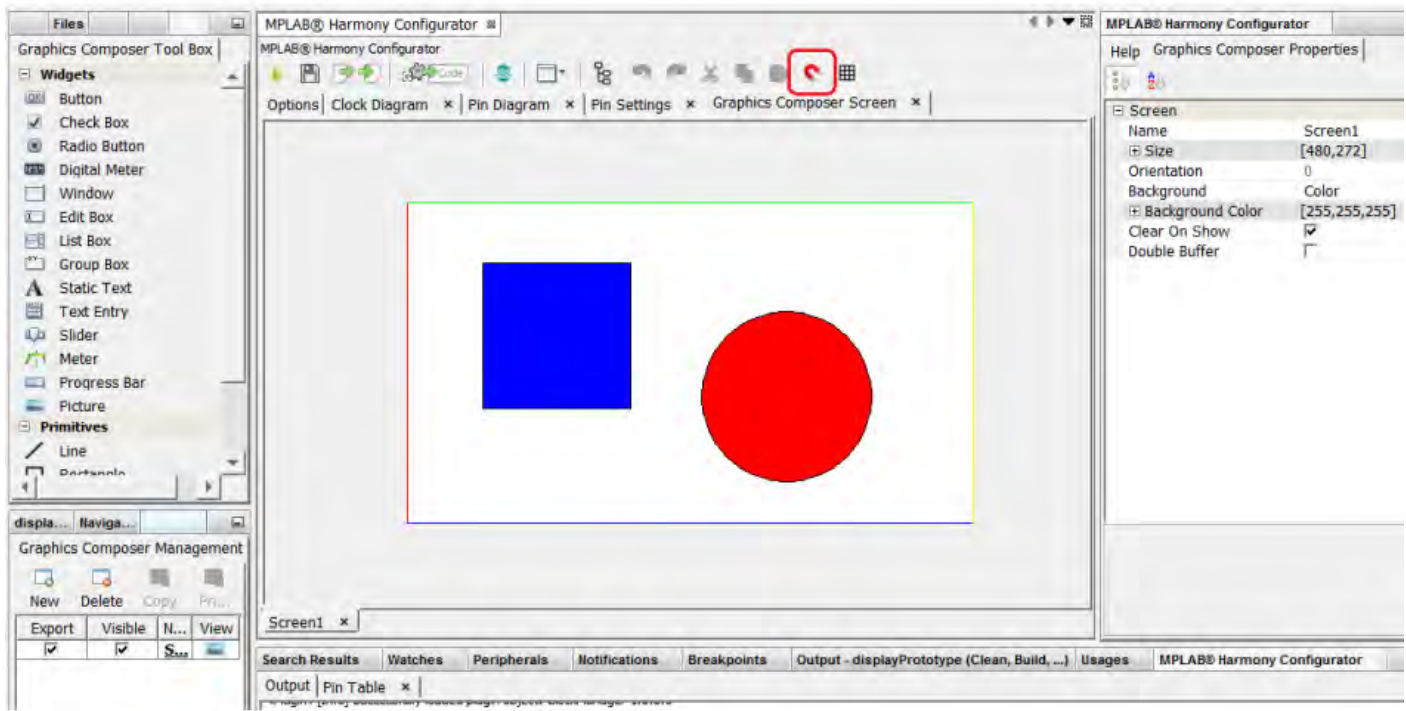
19. The project is now ready to be generated and deployed to the MEB II. During generation, the Display Manager will add the generated LCC driver files `drv_gfx_lcc_generic.h` and `drv_gfx_lcc_generic.c` to `app/system_config/<configuration name>/framework/driver/gfx/controller/lcc`.
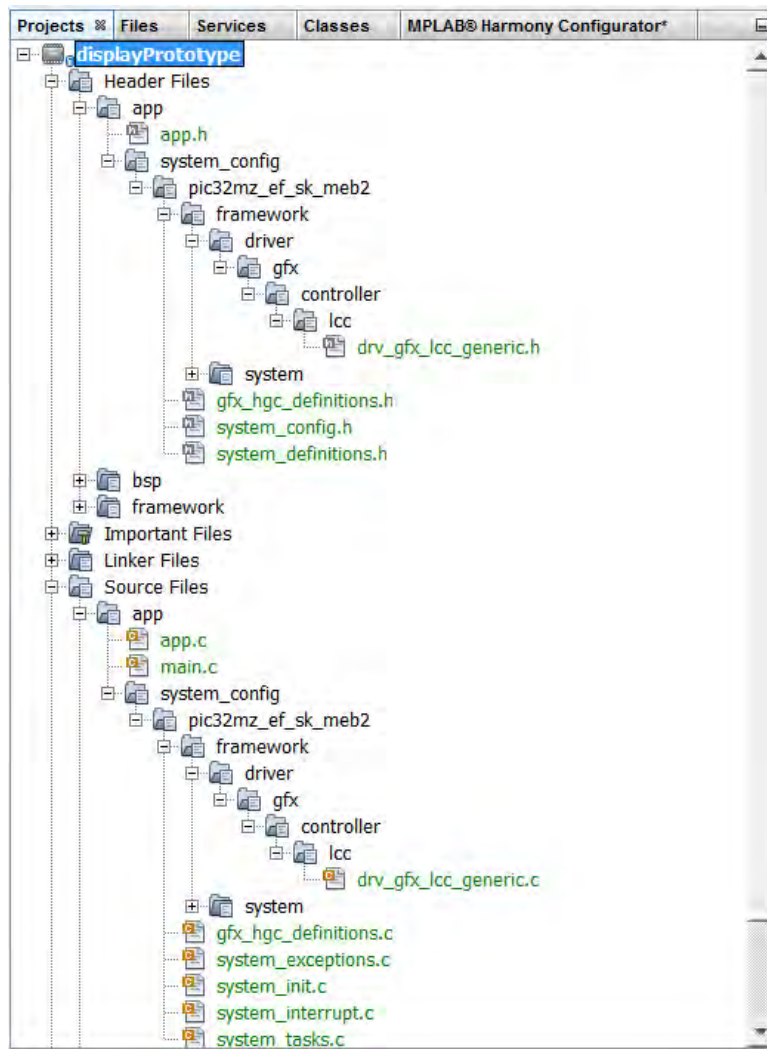
20. Once deployed, you should see the test screen rendered and the single pixel-width color lines in the test screen should be rendered right at the edge of the screen. If you do not see the lines rendered at the edge or they are only partially rendered, you may need to tune the timing values using the Settings Tab. Use the Active Area in the Display Diagram tab as your positional reference.
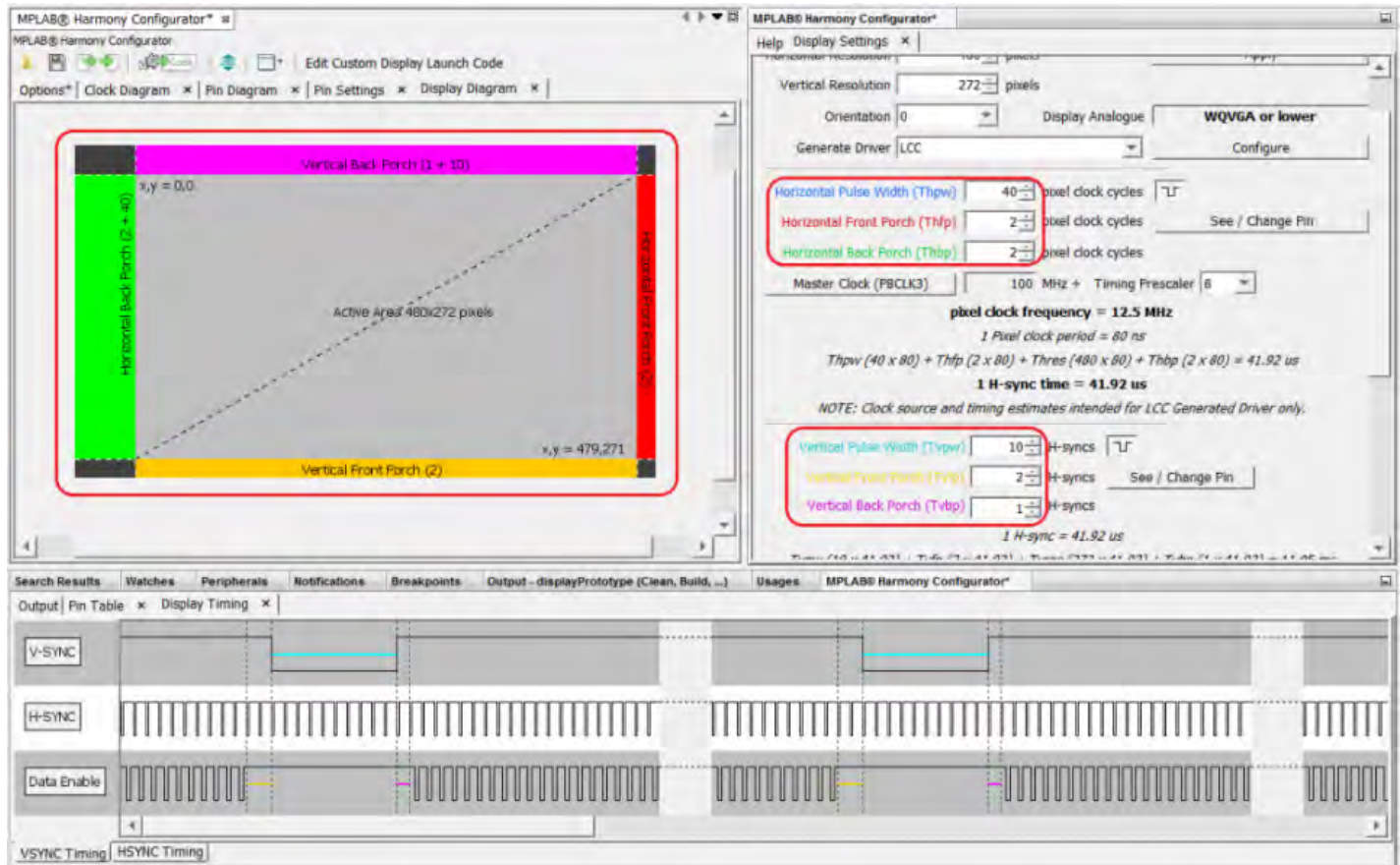
21. You now have a baseline LCC driver configuration for rendering static graphics on your display. However, the driver may not handle image decoding, motion, or screen transition well. Enabling Double Buffering or adjusting the Display Refresh Rate are two ways to improve display driver performance.

The process is now complete; however, you may want to revisit the Display Manager later and use it as a display tuning tool by allowing a short adjustment-to-deploy feedback cycle.

MPLAB Harmony Graphics Composer User's Guide

This section provides user information on using the MPLAB Harmony Graphics Composer.

Introduction

This user's guide provides information on the MPLAB Harmony Graphics Composer (also referred to as the graphics composer), which is included in your installation of MPLAB Harmony.

Description

The MPLAB Harmony Graphics Composer is a graphics user interface design tool that is integrated as part of the MPLAB Harmony Configurator (MHC). This tool allows a user to easily configure and visually design for the MPLAB Harmony Graphics Primitive Library and the MPLAB Harmony Graphics Object Layer.

The overall development flow of Composer consists of:

* Import image and font assets
* Create screens and schemes
* Add objects to screens
* Configure objects
* Generate MHC configuration
* Upload program to device

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.
Object	An abstract term defining an entity that resides in a user interface screen.
Primitive	An object that represents a Graphics Primitive Library object.
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	An object that represents a Graphics Object Library (GOL) widget.

Getting Started

This topic provides information on getting started with the graphics composer.

Description

To begin using the graphics composer, which is part of the MPLAB Harmony Configurator (MHC), you will need to create a new MPLAB Harmony project and select a PIC32 device that is graphics-capable. For example, your project could be named *composer_demo*. Once you've created your project do the following:

1. Open MPLAB Harmony Configurator.
2. In the Harmony Framework Configuration tree expand Graphics Library and select **Use Graphics Library**.
3. Next, click the **Execute** button located next to Create a Design With MPLAB Harmony Graphics Composer.

MPLAB Harmony & Application Configuration
- Application Configuration
- Harmony Framework Configuration
  - Bluetooth Library
  - Bootloader Library
  - Cryptographic (Crypto) Library
  - Drivers
  - Graphics Library
    - ☑ Use Graphics Library? ◄───
      - Create a Design With MPLAB Harmony Graphics Composer (HGC) [Execute] ◄───
      - ☐ Use HGC Design
      - Preemption Level [GFX_NO_PREEMPTION ▾]
      - Color Depth [GFX_COLOR_DEPTH_16 ▾]
      - ☐ Use Fonts?
      - ☐ Use Images?
      - ☐ Use GOL Widgets?
      - ☐ Use Special Effects?
      - ☐ Use Input Devices?
    - Graphics Resource Converter (GRC)
  - Math Library
  - Operating System Abstraction Layer (OSAL)
  - Peripheral Library
  - System Services
  - TCPIP Stack
  - USB Library
  - Sample Module

User Interface

This section describes the layout of the MPLAB Harmony Graphics Composer user interface.

Description

User Interface Layout

The following figure shows the initial user interface layout.
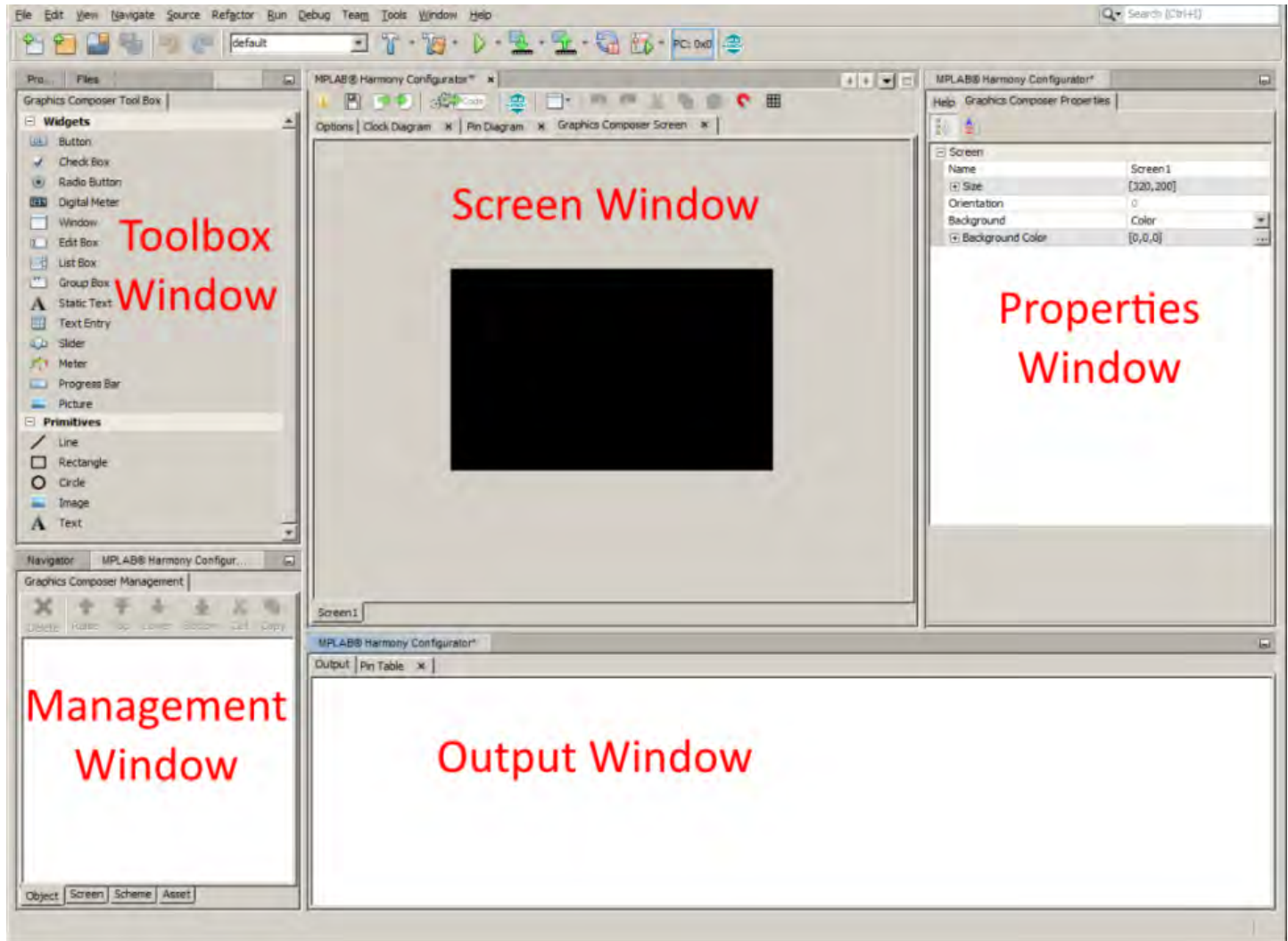
Object Toolbox

The Object Toolbox displays all of the available widgets and primitives to the user.

Composer Management Window

This window allows the user to manage objects, screens, schemes, and assets.

Screen Window

The screen window is the Graphics representation of how objects will appear when displayed on the device.

Properties Window

This window provides the user with the means to adjust properties for objects and screens.

Output Window

This window displays any output generated during your session.

Object Toolbox

Describes the features of the Object Toolbox.

Description

The Object Toolbox is the interface by which users add widgets and primitives into the screen representation. There are two primary methods for creating new objects: clicking and dragging.
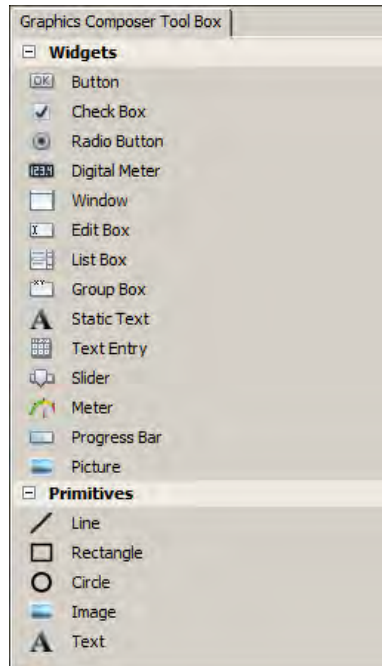
Click Method

The following actions can be performed using the Click method:

- Clicking on 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 will deactivate it

Drag Method

Dragging and dropping a tool item into the Screen Window will also create a new instance of an object. When dragging a tool item, releasing the cursor outside of the Screen Window will cancel the drag operation.

Interactive Object Creation

The Primitives "Line" selection offers an interactive method for creating lines. Activating the Line primitive will open the Line Primitive Create tool. The user will then be prompted to create line points. Lines can be created using two discrete clicks or using a single click and drag operation. When creating the second line point, the <Shift> key can be pressed to lock to the X or Y axis of the first point.

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 only the Graphics Library code necessary for your design is included in the project.

Composer Management

This topic describes the features of the Composer Management window.

Description

The Composer Management window provides four tabs for graphics management.

- Object
- Screen
- Scheme
- Asset

Object Tab

Describes the features of the Object tab.

Description

The Object tab of the Composer Management window provides the capability to delete, select, and manage the placement of objects in the active screen.

The follow actions can be performed in the Object tab:

- Left clicking on an object will select it
- Left clicking no objects will clear a current selection
- Shift-Left Clicking will do a group select. Ctrl-Left click will perform a toggle select
- The Delete button will delete the selected objects
- The Raise, Top, Lower, and Bottom buttons control object placement in the list. Objects are drawn from the bottom up and higher objects will cover lower ones.

Note: The current Primitive Library implementation in the Graphics Library ignores this type of ordering. Currently, primitives are placed above widgets.

Screen Tab

Describes the features of the Screen tab.

Description

The screen management tab in the management window allows the user to create new screens, delete existing screens, and change some screen options.
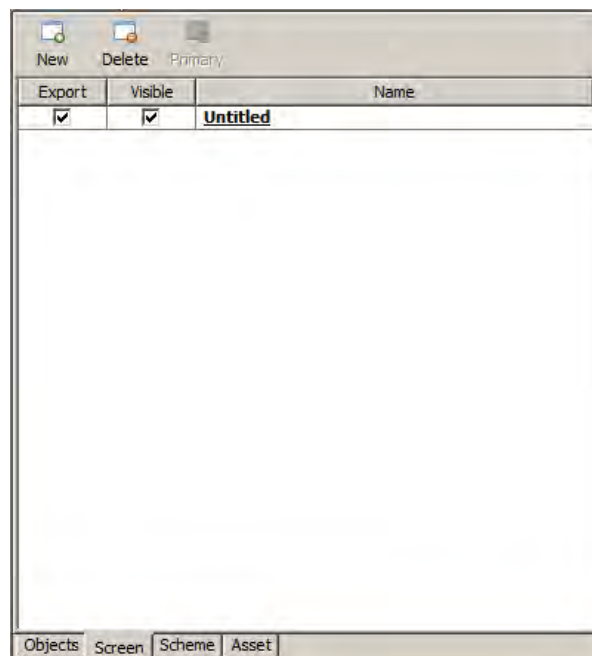
Button Descriptions

The following selections are available:

- New – Creates a new screen. Note that screen names must be unique
- Delete – Deletes the selected screen. Screens can be selected by clicking on their row in the table
- Primary – Designates the selected screen as the primary screen. The primary screen is the screen that will be shown first when the UI is activated.

Table Descriptions

- Export – Controls whether the associated screen is exported when converting the project to code
- Visible – Controls whether the associated screen is visible on the screen window tab bar

Screen Status

Screen names may be in **Bold** type or underlined, which represent different screen states.

- Bold – The screen with the name in **Bold** type is the currently active screen in the screen window
- Underline – The underlined screen is designated as the primary screen

Scheme Tab

Describes the features of the Scheme tab.

Description

The Scheme tab of the management window allows for the management of display schemes.
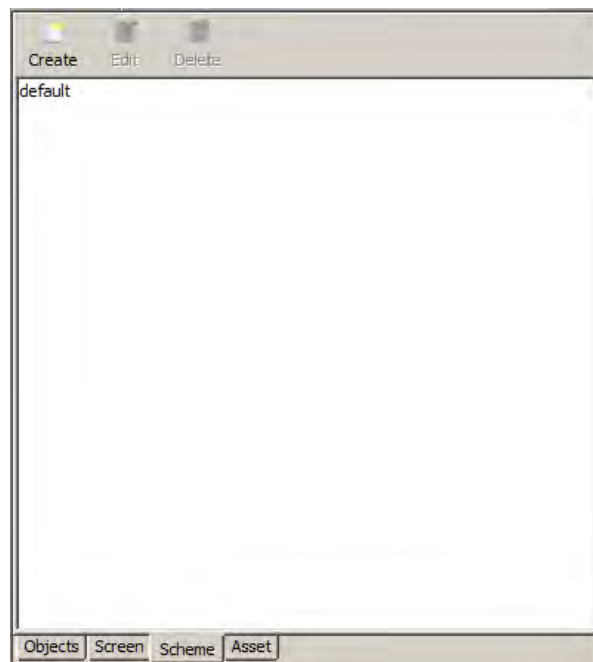
Button descriptions

The following features are available:

- Create – Create a new display scheme. Scheme names must be unique.
- Edit – Edit an existing display scheme.
- Delete – Delete an existing display scheme.

Editing a Scheme

To edit an existing scheme, select the scheme from the list and click **Edit**. The Edit Scheme dialog appears, which allows the user to change various options associated with graphics display schemes.

- Font - This drop-down box allows the user to assign a font to this scheme. The box field is currently blank as no fonts have been imported into the graphics composer.
- Background Offsets - The background is to be offset by the specified X and Y coordinates
- Alpha - Defines the Alpha value
- Colors - Colors may be changed by selecting the corresponding ellipsis button

- Fill Style - Sets the fill style
- Background Type - Sets the background type
- Preview - The Preview window shows how the scheme would appear when applied to a button widget. The color box in the lower right corner of the Preview window allows the user to change the Preview window background.
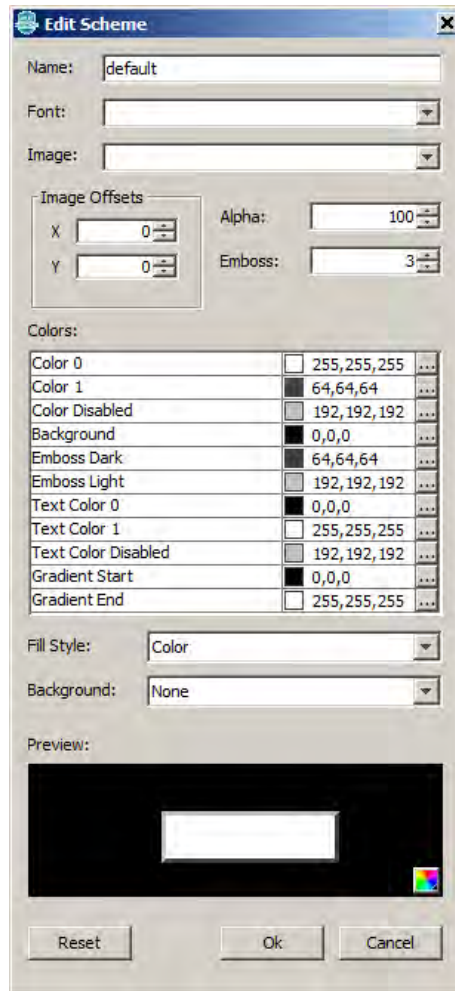
Asset Tab

Describes the features of the Asset tab.

Description

This interface allows the user to import and convert images, fonts, and binary data into assets that the graphics composer will display and output during code generation. Users experienced with the Graphics Resource Converter (GRC) utility will be familiar with these functions.

Imported assets are stored in a binary format file named `asset.cache`. This file resides in `firmware\src\system_config\$CONFIGURATION_NAME`. If this file deleted, all imported assets will be unavailable and must be imported again.

Button Descriptions

> 📝 **Note:**   Beginning in MPLAB Harmony v1.05, the HConfig tree-based Graphics Resource Converter (GRC) interface has been removed, and the Asset Tab is the only available integrated method for importing fonts and images.

The following selections are available:

- Image – Opens the Import Image dialog
- Font File – Opens the Import Font File dialog
- System Font – Opens the Import System Font dialog
- Binary Data – Opens the Import Binary Data dialog

Importing Images

Open the Import Image dialog by clicking **Image**.
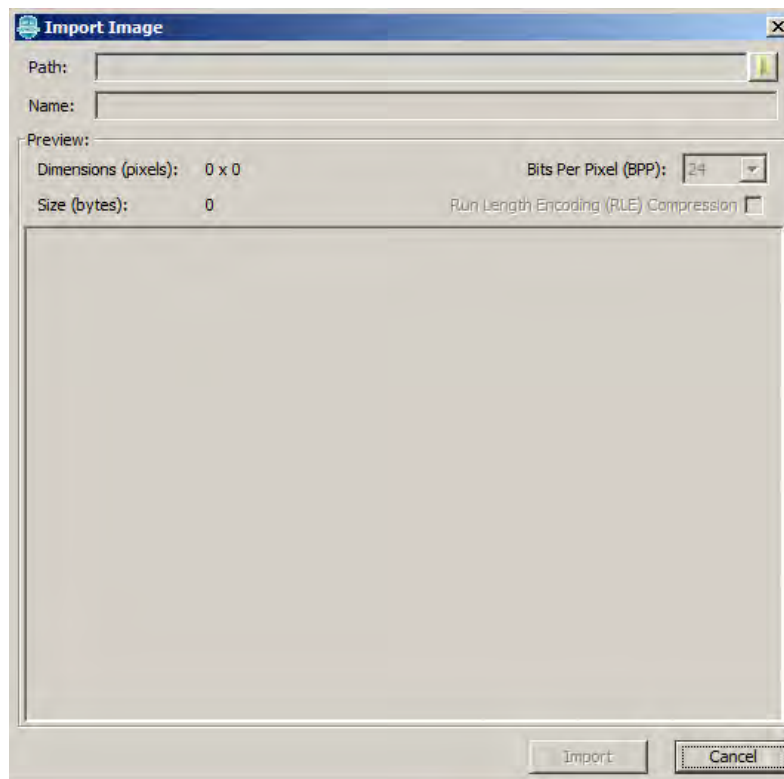
Click the Path: Browse icon and navigate to an image. The graphics composer supports sourcing from all image formats that are natively supported by Java. To be specific, all formats will convert to 16-bpp BMP with the exception of JPEG, which is supported by the JPEG decoder at runtime, and therefore, do not require conversion.

Auto-Configuration

The graphics composer will detect that a JPEG asset has been added and automatically configure the MHC Tree with the JPEG decoder. To inspect or change this in the MHC Tree, see *Harmony Framework Configuration > Graphics Library > Harmony Graphics Library > Use Graphics*

Library? > Use Images? > Enable JPEG Support
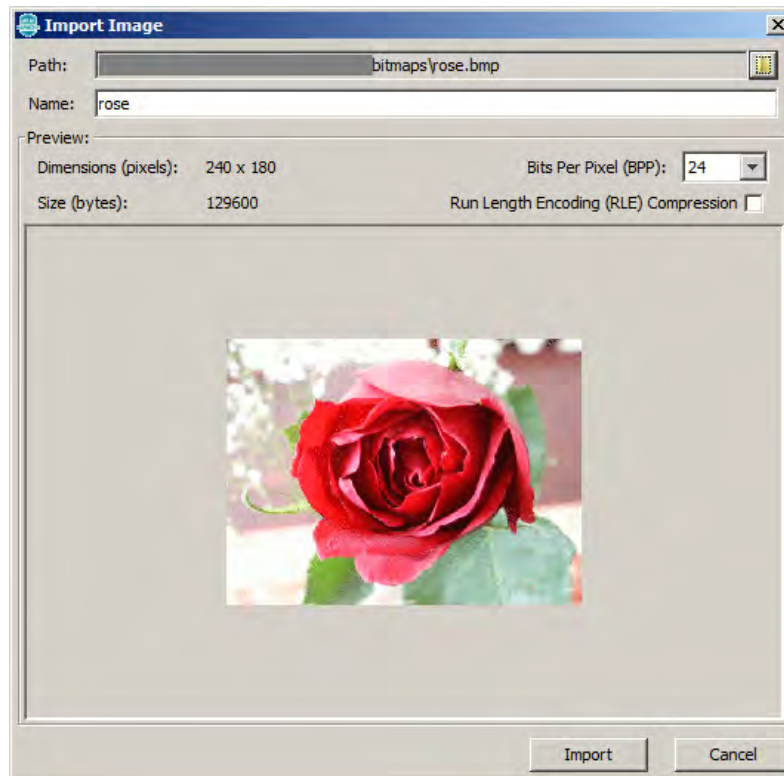
After selecting an image the dialog will display a preview of the image and asset size. The name field shows the asset name and must be unique. The Bits Per Pixel and Compression settings can be changed to see how the image and asset size change.

Note: These values are only for previewing. The current Graphics Library only supports a global BPP setting. The compression setting can be toggled in the asset management table.
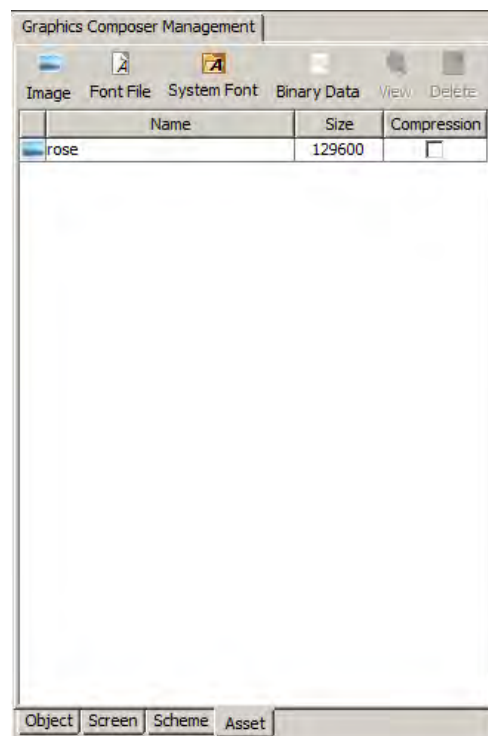
Upon selecting **Import**, the asset table will update to reflect the change. At this point, the asset can be renamed or compression can be enabled.

Selecting the asset and clicking **View** will show the image asset preview dialog.
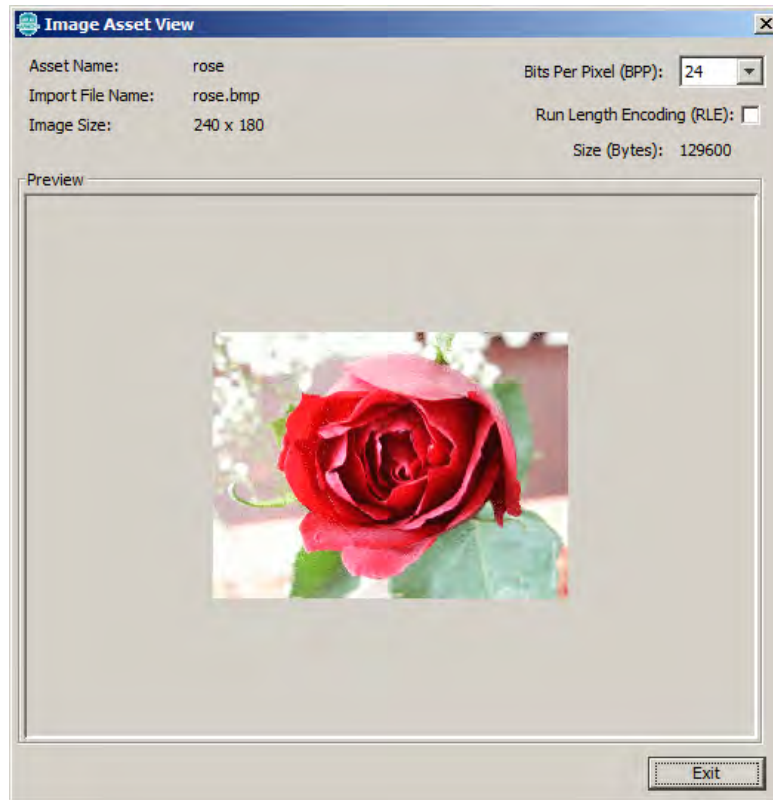
Importing Font Files

Clicking **Font File** opens the Import Font File dialog.
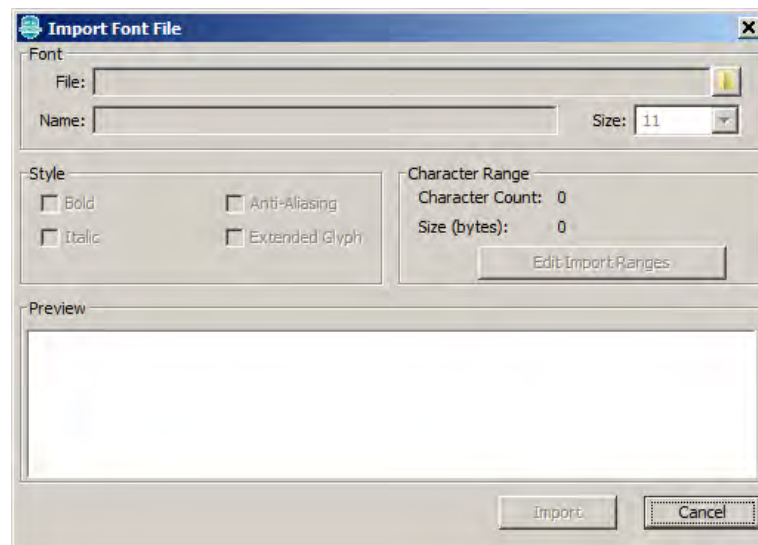
Use the File: Browse icon to locate a font file to import.
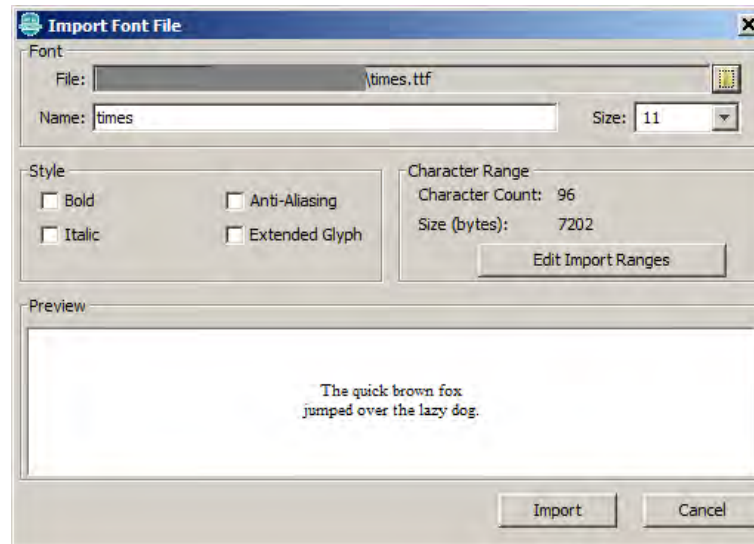
Upon selecting a font, the dialog will enable all of the options and display a preview of the font. Again, the asset name must be unique.

Font Option Descriptions

The following selections are available in Import Font File dialog:

- Bold – Renders the font as **Bold** type
- Italic – *Italicizes* the font
- Anti-aliasing – Enables anti-aliasing for this font in the Graphics Library
- Extended Glyph – Expands the range of imported characters

After finalizing your selections, click **Import**.
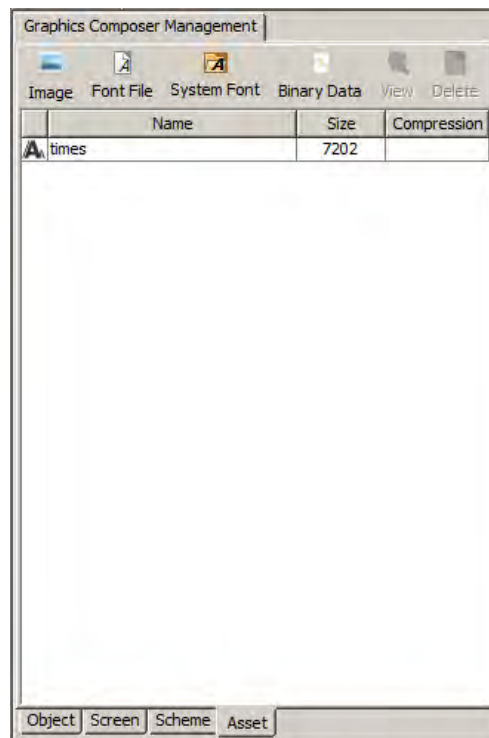
Font Range Selection

The font range dialog provides the method by which users can select multiple font glyph ranges from an imported font file. Only the selected glyph ranges will be converted into program data.

To open the font range configuration dialog click **Font Range**.
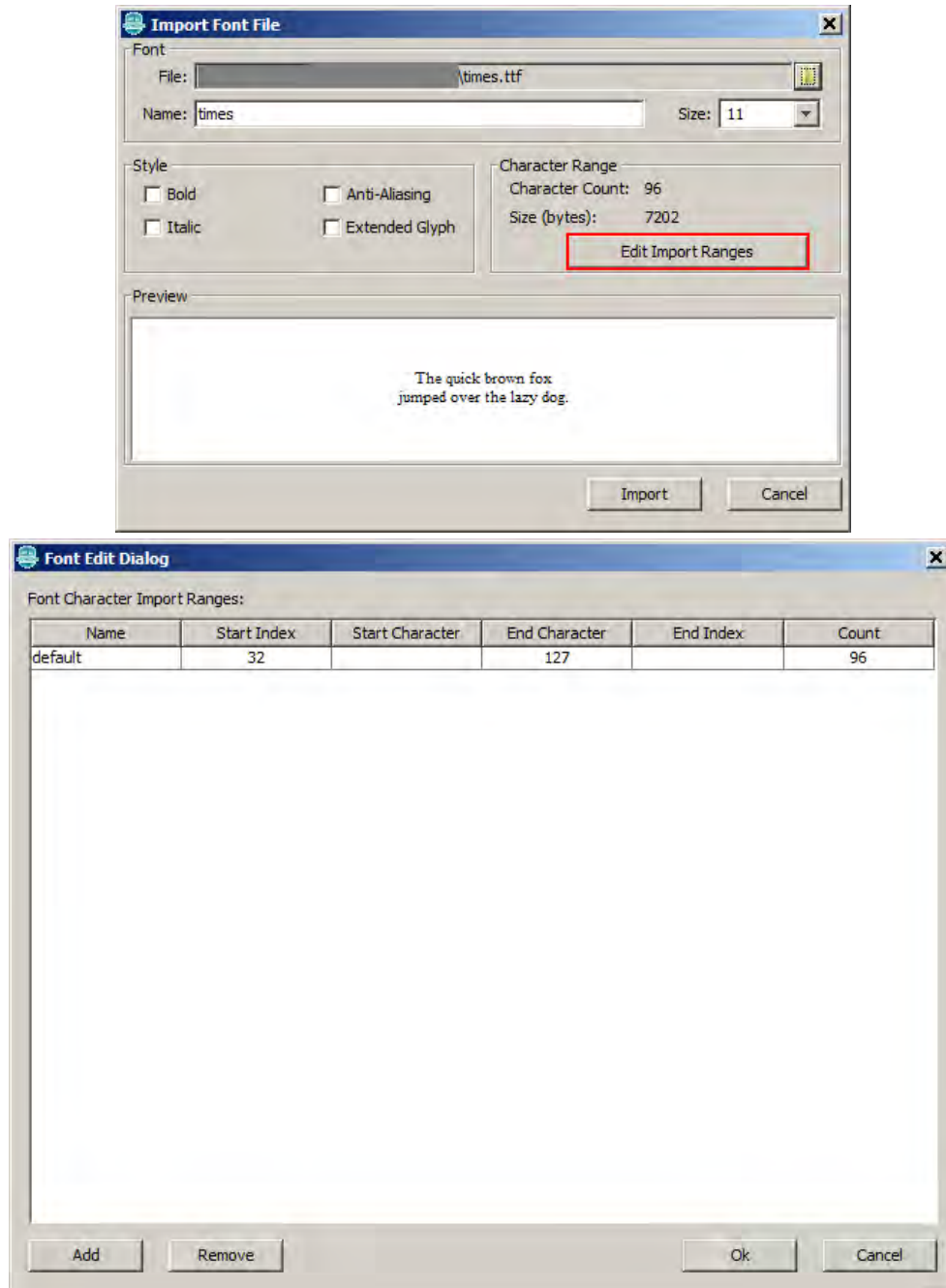
By default, the standard ASCII character range is added for every imported font. Users can either edit this range directly through the table or click **Add** to add a new range.
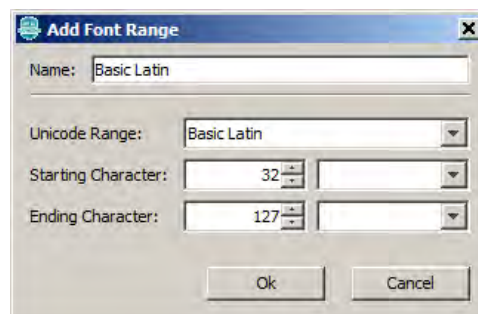
The Add Font Range dialog allows the user to add a glyph import range to the associated font file.

The process for adding a new range is:

1. Provide a glyph range name (if desired)
2. Select an overall Unicode glyph range

3. Choose a starting and ending glyph for this range.

4. Click **OK**.

The new range will appear in the font range list.

16-bit Unicode Character Support

The GRC also supports 16-bit Unicode characters. To guarantee 16-bit Unicode support, be sure to set the Font Character Size to GFX_FONT_SIZE_16 in the MPLAB Harmony Configurator options (*Harmony Framework Configuration > Graphics Library > Harmony Graphics Library > Use Graphics Library? > Use Fonts? > Font Character Size*).

Editing a Font Asset

Font assets can be changed after import. Select the desired font to be changed and click **View** to open the Font Edit dialog.
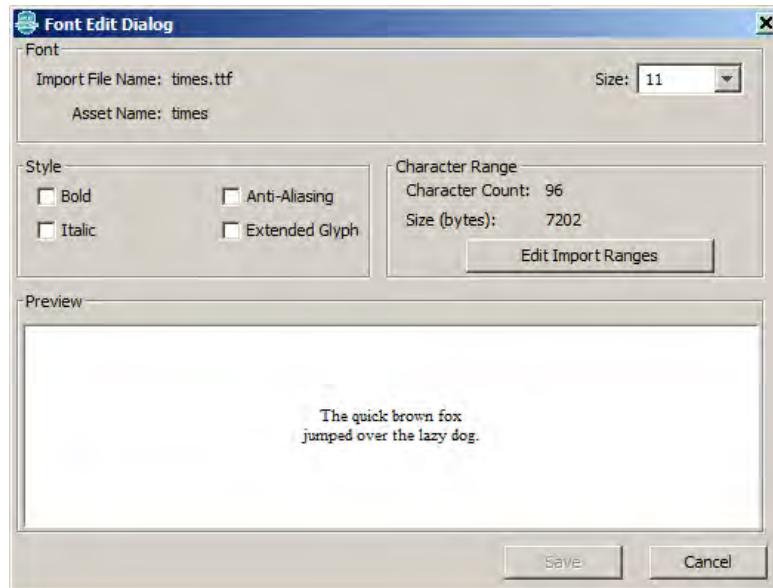
Importing System Fonts

Click **System Font** to open the Import System Font dialog.
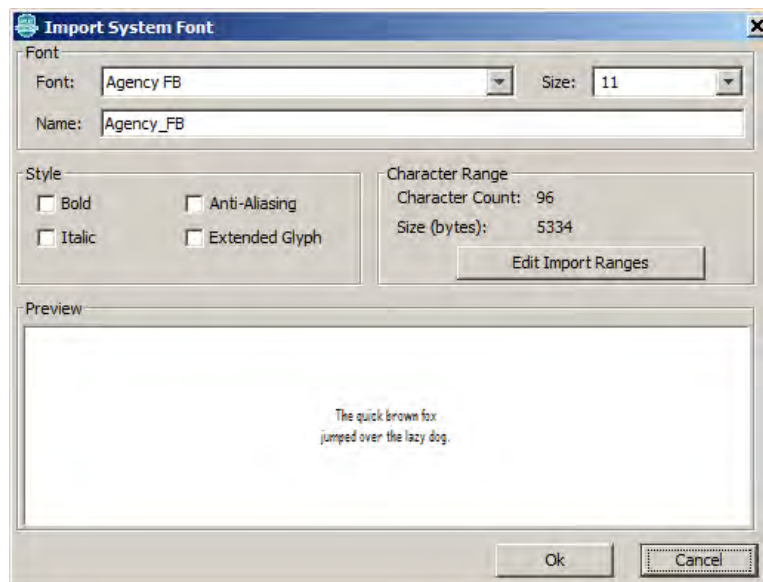
Importing system fonts works similarly to font files with the exception that instead of browsing for a physical file, the user selects from a list of installed fonts.

Importing Binary Data

To import binary data, click **Binary Data**, which opens the Import Binary Data Dialog.
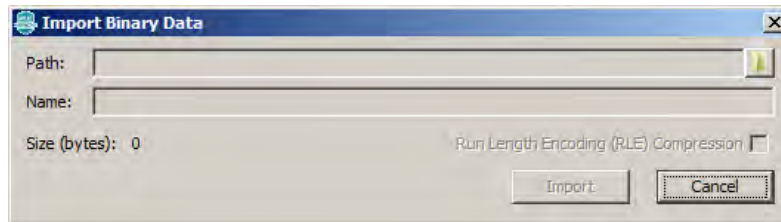
Select a file from using the Path: Browse icon and give it a unique name. The compression flags allows the user to preview if compression provides a size reduction benefit when storing this binary data.

Screen Window

Describes the features of the Screen Window.

Description

The screen window provides an approximate visual representation of the resultant embedded user interface.
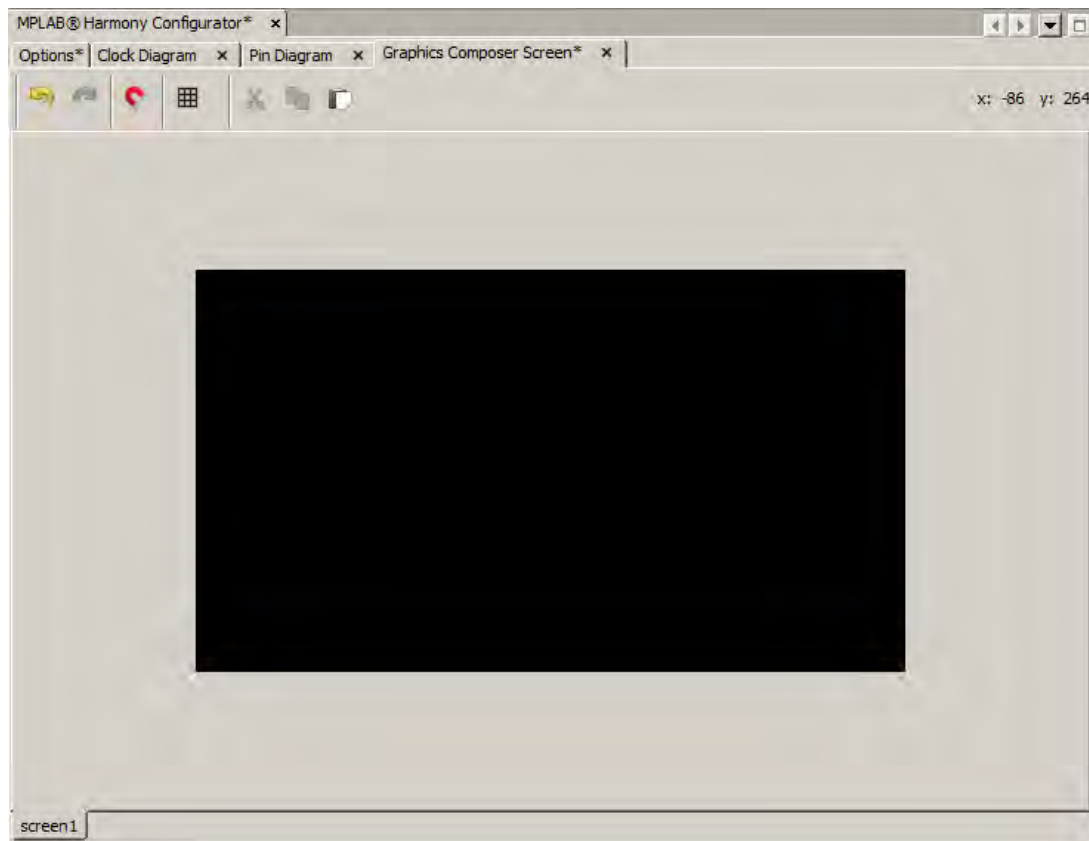
Centered in the screen is an area that matches the size of the currently selected display device. This area will automatically resize when the display device changes in HConfig. The top-left corner of the box is at coordinates 0,0. The current cursor coordinates (in screen space) will be displayed in the top-right corner of the Screen Window when the cursor is inside the Screen Window.

The tabs at the bottom show the screens that are currently visible. These can be changed in the Screen tab of the Composer Management window.

Buttons

The following selections are available:

- Magnet – This icon enables line snapping while moving objects or points
- Grid – This icon enables a visual grid that can be snapped to. When selected, the user can adjust the grid size and color.
- Left Arrow – This icon performs an "undo" of the last action
- Right Arrow – This icon performs a "redo" of the last action
- Scissors – This icon performs a "cut" of the currently selected objects
- Pages – This icon copies the currently selected objects
- Clipboard – This icon pastes the currently selected objects to the clipboard

Manipulating Objects

The Screen window provides the ability to graphically configure the objects of a screen. Given a screen with the following layout, left click an object to select it.
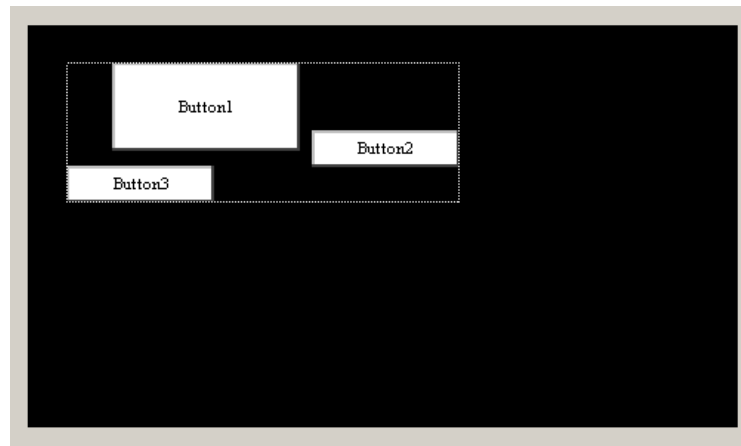
The manipulator can be moved by left-click-dragging it. The white circles represent the handles that allow the manipulator to be resized
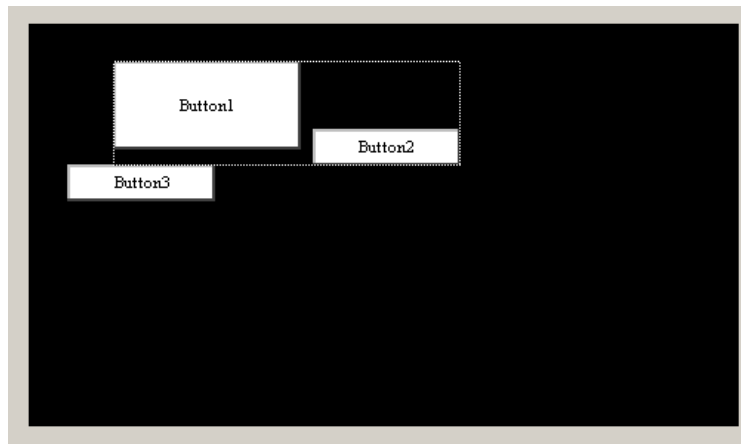
Selecting Multiple Objects

Several methods are available to select objects. Refer to the Object Tab topic for details.

Marquee Select

Left click in the screen and drag the marquee (i.e., dotted) box around the desired objects.
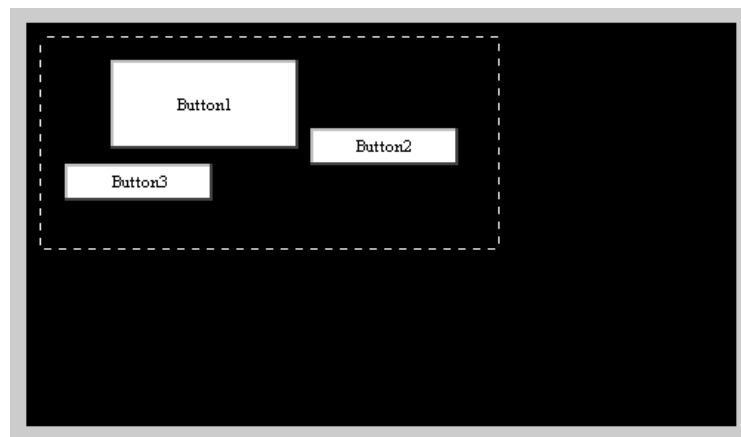
The Object Group Move box will appear. Drag the box to move the selected objects.
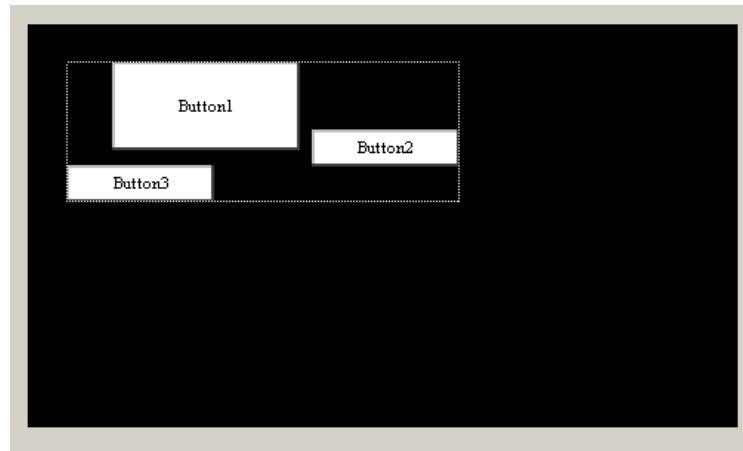
Managing Object Selection

Objects can be added or removed from a group selection. To remove Button3 from the previous selection, press and hold the <Ctrl> key and left click inside Button3. Button3 will be unselected. The object can be added back to the group by pressing and holding the <Shift> key and left clicking the Button3 object. A toggling selection can be performed by pressing and holding <Ctrl>+<Shift> and selecting either with a left click or a marquee box selection.
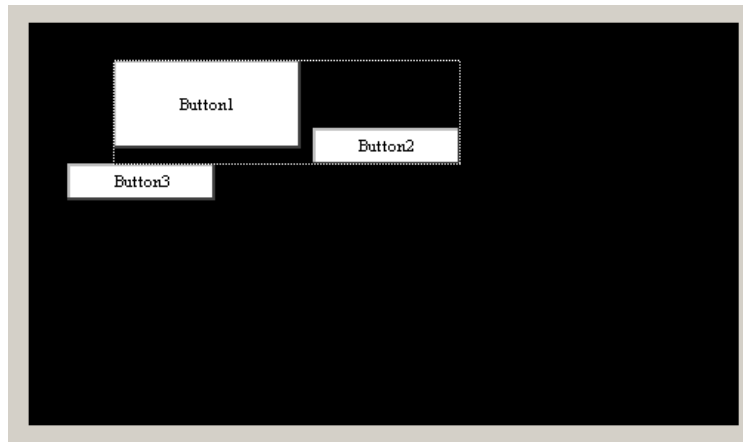
Before:
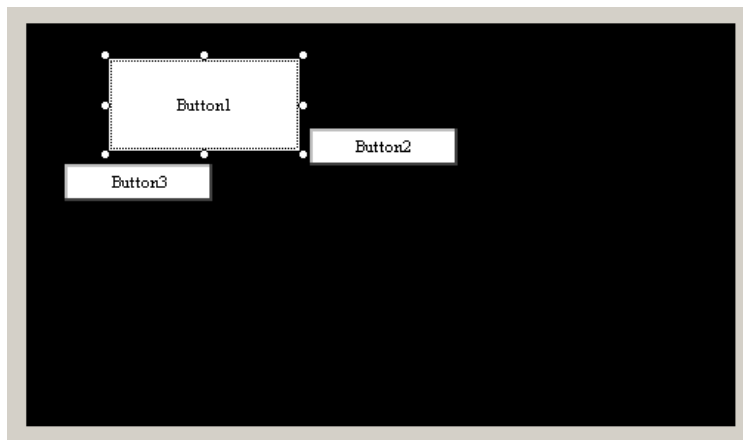
After:
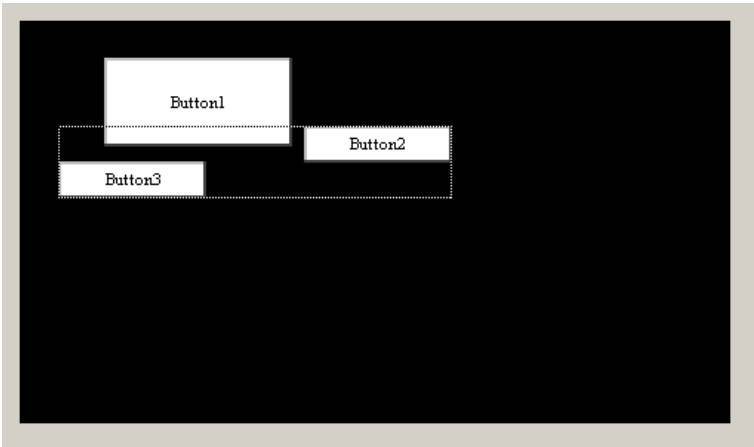
Order Management Through the Screen Window

Object ordering can be managed from the Screen Window as well as the Object tab in the Composer Management Window. Right click an object or a group of objects to display the context menu.
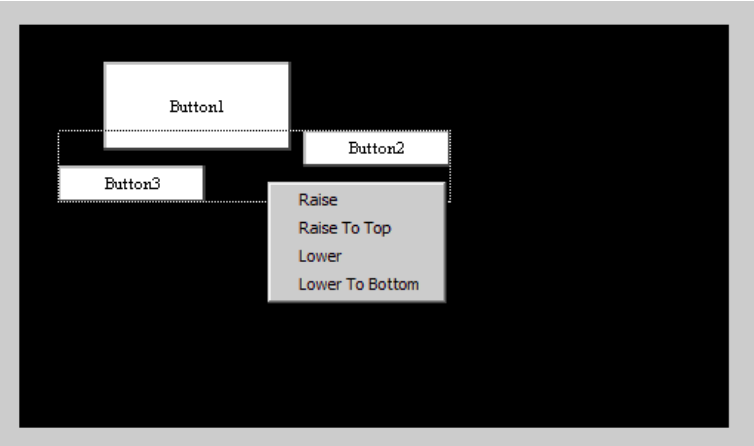
Properties Window

Describes the features of the Properties Window.

Description

The Properties Window displays options for the currently selected object, 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 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.
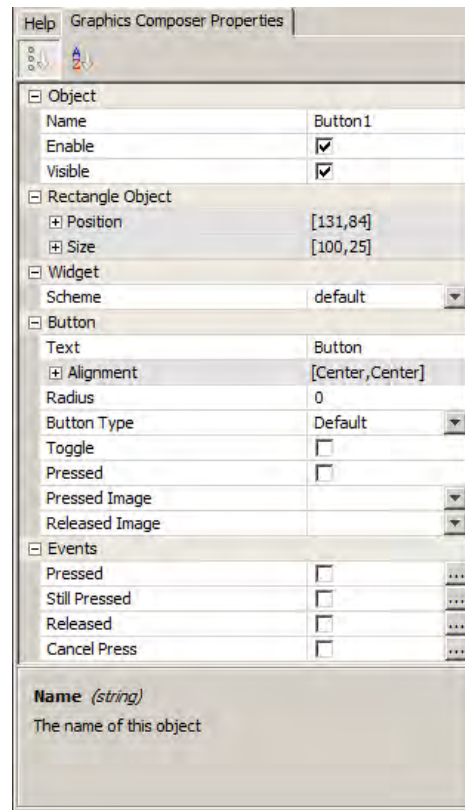
Notice that the bottom panel provides help text for each property, which provides the type of data expected and a description of what the property represents. Some properties are configured to reject invalid settings.

Color Picker Dialog

The Color Picker dialog 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.
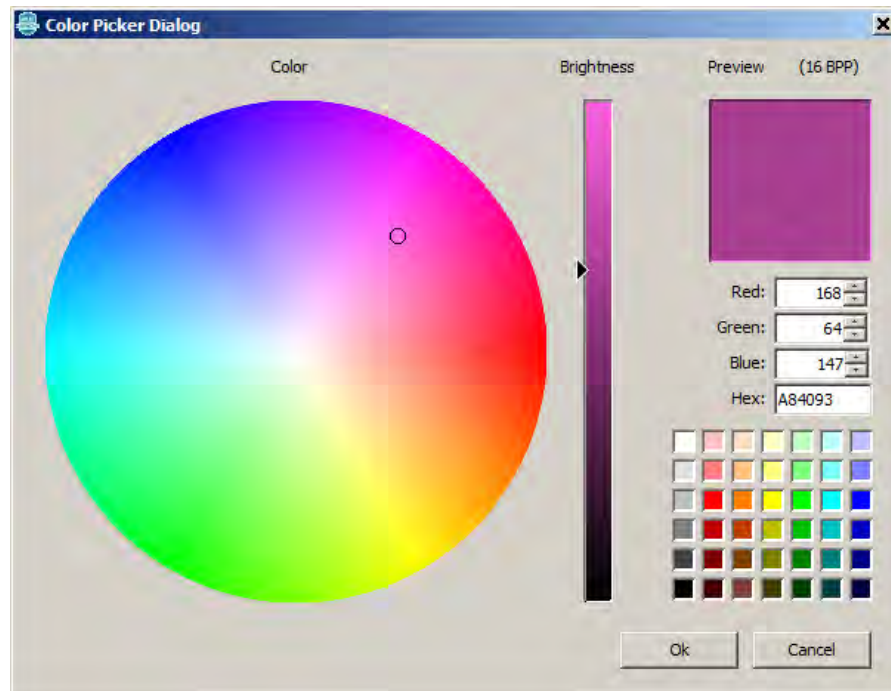
Event Generation

This topic describes using the graphics composer to generate events.

Description

Some objects have events that can be enabled and defined. The graphics composer provides the capability to generate event handler code using a visual interface.

As shown in the following figure, these events are associated with the GOL Button Widget.
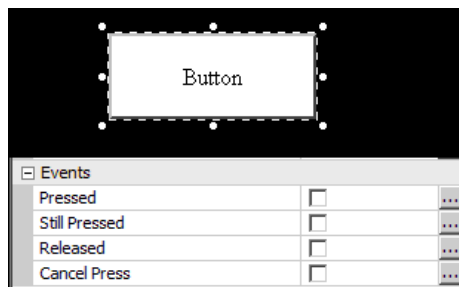
Defining Events

To define events, select the object on the screen for which events are to be defined. If an object that supports events is selected, the properties table will display the events that can be defined for that object. Select the check box to enable the event. If the generate process is run at this point, an empty event handler will be created for that event.
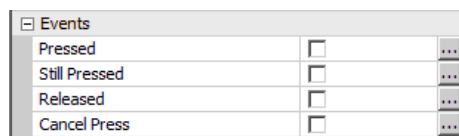
Defining Event Actions

Each event property has a corresponding ellipsis button. Clicking this button opens the Event Editor dialog.

In this case, the Event Editor displays the event state for the "Pressed" event of a GOL Button Widget. To add an action, click **Add**. This action will open the Create Action dialog. Actions can be edited after creation and can be removed using this dialog. Action code is generated in top-down order. The arrows on the right change can the order of the action list to configure action precedence.

There are two types of actions that can be defined for an event: Template and Custom. Template events allow the user to choose a source screen, an action target, an action, and potentially data associated with an action.

Creating a Template Action

To create a template action, follow these steps:

1. From the first column select the screen that is, or contains, the target of this action.
2. From the second column select the target of this action. Screens are highlighted in green.
3. From the third column select the action to perform on the selected target.
4. From the fourth column input the requested data for the action (not always required).

In this example "screen1" will be selected as the source.
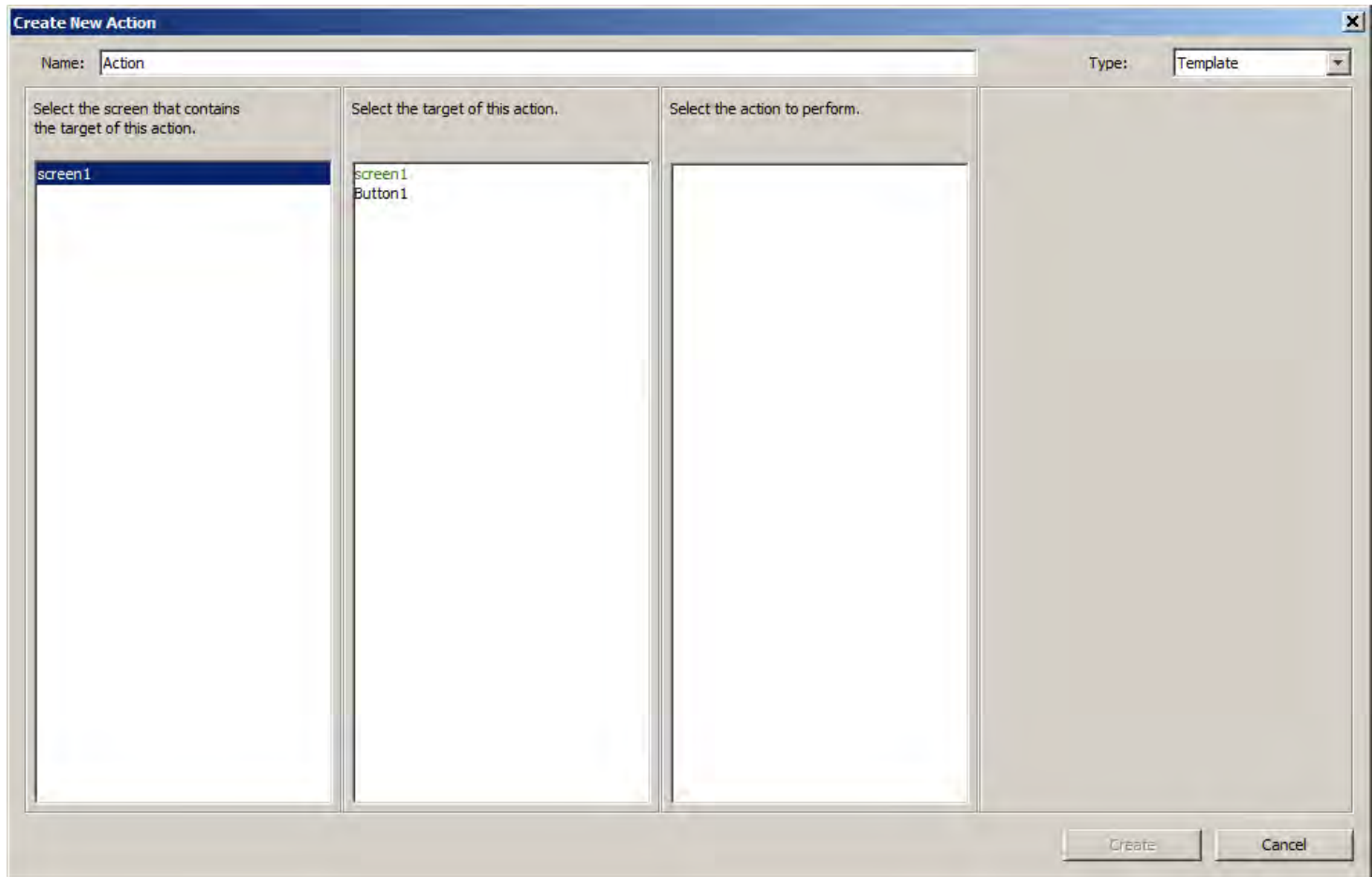
The object named "Button1" will be selected as the target.
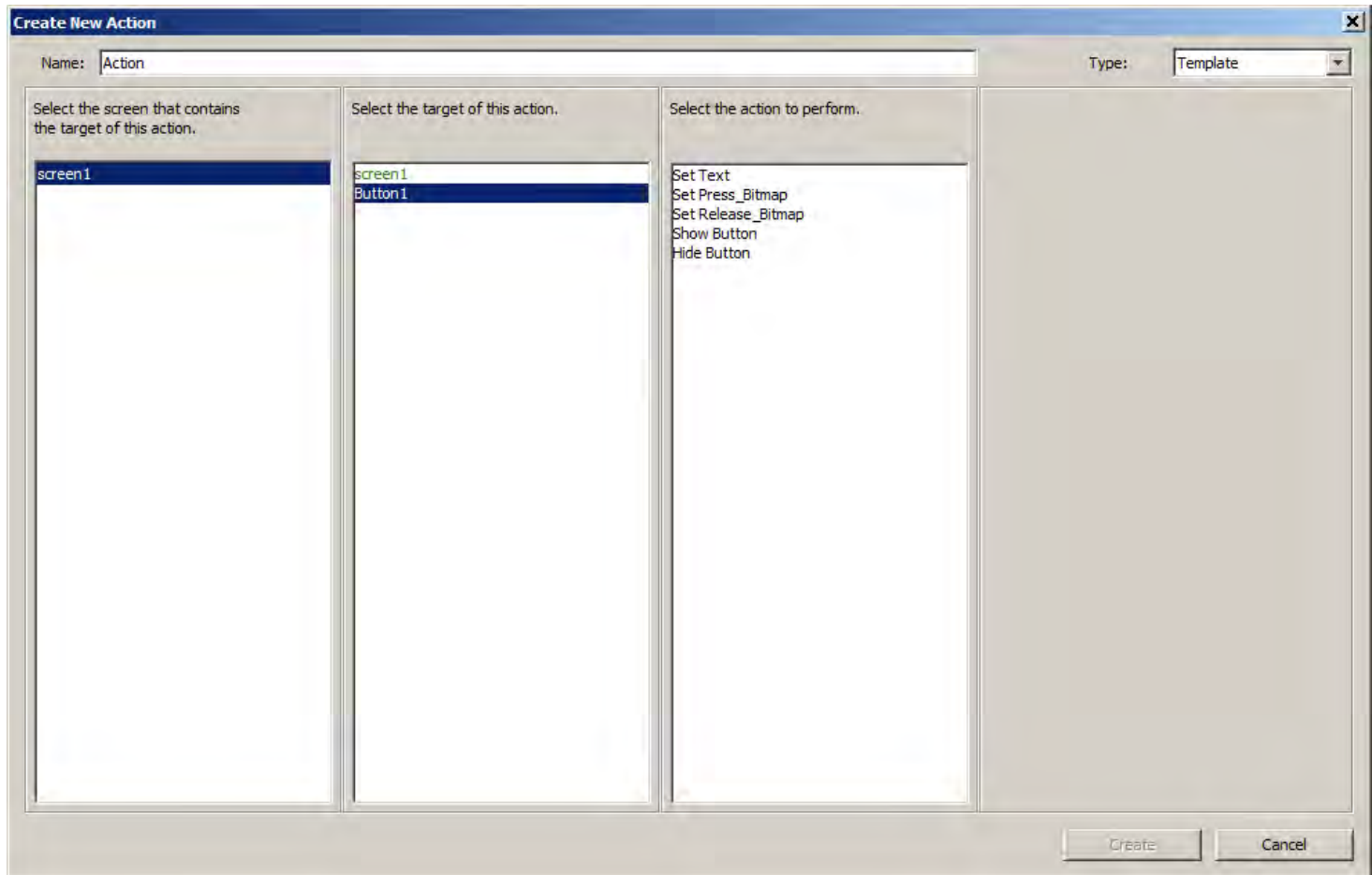
The action that will be selected is "Set Text".
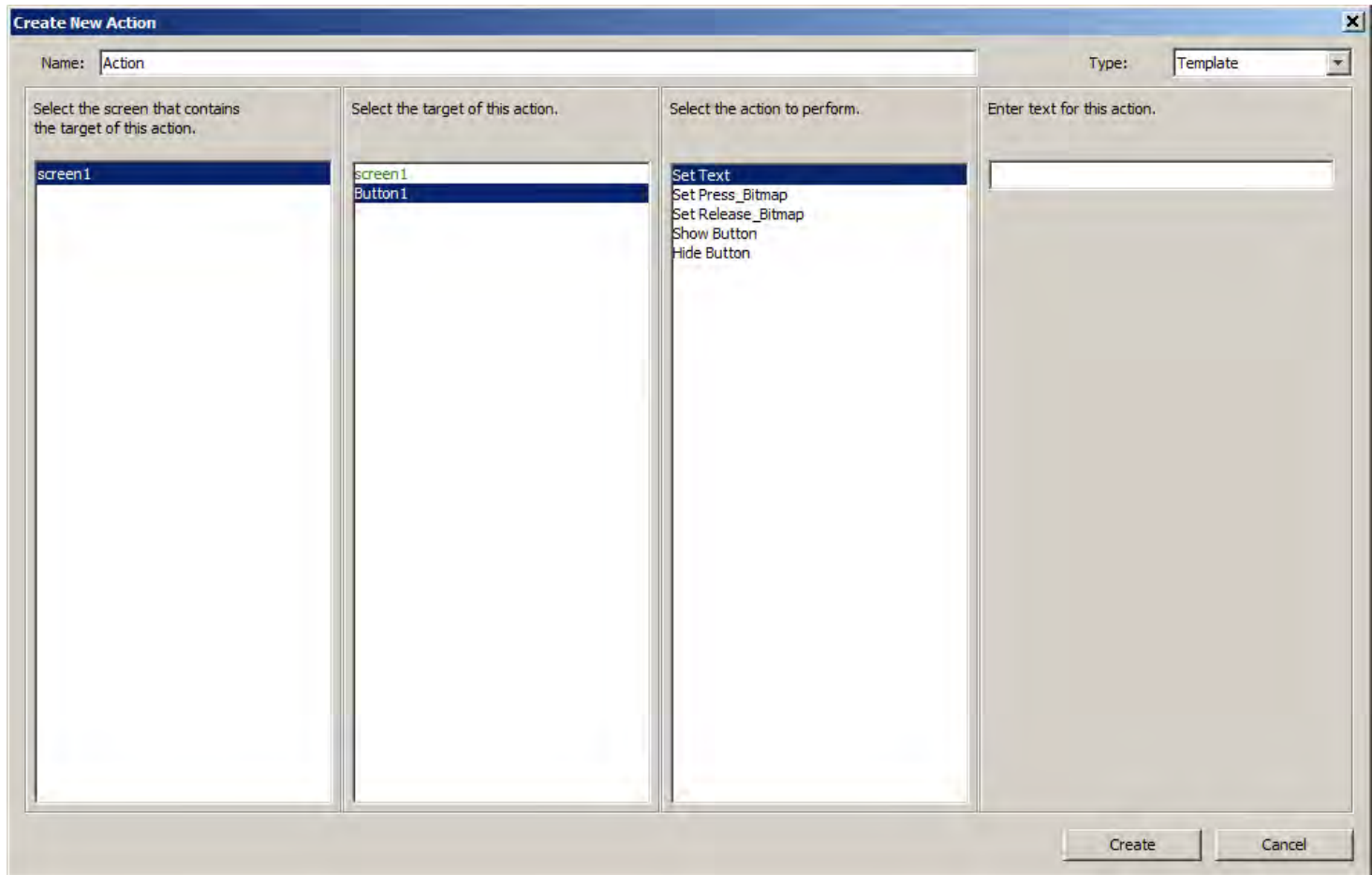
This action requires that the user input the text to assign to "Button 1". The user can also assign a unique name to an action.
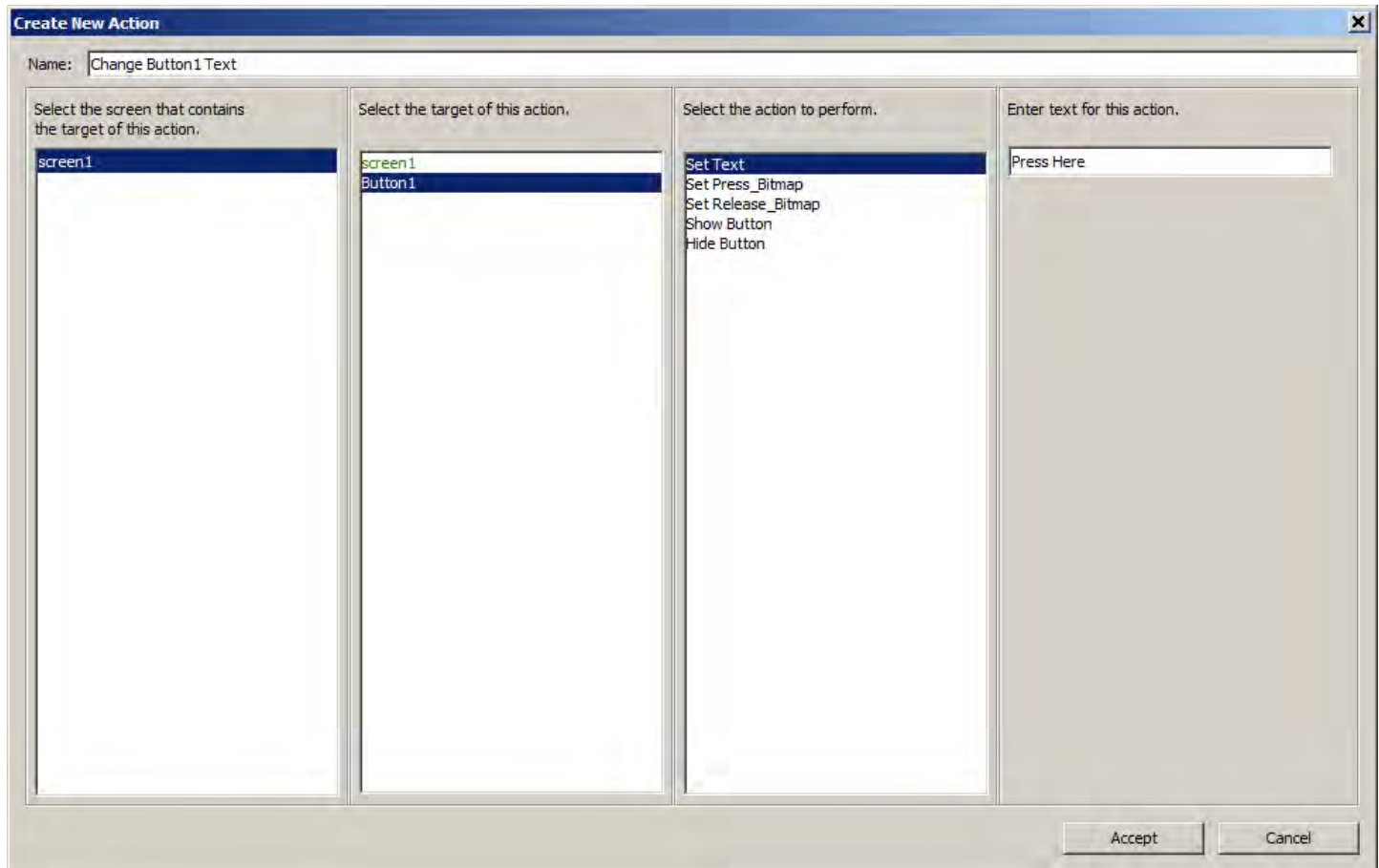
Once these steps are complete, the dialog will enable the "Create" button and the action can be finalized. When **Create** is clicked, the dialog will close and the action will be added to the action list for the event.
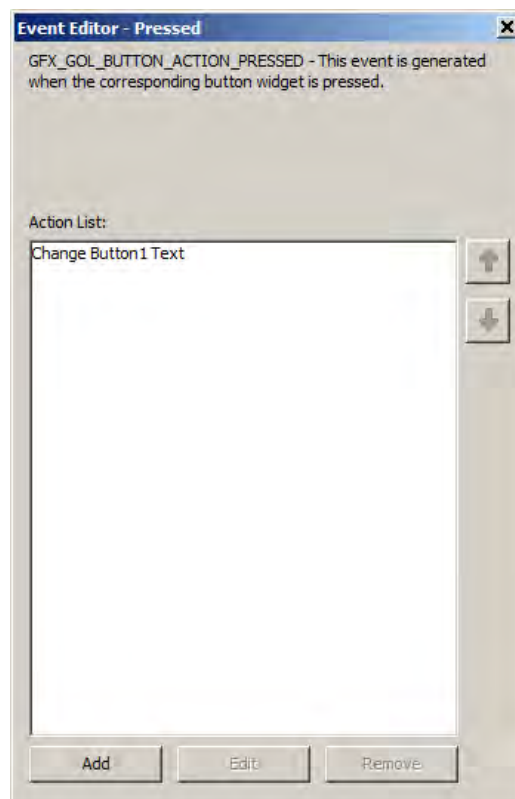
Creating a Custom Action

The second type of action is a Custom action. This type allows the user to include custom code and have it inserted into the event handler

function. The graphics composer is not responsible for ensuring that the code input is valid.

To create a custom action click **Create** in the Event Editor dialog and change the type selection in the top right corner to Custom.
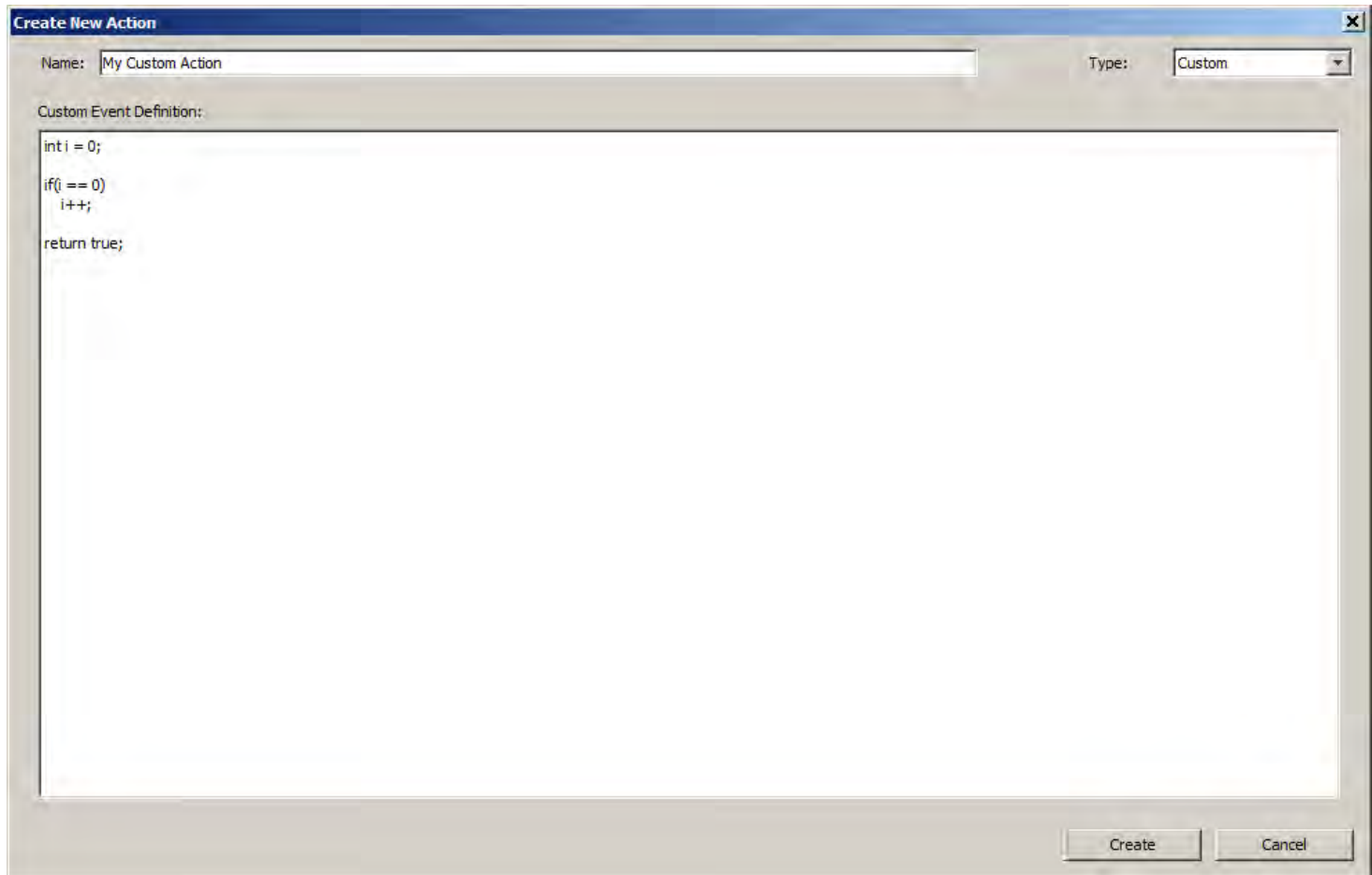
Clicking **Create** adds the custom action to the event action list.
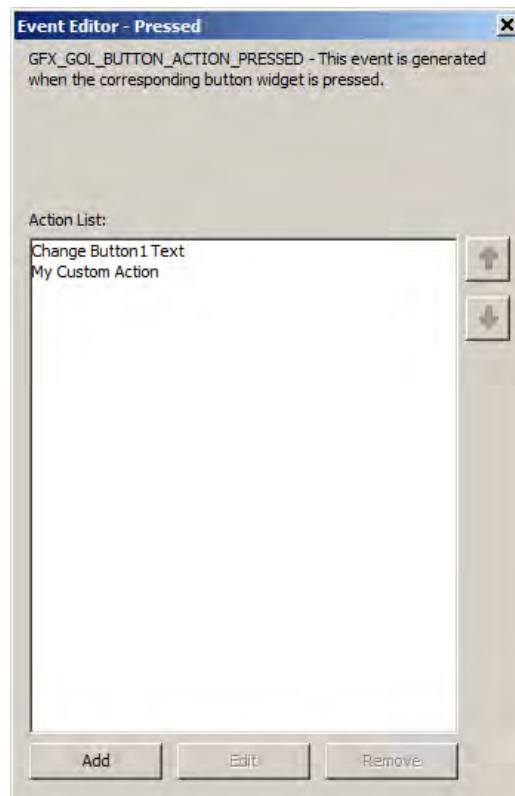
The code generator will automatically generate or include the defined actions during generation.

```
Remarks:
    Handles GFX GOL Button events
*/
bool GFX_HGC_MsgButtons(uint16_t objMsg, GFX_GOL_OBJ_HEADER *pObj)
{
    switch (GFX_GOL_ObjectIDGet(pObj))
    {
        case Button1:
            if (objMsg == GFX_GOL_BUTTON_ACTION_PRESSED)
            {
                // Button Pressed Event Code
                // Change Button1 Text
                GFX_GOL_ButtonTextSet(((GFX_GOL_BUTTON*)(GFX_GOL_ObjectFind(GFX_INDEX_0, Button1))), (GFX_XCHAR*)"Press Here");
                GFX_GOL_ObjectStateSet(((GFX_GOL_BUTTON*)(GFX_GOL_ObjectFind(GFX_INDEX_0, Button1))), GFX_GOL_BUTTON_DRAW_STATE);

                // My Custom Action
                int i = 0;

                if(i == 0)
                    i++;

                return true;
            }

            return true;
        default:
            return false; // process by default
    }

    return true;
}
```

Auto-configuration

MPLAB Harmony Composer will automatically enable touch input support in the MHC tree when a event is created and enabled. This setting can be manually overridden in the MHC tree. Refer to *Graphics Library > Harmony Graphics Library > Use Graphics Library > Use Input Devices? > Enable Touchscreen Support* in MHC for details.

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.

For function calls to the gfx_gol and gol_primitives library, the graphics composer will add the `gfx_hgc_definitions.h` and `gfx_hgc_definitions.c` files to `app/system_config/<configuration_name>/framework/gfx` folder in the header and source file structure of the MPLAB X IDE Project.

For generated asset data, the graphics composer will add the `gfx_resources.h`, `gfx_resources.S`, and `gfx_resources_reference.c` files to the `app/system_config/<configuration_name>/framework/gfx` folder in the header and source file structure of the MPLAB X IDE Project.
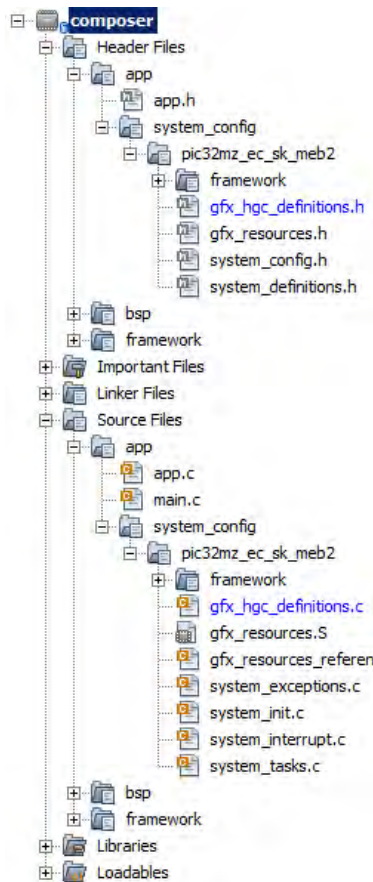
You may want to monitor the progress of the generation in the Output window.

The asset resource is the first to get generated. To confirm accurate file generation, look for output such as that highlighted in the following figure.
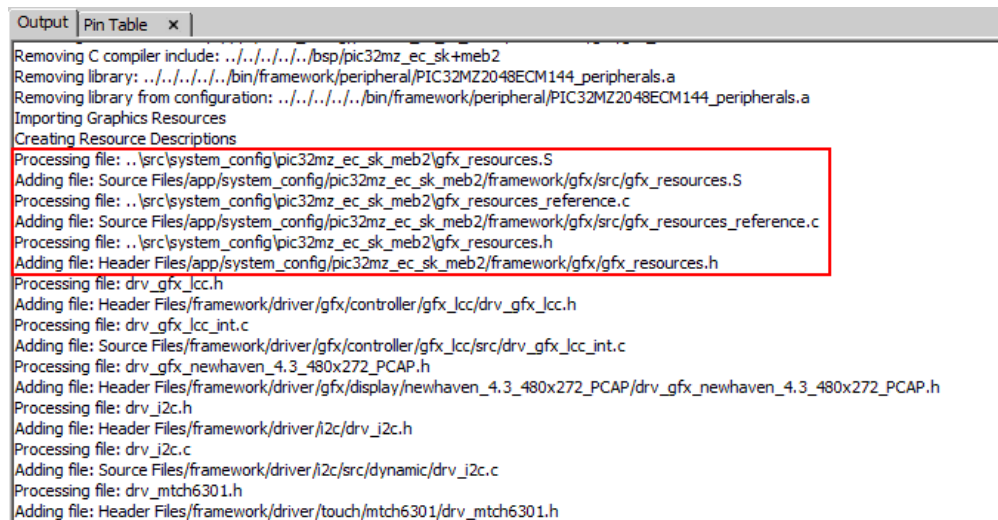
For accurate generation of the draw function calls, you should expect output such as that highlighted in the following figure.

Saving and Loading Data

This topic describes using the graphics composer to save and load data.

Description

The graphics composer saves and loads its data into the `configuration.xml` file of the MHC configuration. This file is always located within `<install-dir>\apps\<feature>\<demonstration_name>\firmware\src\system_config\$CONFIGURATION_NAME`. The saved data is loaded when MHC starts and is saved when the configuration is saved through the Save or Save As dialogs.
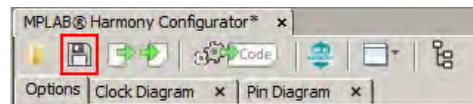
Importing and Exporting Data

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

Description

The MPLAB Harmony Graphics Composer provides the capability for users to import and export graphics composer-related data. The user can export the state of an existing graphics composer configuration, import another graphics composer configuration, and import projects from the Graphics Display Designer X (GDD X) utility.

The import and export interfaces are located in the Configuration dialog of the MPLAB Harmony Configurator, which is accessible from the Options tab.
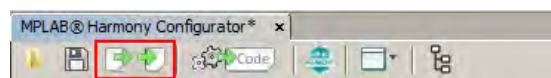
Importing Data

To import data into graphics composer, click **Import** from the main window toolbar. The Import dialog will appear.
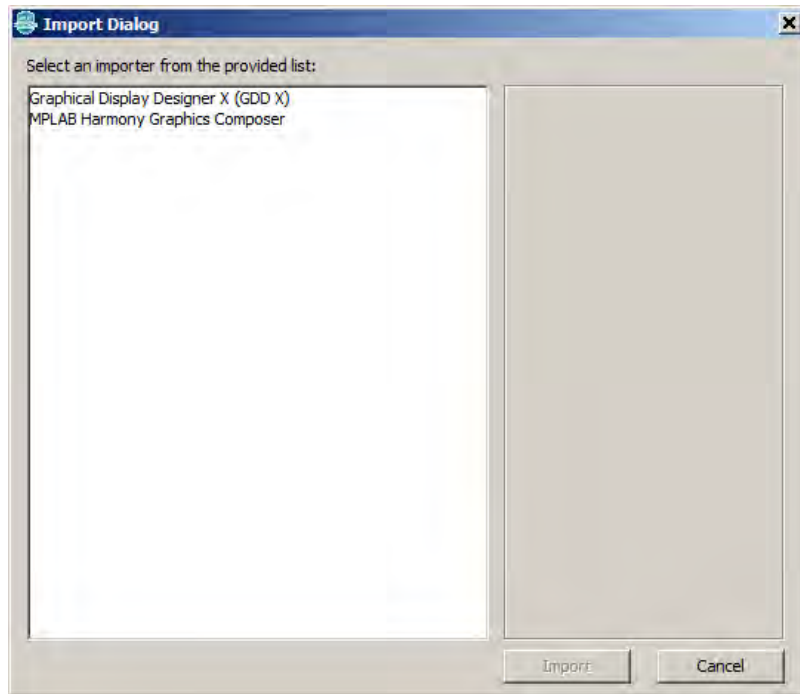
The user can choose to import either GDD X or graphics composer data. Upon selecting a format and clicking **Import**, a path dialog will appear and the user can browse to either a graphics composer XML file or a GDD X project file.
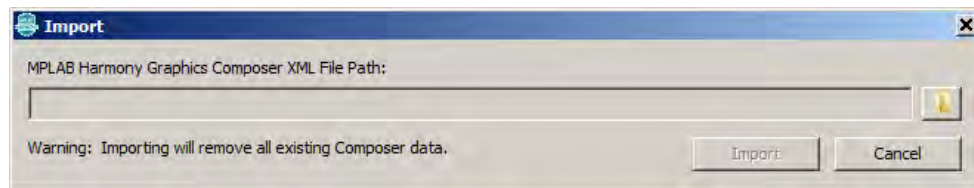
⚠️ **Warning**   Importing data will remove all currently existing graphics composer data.

Exporting Data

To export a Composer configuration click **Export**.
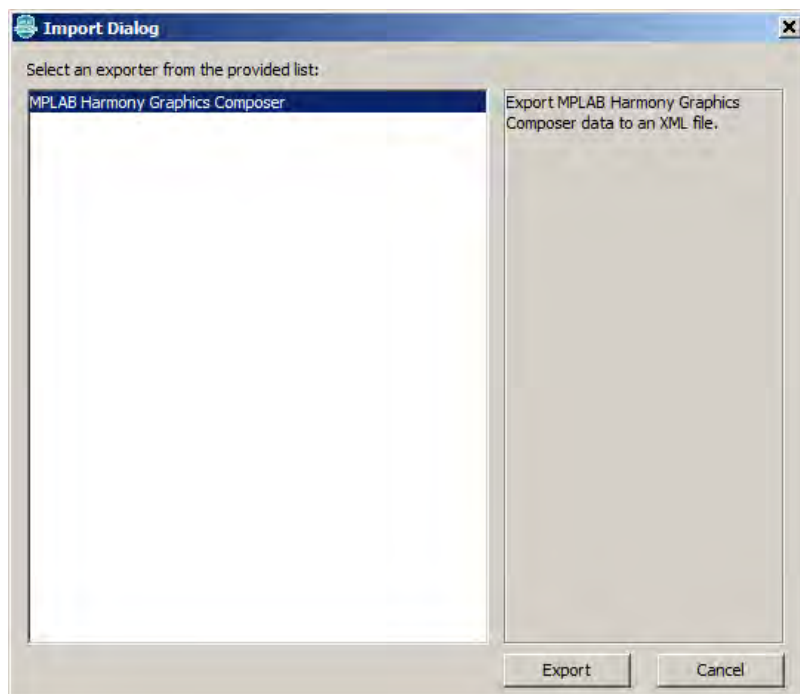
Select MPLAB Harmony Graphics Composer from the list and click **Export**.
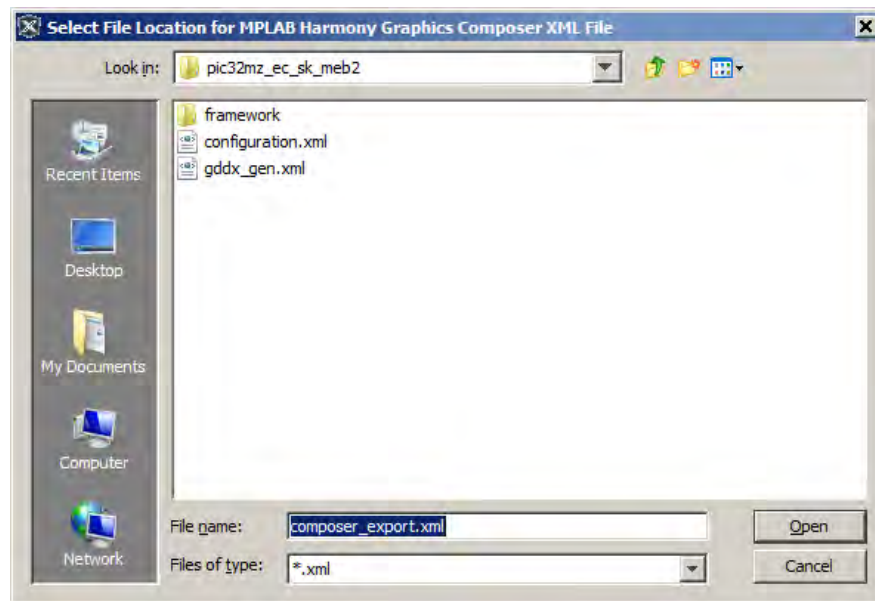
Select the file path where the exported data should be placed and click **Open**. The current graphics composer data will be written into this file.

Managing Graphics Composer Features

This topic describes how to manage graphics composer features.

Description

Users can easily enable or disable all graphics composer features using the option **Use MPLAB Harmony Graphics Composer Design**.

If this configuration flag is enabled, the graphics composer will generate its respective state machine code and will also take responsibility for managing many of the Graphics Library options.

Index