MPLAB® Harmony Help - Volume V - Third Party Products

MPLAB Harmony Integrated Software Framework v1.11

Volume V: Third-Party Products

This volume describes the third-party libraries that are available in MPLAB Harmony.

Third-Party Products Overview

This section provides an overview for the third-party products that are included in MPLAB Harmony.

Introduction

This topic provides an overview of the Third-Party Libraries in MPLAB Harmony.

Description

MPLAB Harmony is a flexible, abstracted, fully integrated firmware development platform for PIC32 microcontrollers, which enables seamless integration of third-party solutions, such as RTOS, Middleware, Drivers, and so on, into the software framework.

Important Licensing Information

OPENRTOS

The OPENRTOS demonstrations provided in MPLAB Harmony use the OPENRTOS evaluation license, which is meant for demonstration purposes only. Customers desiring development and production on OPENRTOS must procure a suitable license. Please refer to one of the following documents, which are located in the `third-party` folder of the MPLAB Harmony installation, for information on obtaining an evaluation license for your device:

- `OpenRTOS Click Thru Eval License PIC32MXxx.pdf`
- `OpenRTOS Click Thru Eval License PIC32MZxx.pdf`

Micriμm

All μC/OS-III demonstrations have added the `crt0.S` "C" run-time library start-up file to the project. The demonstration sets the linker option "do not link startup code". This is necessary for μC/OS-III to work correctly with PIC32 devices as the general exception vector is located in `crt0.S`. μC/OS-III overrides this interrupt source (general exception handler) to perform OS-specific functionality.

If the user wants to implement their own application using μC/OS-III and a PIC32 device, they must add the `crt0.S` file to their project and override the general exception interrupt vector. See the current RTOS examples for this implementation.

A `crt0.S` template file can be found in the MPLAB XC32 C/C++ Compiler installation directory:
`..\Microchip\xc32\<version>\pic32-libs\libpic32`.

Important! The Micriμm μC/OS-II and μC/OS-III source code that is distributed with MPLAB Harmony is for FREE short-term evaluation, for educational use, or peaceful research. If you plan or intend to use μC/OS-II and μC/OS-III in a commercial application/product, you need to contact Micriμm to properly license μC/OS-II and μC/OS-III for its use in your application/product. The source code is provided for your convenience and to help you experience μC/OS-II and μC/OS-III. The fact the source is provided does NOT mean that you can use it commercially without paying a licensing fee. Knowledge of the source code may NOT be used to develop a similar product. If you are unsure about whether you need to obtain a license for your application, please contact Micriμm and discuss the intended use with a sales representative (www.micrium.com).

Express Logic ThreadX

The source code for the ThreadX demonstration is not freely distributed. To obtain source code and the proper licensing agreement go to the Express Logic ThreadX website: http://rtos.com/products/threadx/.

Software License Agreement

Refer to the MPLAB Harmony Integrated Software Framework *Software License Agreement* for complete licensing information. A copy of this agreement is available in the `<install-dir>/doc` folder of your MPLAB Harmony installation.

Decoder Library Help

This section provides information on the Decoder Library.

Introduction

This topic provides an overview of the Decoder Library in MPLAB Harmony.

Description

The Decoder Library source files provided in your installation of MPLAB Harmony are based on the Version 6B release from the Independent JPEG Group (IJG). IJG is an informal group that writes and distributes a widely used free library for JPEG image compression.

More Information

For more information, please visit: http://ijg.org/. IJG documentation and archive files are accessible at: http://ijg.org/files/.

Additional information is also available from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

Express Logic ThreadX Library Help

This section provides information on the Express Logic ThreadX Library.

Introduction

This topic provides an overview of the Express Logic ThreadX Library in MPLAB Harmony.

Description

ThreadX is Express Logic's advanced Real-Time Operating System (RTOS) designed specifically for deeply embedded applications. ThreadX provides advanced scheduling facilities, message passing, interrupt management, and messaging services, as well as many others. ThreadX has many advanced features, including its picokernel™ architecture, preemption-threshold™ scheduling, event-chaining,™ and a rich set of system services.

More Information

For more information, please read the related documentation, which is available at: http://rtos.com/products/threadx/. Additional information is also available at http://rtos.com/products/threadx/Microchip_PIC32 and from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

Demonstrations

See RTOS Demonstrations for information.

FreeRTOS Library Help

This section provides information on the FreeRTOS™ Library.

Introduction

This topic provides an overview of the FreeRTOS™ Library in MPLAB Harmony.

Description

FreeRTOS is a small footprint, portable, preemptive, open source, real time kernel that has been designed specifically for use on microcontrollers.

FreeRTOS v8.2.3 has added support for the PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Family devices.

Use the following procedure to enable PIC32MZ EF FPU support in FreeRTOS tasks:

1. Define the configuration macro, configUSE_TASK_FPU_SUPPORT, to '1' in the `FreeRTOSConfig.h` file. Enabling this configuration macro allows the application to use FPU operations in `main()` before scheduler starts.
2. When it is desired to have FPU operations in selected FreeRTOS tasks, call the vPortTaskUsesFPU function at the start of *only* those tasks that use the FPU.

More Information

For more information, please read the FreeRTOS Quick Start Guide, which is available at: http://www.freertos.org/FreeRTOS-quick-start-guide.html. Additional information is also available from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

Demonstrations

See RTOS Demonstrations for information.

InterNiche Library Help

This section provides information on the InterNiche Library.

Introduction

This topic provides an overview of the InterNiche Library in MPLAB Harmony.

Description

Legal Disclaimer

A particular InterNiche library is provided as object code for a specific MCU family, and is licensed for distribution with a single product. For additional information, contact sales@iniche.com.

More Information

For more information, please read the related documentation, which is available at: . Additional information is also available at http://www.iniche.com/ and from the Microchip Third-Party Software Stacks web page: http://www.microchip.com/devtoolthirdparty

iREASONING Networks MIB Browser

This section describes the iREASONING Networks MIB Browser, which can be used when running certain MPLAB Harmony demonstration applications.

Introduction

This topic provides an overview of the iREASONING Networks MIB Browser.

Description

This help file describes how to use the iREASONING Networks MIB Browser to run the TCP/IP SNMP demonstration applications. The MIB Browser can be obtained from: http://www.ireasoning.com/downloadmibbrowserlicense.shtml. The MIB script upload, the MIB tree structure display, and the SNMP query mechanism procedures vary from browser to browser.

Important! The use of a MIB browser or other third-party products may require that users review and agree to the terms of a license. Microchip's reference to the iREASONING Networks MIB Browser is for the users' convenience. It is the user's responsibility to obtain information about, and comply with the terms of, any applicable licenses.

Getting Started

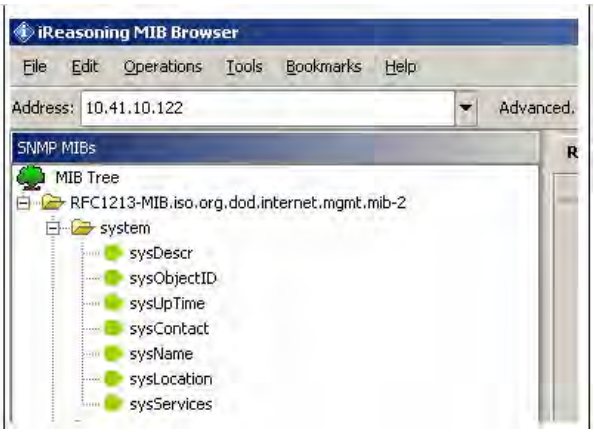
This topic describes how to get started after installing the iREASONING Networks MIB Browser.

Description

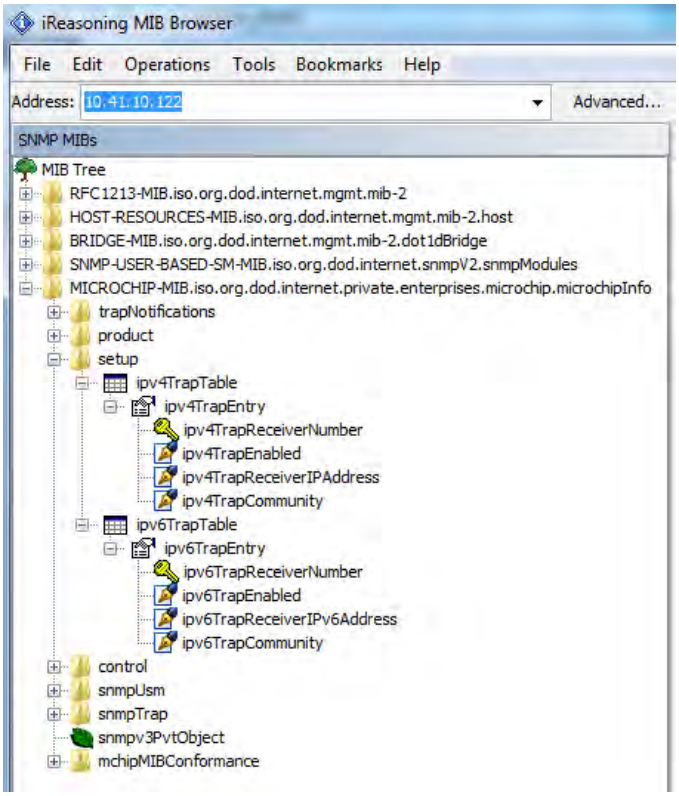
Once your browser installation has been completed, perform the following steps:

1. Copy the `mchip.mib` file to the MIB file directory of your browser (e.g., `C:\Program Files\ireasoning\mibbrowser\mibs`).
2. Open the MIB Browser and select *File>Load MIBs*, and select the `mchip.mib`, `RFC1213.mib`, and `SNMP-FRAMEWORK-MIB.mib` (If SNMPv3 server is enabled) files. The Microchip MIB directory will be displayed in the SNMP MIB pane.

The minimum set of RFC 1213 MIB2 variables that are required to identify the Microchip node as an SNMP node to the network are implemented. These variables can be accessed by any SNMP browser with a "public" type community name. Refer to the Microchip application note, AN870 "SNMP V2c Agent for Microchip TCP/IP Stack" (DS00000870) for more details on the MIB scripts, community names, and demonstration SNMP MIB variable tree structure. The following figure shows the variables implemented in the Microchip SNMP Agent.



The ASN.1 format `mchip.mib` file is defined with a private variable tree structure for the MIB variables. Also the `mchip.mib` file is added with a number of OIDs, which can be accessed only with a SNMPv3 request. The browser can access every variable in the MIB database provided the community name matches. The access to the MIB variables is restricted to the type of the request. The RFC1213 MIB variables can be accessed with a SNMPv2c/v3 request. However, the `SNMP-FRAMEWORK-MIB.mib` variables can only be accessed with a SNMPv3 request if the credentials are matched and the message is authenticated. To modify these MIB variables, the corresponding changes must be made to both MIB scripts (`snmp.mib` and `mchip.mib`). The following figure shows the Microchip private MIB variable tree structure in the browser.



Configuring the Browser

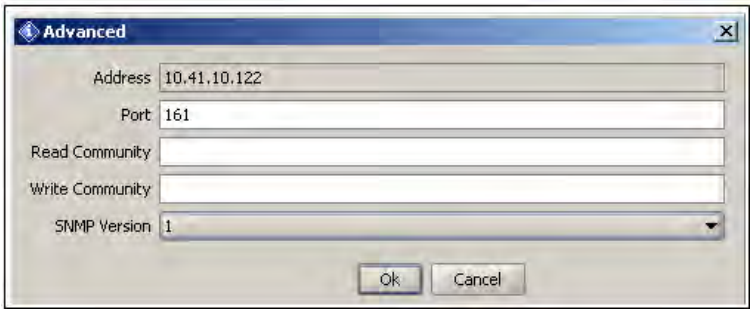
This topic describes how to configure the iREASONING Networks MIB Browser for use with the TCP/IP SNMP demonstrations.

Description

To configure the MIB browser:

1. Select the 'Advanced' tab in the browser.

The following configuration window appears:



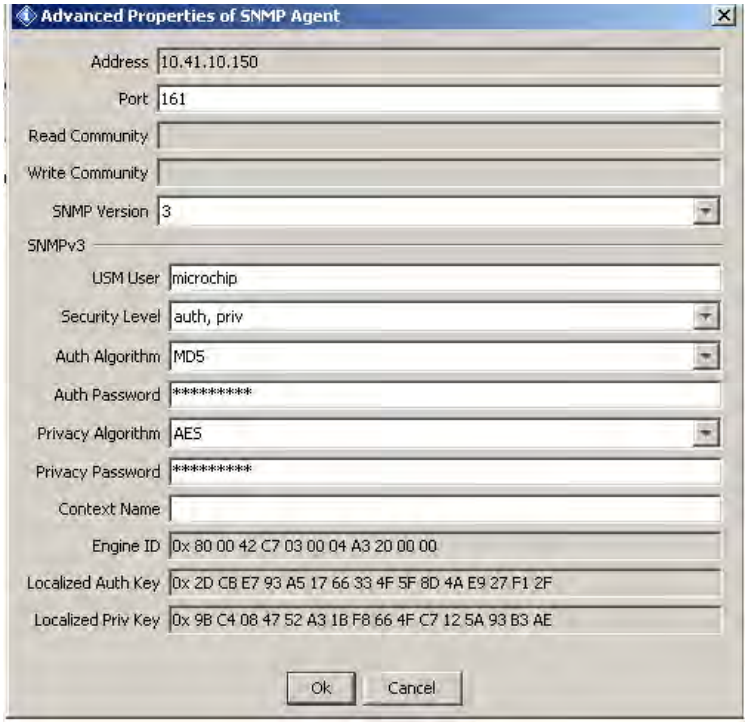
2. If V2c services are required, select SNMP version V2c, and configure the Read and Write community to the browser.

- The V2c agent will respond only to the queries from SNMP MIB browsers using the same community. That is, the V2c agent and the browser should be members of the same community.
- If the community fields are left blank, the manager sends the SNMP request with the community name as "public"
- The V2c agent is configured by default with three Read communities ("public", "read", " ") and three Write communities ("private","write","public")
- The default maximum community length is 8 characters
- As the default communities also contain the "public" community name, the agent will respond to all of the browsers requesting the "public" community
- At run time, the community names can be dynamically configured using the HTTP interface for SNMP community name configuration

If the V2c agent receives an SNMP request with an unknown community name, the agent will generate an Authentication trap.

The V2c agent's multiple community support feature enables the user application to provide limited access to the requesting browser based on the community name used by the browser to access the MIB database variables of the agent.

3. If SNMPv3 services are required, select the SNMP Version as 'V3' in the 'Advanced' tab of the SNMP MIB Browser. The following configuration window appears:



4. If SNMPv3 services are required, the SNMPv3 browser is required to be configured with the user name, authentication and privacy password, message authentication hash type, privacy protocol type. The SNMP server will respond only if one of the user credentials and user security parameters in the following table is configured at the manager. This table is stored in the global structure with the SNMPv3 server stack. The SNMPv3 server would only respond if the request credentials of the MIB browser matches to that of the stored user database of the SNMP server.

	USER 1	**USER 2**	**USER 3**
USM User	microchip	SnmpAdmin	root
Security Level	auth, priv	auth, no priv	no auth, no priv
Auth Algorithm	MD5	SHA1	N/A
Auth Password	auth12345	ChandlerUS	N/A
Privacy Algorithm	AES	N/A	N/A
Privacy Password	priv12345	N/A	N/A

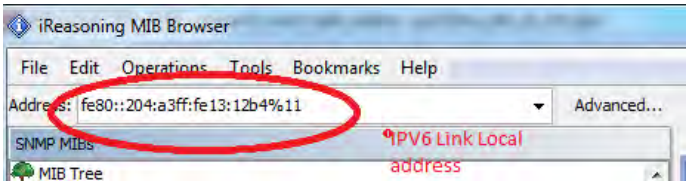
5. The Microchip SNMPv3 stack does support only one Context Engine ID with the server. Leave the "Context Name" option in the "Advanced" tab empty. It is ignored on the server.

6. According to the user and the auth and privacy protocols configured with the SNMP browser, the UDP authenticated and encrypted message would be exchanged between server and the client.

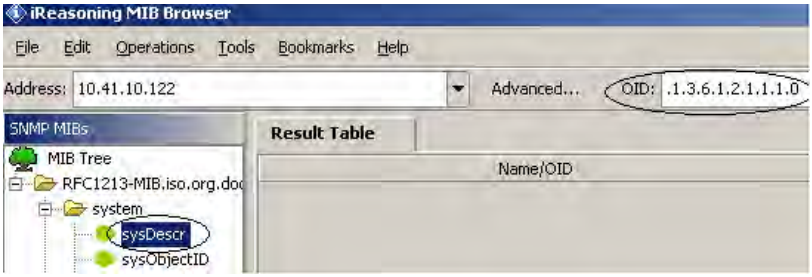
- If the USER 1 values, as shown in the table, are configured in the MIB browser, the data exchange between the client and server is encrypted and authenticated. The PDU can be captured in the Ethernet packet sniffer like WireShark and examined. As the data is encrypted and authenticated, the data integrity and the privacy is achieved.
- If the USER 2 values, as shown in the table, are configured in the MIB browser, the data exchange between the client and server is authenticated. The data integrity will be checked once the data is received at either end. The message authentication mechanism protects from the possible data sniffing and modification threat, and also guarantees that the data is received from the authenticated and guaranteed source.
- If the USER3 values, as shown in the table, are configured in the MIB browser, the data exchange between the client and server is neither authenticated nor encrypted.

Considering the three USER configurations, if the SNMP server is to be accessed over WAN in the Internet cloud, the data should be encrypted and authenticated to have the highest level of data privacy and integrity.

7. Configure the IPv4 or IPv6 address of the SNMP agent to the 'Address field'.



8. Select the variable to be accessed from the agent MIB database from the SNMP MIBs pane. The OID of the selected variable can be seen in the OID tab in the following figure.



9. Select the SNMP Get operation from the operations tab.

10. The SNMPv3 server demonstration MIB is included with RFC1213 SNMPv2 MIB variables, private MIB variables, and the SNMP-FRAMEWORK-MIB variables. If the SNMPv2C request with a validated community name is generated from the MIB Browser, only a limited set of variables is accessed. The access to the MIB variables is restricted to the type of SNMP version request received. If the SNMPv3 request with correct credentials is generated from the MIB Browser, the complete MIB access is provided.

11. The user will need to decide which part of the MIB should be required to be restricted depending upon the SNMP version type. The MIB design is the one of the important steps in deciding the MIB tree structure and the variable to be placed accordingly.

12. The SNMP server demonstration MIB is added with a static variable OID named "snmpv3PvtObject" with a OID value as 43.6.1.4.1.17095.6.1. This variable is placed in the private branch of the MIB by creating an independent branch. All of the other variables in the private branch are accessible by a SNMPv2c request. The access to this static variable is restricted by the SNMP version type. Only the SNMPv3 request with correct credentials can access this variable.

SNMP Operations

This topic describes the SNMP operations that can be used with the TCP/IP SNMP demonstrations.

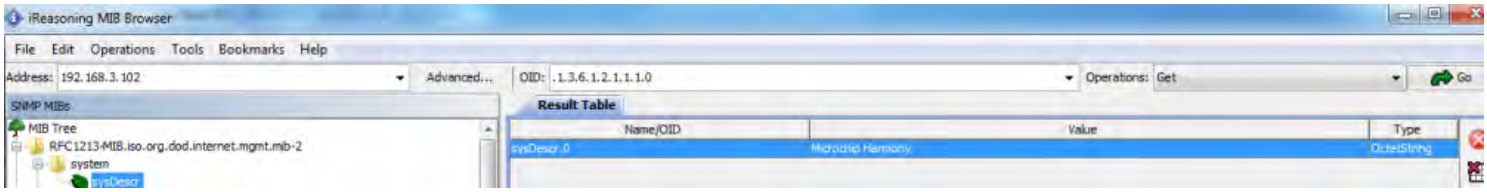
Get Operation

Get operation description.

Description

1. Select the "Advanced" tab and configure the SNMP version to '1' and the Read community to "public".
2. Select "Get" from the operations menu.
3. Select the sysDescr variable from the MIB Tree.

The Result Table displays the sysDescr variable information. Repeat this procedure for any MIB variable. For SNMP V2c, repeat the same procedure, substituting '2' in place of '1' in the version configuration.



As explained earlier, the V2c agent is configured with three Read and Write community defaults. Configure the browser to use any of these communities and try accessing the MIB variables. You should be able to access some of the MIB variables even with the Read Community configured as any of the 'write' community defaults. For Get operations, if the Read or Write community matches, the agent processes the request.

For Set operations, the received community names must match any of the 'write' community names.

For SNMP V3, substitute '3' in place of '1' in the version configuration in the "Advanced" tab. Configure the other user based auth and priv credentials as explained in the "MIB Browsers" section.

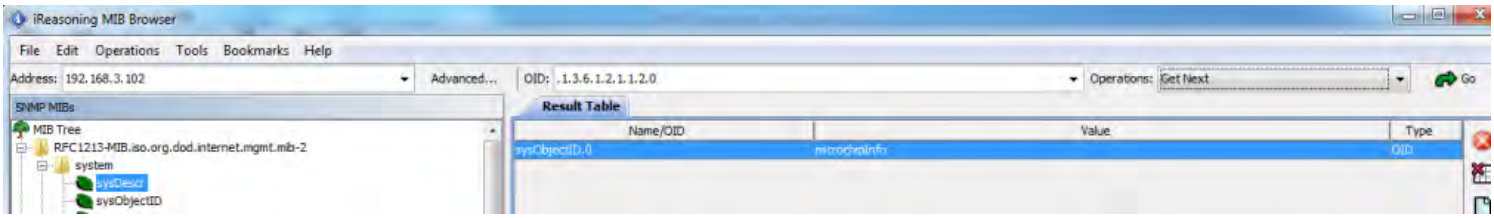
With appropriate credentials, all the MIB variables are accessible. Select any of the MIB variables in the MIB tree and do a GET operation.

Get_Next Operation

Get_Next operation description.

Description

1. Repeat the process for the Get operation.
2. Select the sysDescr variable from the MIB tree, and then select "Get Next" from the operations menu. The result table will display the sysObjectID variable information.



3. Repeat Steps 1 and 2 for additional MIB variables to get the information for the corresponding next variable.
4. Set the SNMP MIB Browser version to v1/v2c. Try to access the private MIB variable "snmpv3PvtObject" with OID value as 43.6.1.4.1.17095.6.1. The access should be restricted. Set the version to V3, configure the credentials, again try a Get_Next operation for the same variable. The access should be granted.

Get_Bulk Operation

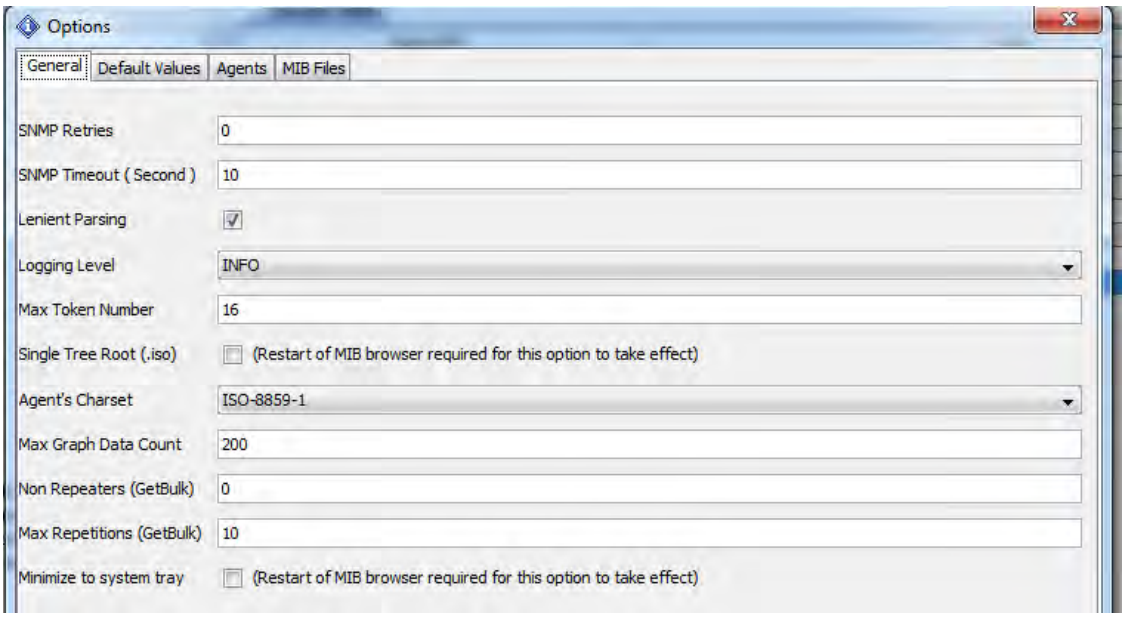
Get_Bulk operation description.

Description

This operation is supported in SNMPv2c and SNMPv3. Get_Bulk enables the collection of bulk information from the agent with a single request from the manager.

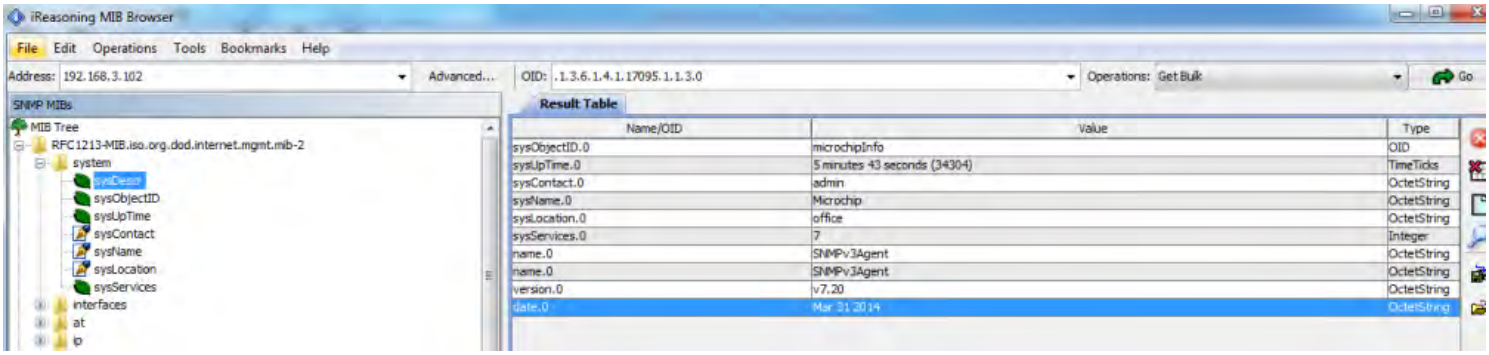
1. Configure the SNMP version to '2' or '3' in the SNMP browser.
2. If version is configured to '2', set the Read Community to 'public' or 'read.'
3. If version is configured to '3', configure the appropriate V3 credentials.
4. Select the sysDescr variable from the MIB tree.
5. Select the Get Bulk operation from the Operations menu.

The default Non Repeaters and Max Repeaters values are '0' and '10', respectively, and get bulk configuration profile to change Non Repeaters and Max Repeaters parameters.



The result table will display information for 10 MIB variables in a single request (if the Max Repetitions = 10 and Non Repeaters = 0 is configured). These variables are the lexicographical successors of the sysDescr variable. The number of variables that the agent will respond with can be

configured in the browser through the menu selections *Tools > Options > Non-Repeaters* and *Tools > Options > Max-Repetitions*. The Non-Repeaters and Max-Repetitions numbers are extracted by the SNMP agent from the received Get_Bulk request and the number of variables that will be included in the response PDU is calculated. for more information on calculating the number of variables, Non-Repeaters, and Max-Repetitions, refer to RFC 3416.



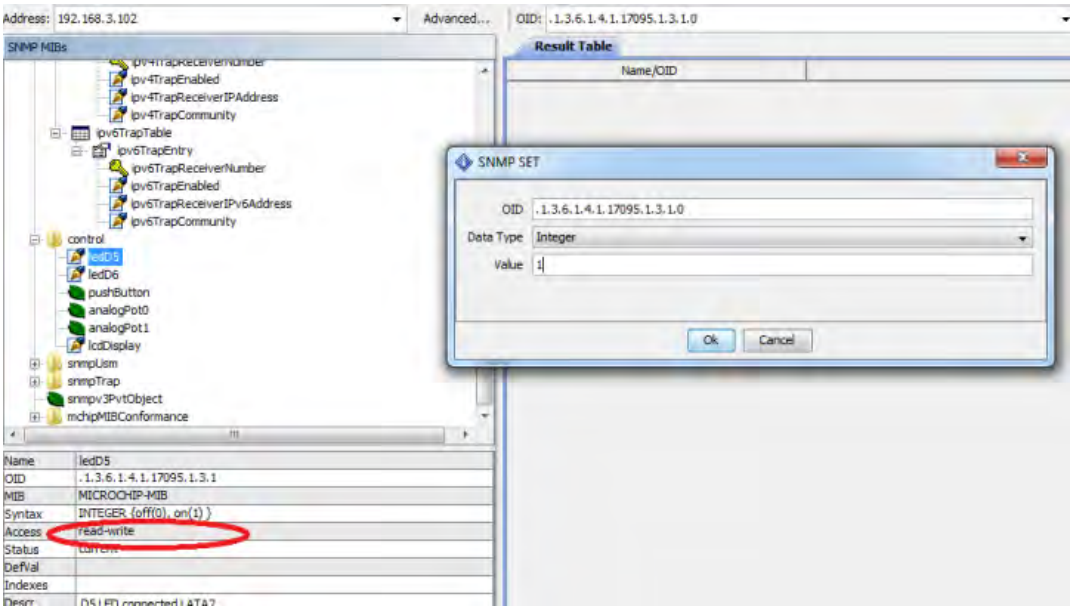
Set Operation

Set operation description.

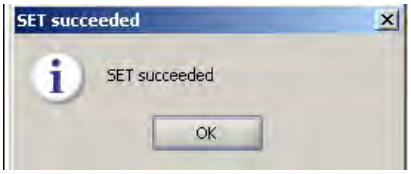
Description

The Set command updates the variable information of the MIB database in the agent. The Set command can be performed only on those variables which are declared as 'READWRITE' in the MIB scripts, and only if the community name matches any one of the 'write' community names configured with the agent.

1. Select the ledD5 variable from the MIB tree.
2. Configure the SNMP version to '1' or '2.' Configure the Write Community to 'public', 'write', or 'private'.
3. If version is configured to '3', configure the appropriate V3 credentials.
4. Select 'Set' from the Operations menu and the SNMP SET window will pop up. Enter the value for the browser in the OID field as per the defined syntax of the `mchip.mib` and `snmp.mib` scripts.



A success message will appear.



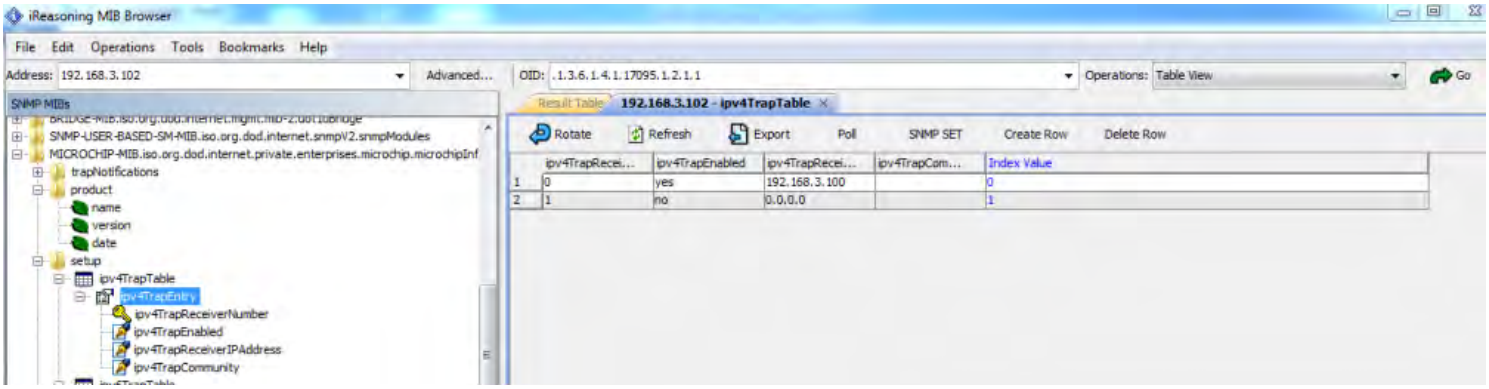
A 'Get' operation for the same variable should now return the new 'Set' value for this variable. LED5 on the demonstration board should now be ON. Repeat the procedure to set LED5 to OFF. LED6 can also be set ON or OFF.

Table View

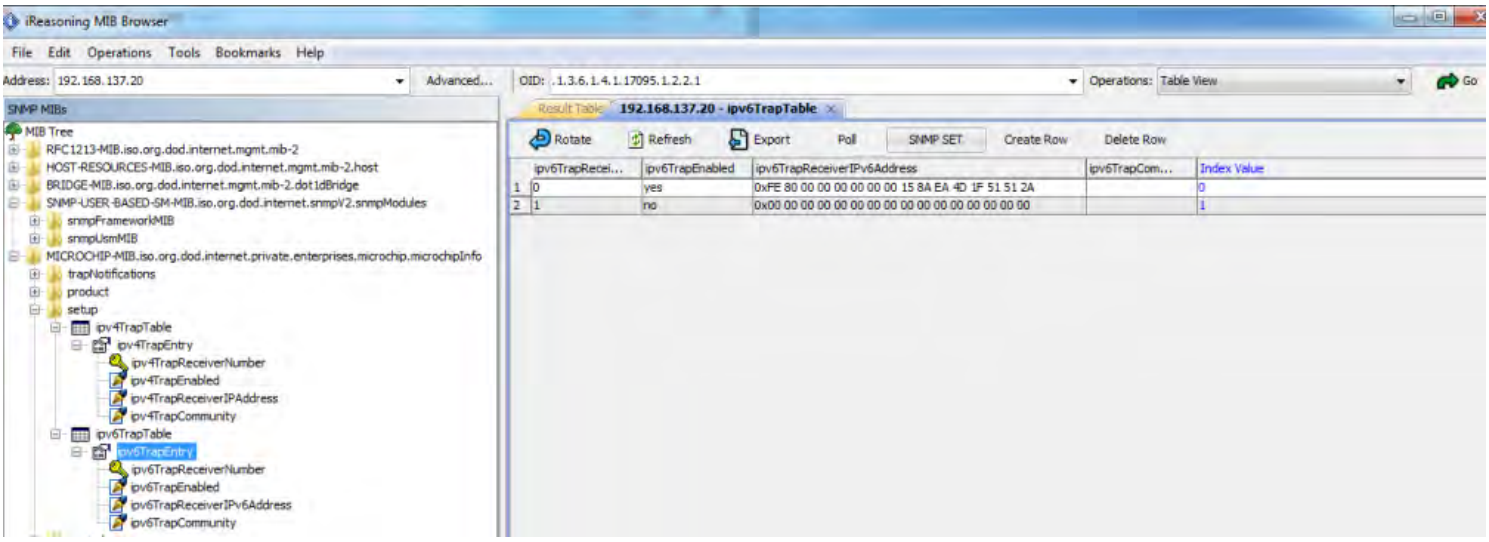
- Like other operations , table view is used for sequence variables
- Create row, delete row is not supported

- Refresh button is used to get the updated tabular values

IPv4TrapTable

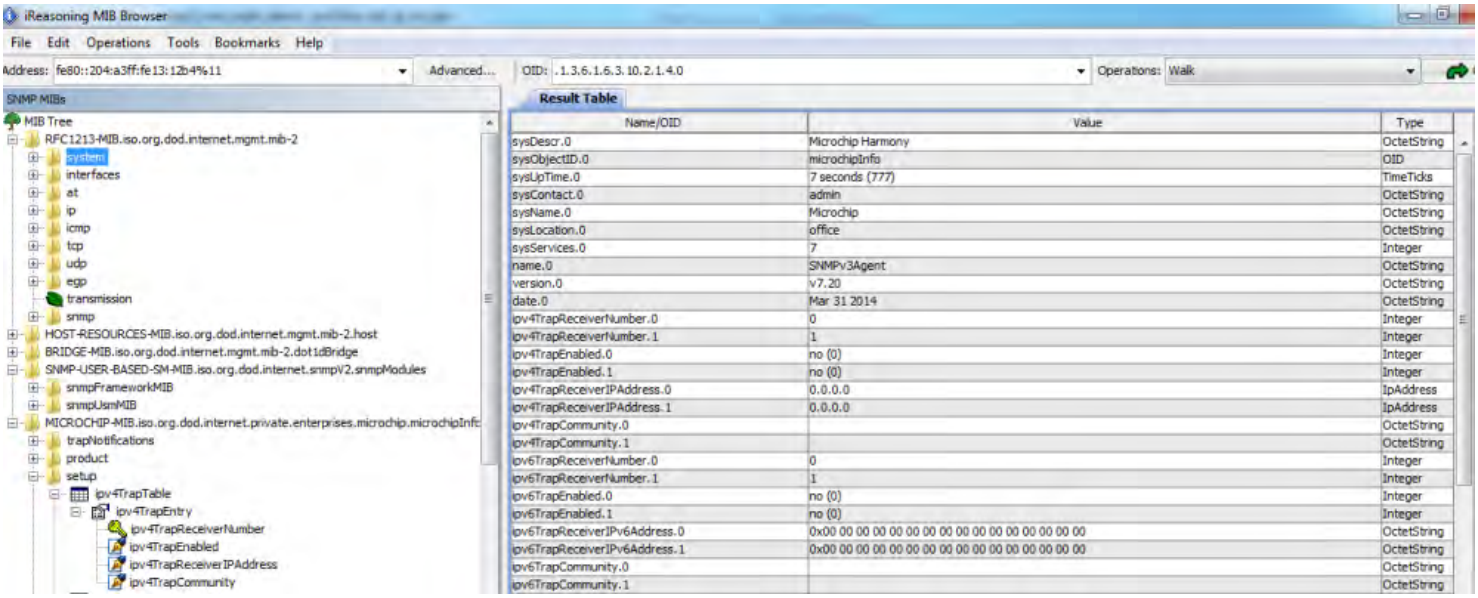


IPv6TrapTable

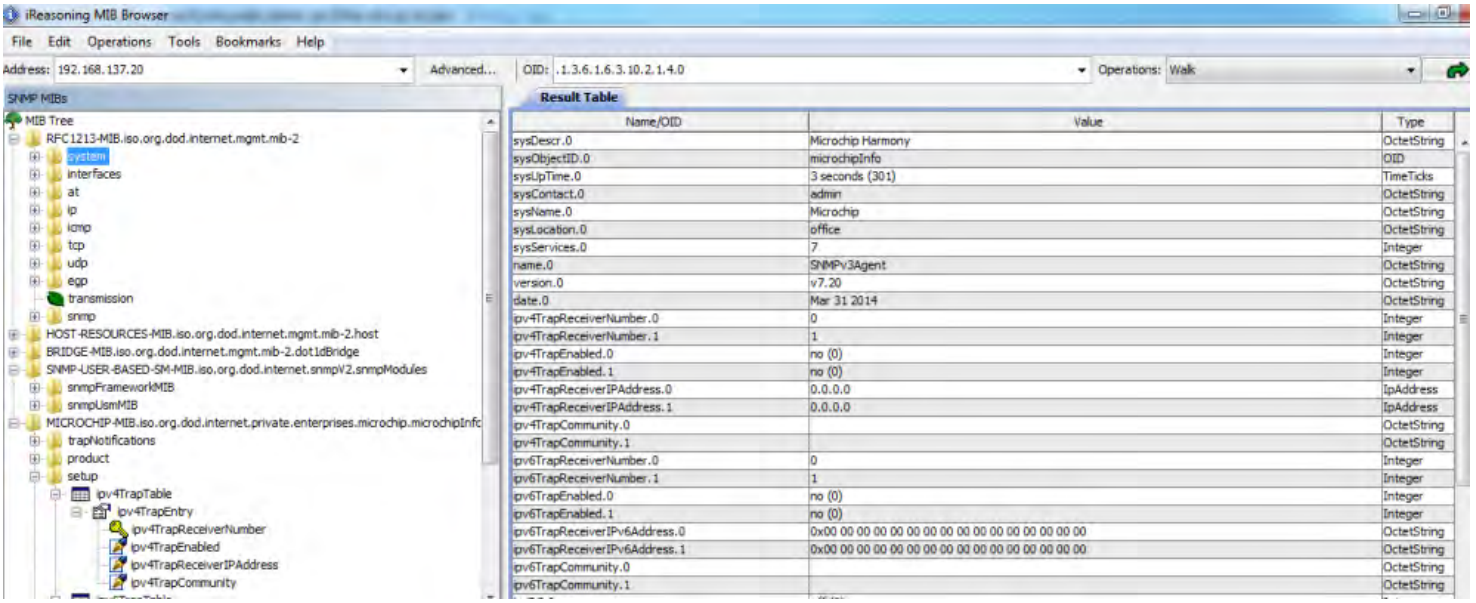


Walk

With IPv4 Address



With IPv6 Address



Trap Test

Trap test description.

Description

Procedure for IPv4 Trap Table Configuration

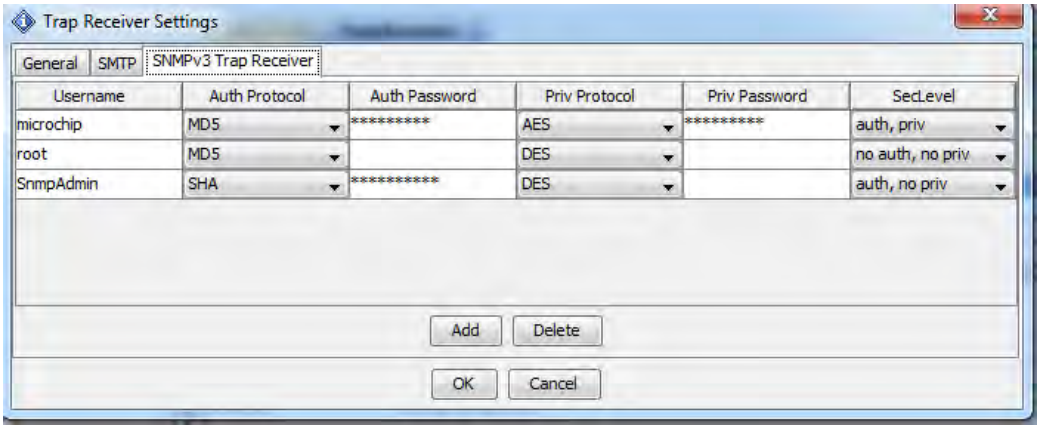
1. Open the 'Advanced' configuration menu, configure the SNMP version to '2,' and configure the Write Community to "public', 'write', or 'private'.
2. Select the 'IPv4trapEnabled.0' variable from the MIB tree.
3. Select 'Set' from the Operations menu.
4. Enter '1' in the value field of the SNMP SET window.
5. Select 'IPv4trapReceiverIPAddress.0' from the MIB tree.
6. Set the value to the IP address of the personal computer on which the SNMP browser is installed and running.
7. Open the "Trap Receiver' utility that was installed with the iREASONING MIB browser (*Start > Programs > iReasoning > MIB Browser > Trap Receiver*).

Procedure for IPv6 Trap Table Configuration

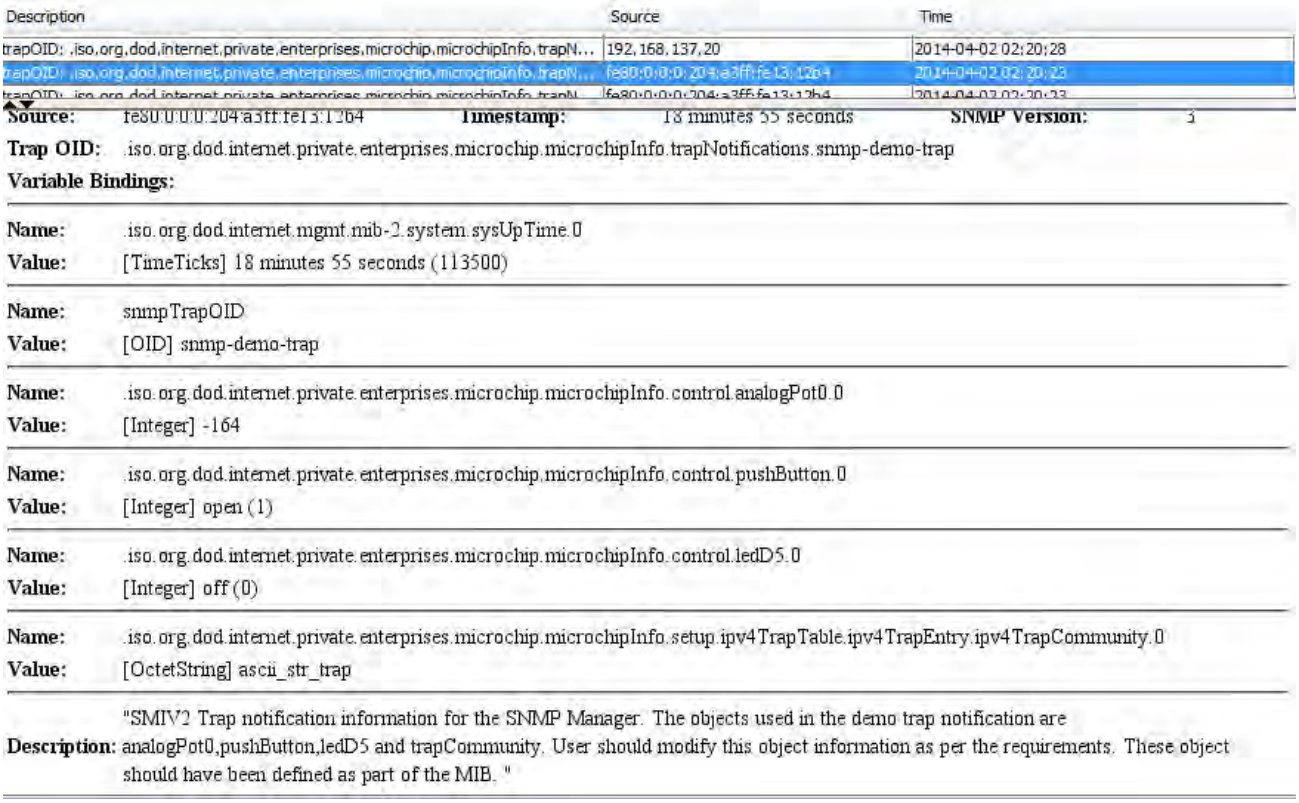
1. Open the 'Advanced' configuration menu, configure the SNMP version to '2,' and configure the Write Community to "public', 'write', or 'private'.
2. Select the 'IPv6trapEnabled.0' variable from the MIB tree.
3. Select 'Set' from the Operations menu.
4. Enter '1' in the value field of the SNMP SET window.
5. Select 'IPv6trapReceiverIPAddress.0' from the MIB tree.
6. Set the value to the IP address of the personal computer on which the SNMP browser is installed and running.
7. Open the "Trap Receiver' utility that was installed with the iREASONING MIB browser (*Start > Programs > iReasoning > MIB Browser > Trap Receiver*).

SNMPv3 Stack Trap Receiver Settings

- iREASONING SNMP version 3 trap receiver receives the traps only with TRAP version 2.
- With respect to iREASONING , need SNMPv3 trap setting to receive traps.
- Open *iReasoning browser > tools > Trap Receiver*
- Open *Trap Receiver >Tools >options >snmpv3TrapReceiver*



Note: The same SNMPv3 user table configuration is required while doing SNMPv3 trap receiver configuration.



HTTP Configuration

HTTP configuration description.

Description

If an HTTP2 server is used with the Microchip TCP/IP Stack, it is possible to dynamically configure the Read and Write community names through the SNMP Configuration web page.

Access the web page using http://mchpboard_e/mpfsupload or http://<Board IP address>(for IPv6 it should be http://<Ipv6 address>:80/index.html), and then access the SNMP Configuration web page through the navigation bar. Use "admin" for the username and "microchip" for the password.


MICROCHIP
MPLAB HARMONY
TCP/IP Stack Demo Application
Overview
Dynamic Variables
Form Processing
Authentication
Cookies
File Uploads
Send E-mail
Dynamic DNS
Network Configuration
SNMP Configuration
SNMP Community Configuration
Read/Write Community String configuration for SNMPv2c Agent.
Configure multiple community names if you want the SNMP agent to respond to the NMS/SNMP manager with different read and write community names. If less than three communities are needed, leave extra fields blank to disable them.
Read Comm1 : public
Read Comm2 : read
Read Comm3 :
Write Comm1: private
Write Comm2: write
Write Comm3: public
Save Config

Micrium uC/OS Libraries Help

This section provides information on the Micriμm® μC/OS-II™ and μC/OS-III™ Libraries.

Introduction

This topic provides an overview of the Micriμm μC/OS-II and μC/OS-III Libraries in MPLAB Harmony.

Description

Micriμm μC/OS-II and μC/OS-III are highly portable, ROMable, scalable, preemptive, real-time, deterministic, multitasking kernels for microprocessors, microcontrollers and DSPs.

More Information

For more information, please read the related documentation, which is available at: https://doc.micrium.com/display/osiiidoc/. Additional information is also available from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

Demonstrations

See RTOS Demonstrations for information.

Application Note

AN1264 "Integrating Microchip Libraries with a Real-Time Operating System" (DS00001264)

Description: This application note examines the reasons for porting to a RTOS-based platform. It then discusses the various changes that may be required to user software to use an RTOS. When discussing this topic, it is easier to do this in the context of a real world application, such as home utility metering, as an example. The demonstration shows how a complex application can be built using Commercial Off-The-Shelf (COTS) hardware and software components. By using an RTOS, the workload involved in integrating multiple libraries has been significantly reduced.

OPENRTOS Library Help

This section provides information on the OPENRTOS® Library.

Introduction

This topic provides an overview of the OPENRTOS® Library in MPLAB Harmony.

Description

OPENRTOS is a small, efficient embedded kernel based on the highly successful FreeRTOS.

License Disclaimer

The OPENRTOS demonstrations provided in MPLAB Harmony use the OPENRTOS evaluation license, which is meant for demonstration purposes only. Customers desiring development and production on OPENRTOS must procure a suitable license. Please refer to one of the following documents, which are located in the `third-party` folder of the MPLAB Harmony installation, for information on obtaining an evaluation license for your device:

- `OpenRTOS Click Thru Eval License PIC32MXxx.pdf`
- `OpenRTOS Click Thru Eval License PIC32MZxx.pdf`

More Information

For more information, please refer to the OPENRTOS documentation, which is available at: http://www.wittenstein-us.com. Additional information is also available from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

SEGGER embOS Library Help

This section provides information on the SEGGER embOS Library.

Introduction

This topic provides an overview of the SEGGER embOS Library in MPLAB Harmony.

Description

SEGGER embOS is a priority-controlled Real-Time Operating System, designed to be used as a foundation for the development of embedded real-time applications.

More Information

For more information, please read the *"embOS CPU-Independent User & Reference Guide"*, which is available at: https://www.segger.com/embos.html. Additional information is also available from the Microchip Third-Party RTOS web page: http://www.microchip.com/devtoolthirdparty/

Demonstrations

See RTOS Demonstrations for information.

Legal Disclaimer

The source code for this SEGGER embOS RTOS demonstration is not freely distributed. To obtain source code and the proper licensing agreement go to the SEGGER embOS website: https://www.segger.com/license-models.html. The SEGGER embOS source has been installed in the following location: `<install-dir>/third_party/rtos/SEGGER` so that the applicable MPLAB Harmony demonstrations can work.

Application Note

AN1264 "Integrating Microchip Libraries with a Real-Time Operating System" (DS00001264)

Description: This application note examines the reasons for porting to a RTOS-based platform. It then discusses the various changes that may be required to user software to use an RTOS. When discussing this topic, it is easier to do this in the context of a real world application, such as home utility metering, as an example. The demonstration shows how a complex application can be built using Commercial Off-The-Shelf (COTS) hardware and software components. By using an RTOS, the workload involved in integrating multiple libraries has been significantly reduced.

SEGGER emWin Graphics Library Help

This section describes the SEGGER emWin® Graphics Library that is available in MPLAB Harmony.

Introduction

Provides information on the SEGGER Microcontroller GmbH & Co. KG emWin software graphics library.

Description

emWin from SEGGER Microcontroller GmbH & Co. KG, is a software graphics library that provides an efficient, processor and LCD controller-independent Graphical User Interface (GUI) for applications that operate with a graphical LCD.

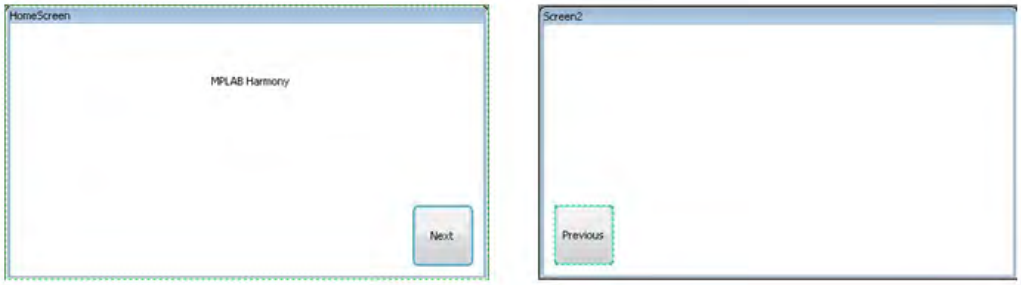
emWin provides a graphical user interface for a graphics application, that is independent of the LCD controller and CPU. Please refer to the SEGGER emWin website for information regarding the architecture and scope of the library by visiting: https://www.segger.com/emwin.html

Getting Started

This section provides getting started information for developing a Graphical User Interface (GUI) application using SEGGER emWin and integrating it with MPLAB Harmony on PIC32 development hardware.

Description

In this demonstration we will create two screens/dialogs. Screens/dialogs are windows that contain one or more widgets. Widgets are elements of the user interface, which react automatically on certain events. Please note that the terms screens and dialogs are used interchangeably in this section. The screen is designed using the GUIBuilder utility. The following figure shows the final screens that will be designed:



The demonstration application will navigate from one screen to another screen using buttons on the screen. The demonstration will use the Next button to navigate from the HomeScreen to Screen2 and the Previous button to navigate from Screen2 to the HomeScreen. The buttons will generate an event on a touch input from the display.

There are other vital SEGGER emWin tools available in the `<install-dir>\utilities\segger\emwin` folder of your MPLAB Harmony installation (see SEGGER emWin Utilities for more information). This demonstration uses only the GUIBuilder utility to provide a glimpse of how to create and run a SEGGER emWin GUI project within MPLAB Harmony. Also, the widgets used here are the simpler ones limiting the steps required to create a GUI.

Hardware Requirements

Provides information on the development hardware used in the demonstration.

Description

The demonstration uses the Microchip PIC32MZ Embedded Connectivity with Floating Point Unit (EF) Starter Kit and the Multimedia Expansion Board II (MEB II).

PIC32MZ EF Starter Kit (DM320007)

The PIC32MZ EF Starter Kit has an on-board 200 MHz PIC32MZ2048EF144 microcontroller with 2 MB Flash, 512 KB RAM and a Floating Point Unit (FPU). It contains an I2C communication peripheral to work with the touch controller. The PIC32MZ EF Starter Kit uses the Low-Cost Controllerless (LCC) driver to drive the the MEB II display without the need of a separate graphics controller.

MEB II (DM320005-2)

The MEB II consists of 4.3” WQVGA PCAP touch display daughter board and optional EBI SRAM memory. The resolution of the display on the 4.3” WQVGA PCAP touch display is 480 x 272 pixels, which also consists of a MTCH6301 Touch Controller.

Software Requirements

Describes the software requirements of the SEGGER emWin application with MPLAB Harmony.

Description

The demonstration described in the Getting Started section uses the following development environment:

- MPLAB X IDE v3.26 or later (http://www.microchip.com/mplab/mplab-x-ide)
- MPLAB XC32 C/C++ Compiler v1.40 or later (http://www.microchip.com/mplab/compilers)
- MPLAB Harmony v1.08 or later (http://www.microchip.com/mplab/mplab-harmony)
- MPLAB Harmony Configurator v1.0.8.6 or later (`install-dir>\utilities\mhc`)
- GUIBuilder v5.32 or later (`install-dir>\utilities\segger\emwin`)

SEGGER emWin GUI Application Design Process

Describes the process to design and run the SEGGER emWin GUI application.

Description

The process to design and run the GUI on the demonstration board includes the following:

- Configuring the Hardware
- Using GUIBuilder to Create Screens/Dialogs
- Creating a New MPLAB Harmony Project
- Loading the GUIBuilder Output into the MPLAB Harmony Project
- Integrating the GUIBuilder Output with MPLAB Harmony
- Build and Program the Application
- Demonstration Output

Configuring the Hardware

Describes how to set up the development hardware for the demonstration.

Description

Setting up the development hardware includes the following four steps:

1. *Jumper Settings:* Short the pins, EBIWE and LCD_PCLK, of Jumper J9 on the MEB II. Shorting the pins will allow the WE pin of the EBI peripheral to be used as the pixel clock of the display for the LCC driver with internal frame buffer configuration.
2. *Power Supply:* Connect the Power supply to the 9V-15V DC Connector on the MEB II.
3. *Connecting starter kit to the MEB II:* Connect the PIC32MZ EF Starter Kit onto the 168-pin Hirose connector on the MEB II.
4. *Connecting the MPLAB REAL ICE:* Connect the MPLAB REAL ICE using the RJ12 connector on the MEB II.

Using GUIBuilder to Create Screens/Dialogs

Provides an example of Screen/Dialog design using the GUIBuilder utility.

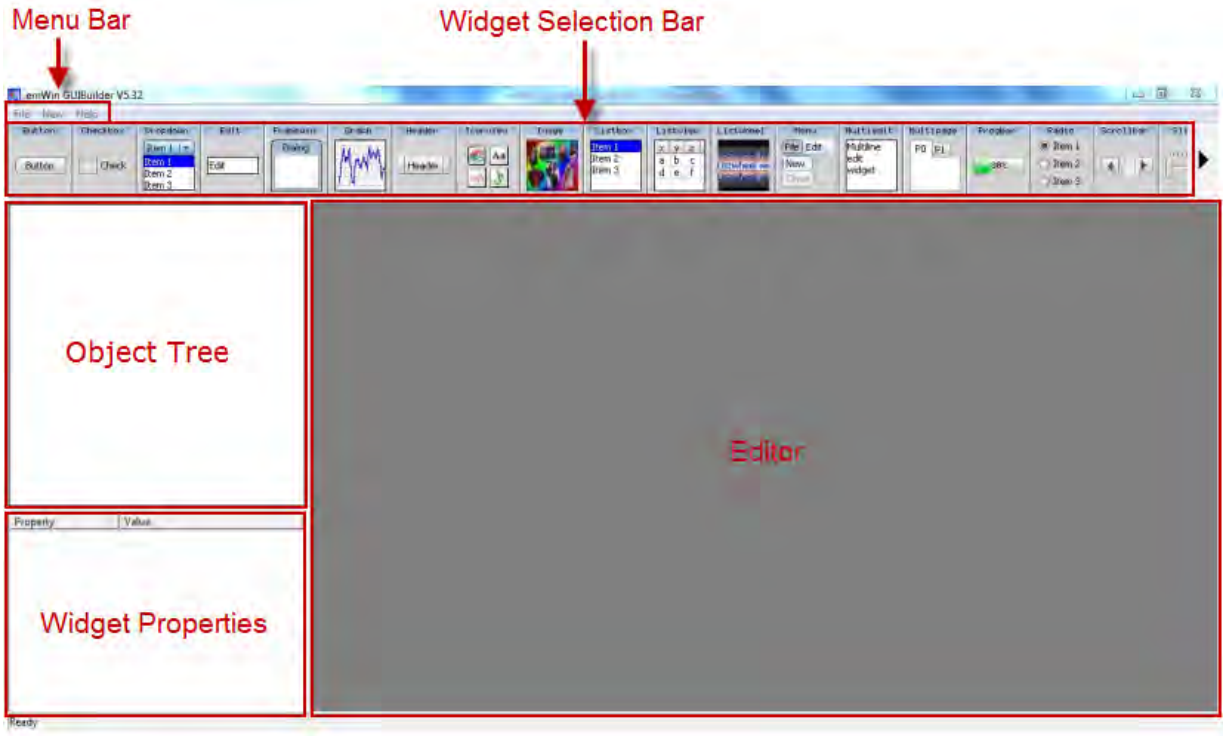
Description

The GUIBuilder utility is a tool for creating dialogs without any knowledge of the C programming language. Instead of writing source code, the widgets can be placed and sized by drag and drop.

GUIBuilder Interface

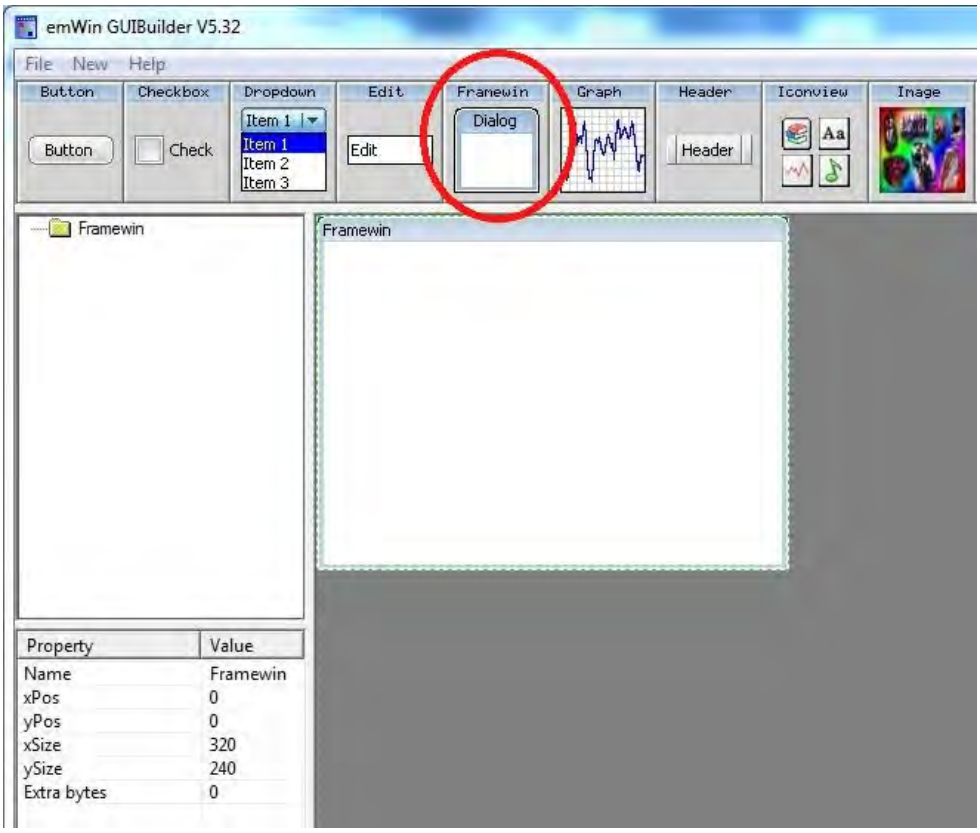
As shown in the following figure, the GUIBuilder Utility window is divided into five areas:

- Menu bar
- Widget selection bar
- Object tree
- Widget properties
- Editor

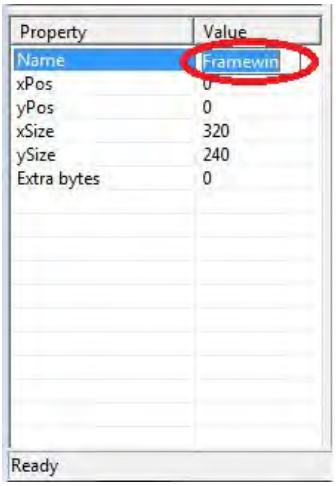


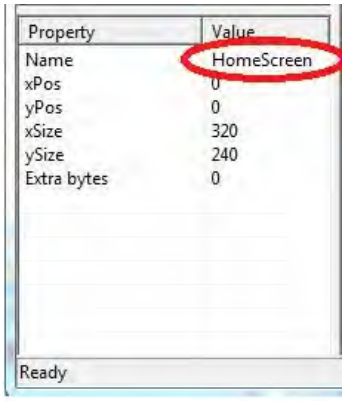
Steps to Design a Screen

1. Start the GUIBuilder utility.
2. To create a screen/dialog, add a parent widget. Add a Framewin widget by clicking **Framewin Dialog** from the widget selection bar. The new widget will appear in the top left corner of the Editor pane, as shown in the following figure.

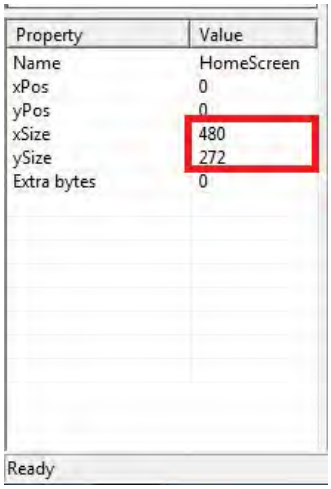


3. Next, we will modify the properties of the Framewin widget. Do the following to edit the name of the widget:
   - Click the cell containing the text “Framewin”
   - Replace the text “Framewin” with “HomeScreen”

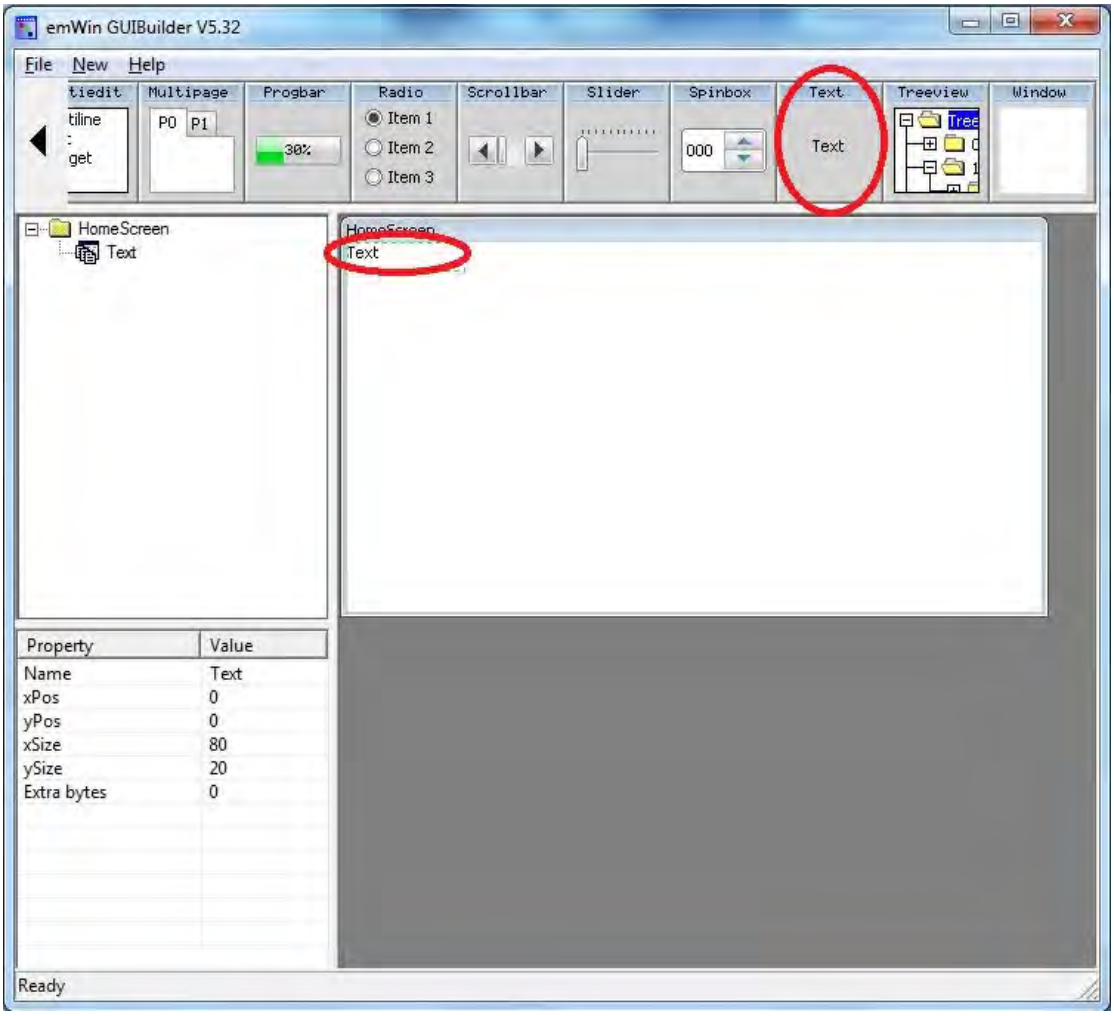




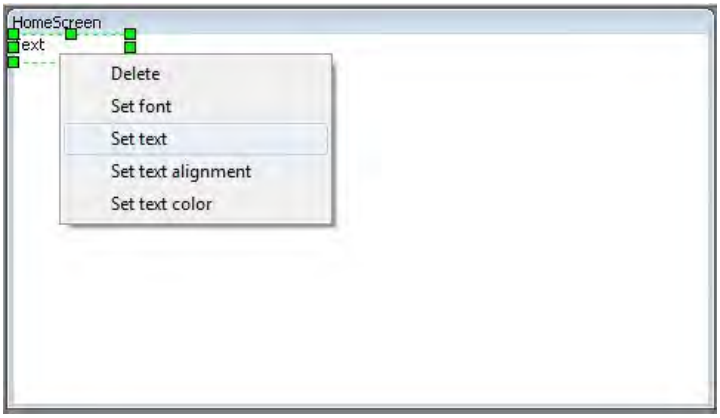
- This screen widget needs to cover the complete display (Resolution 480x272) of the demonstration board. Resize the screen by editing the value of xSize and ySize property of the widget to 480 and 272, respectively.



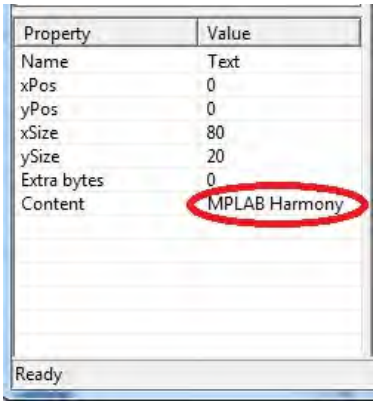
4. Add text to the screen by clicking the **Text** widget from widget selection bar, as shown in the following figure.



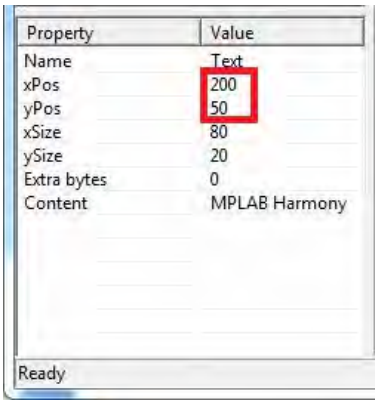
5. Edit the properties of the text, as follows:
   - Right click the Text widget box and select **Set Text**



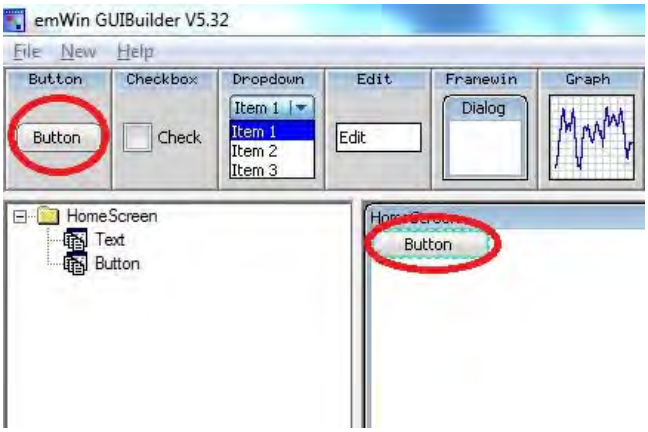
- Click the Content cell and change the value to MPLAB Harmony



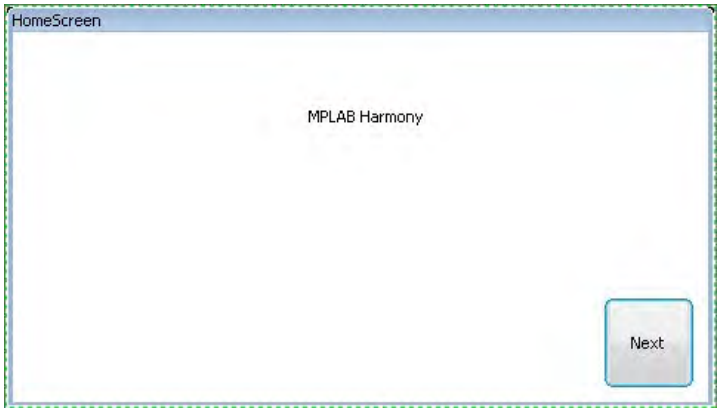
- Change the position of the text widget by setting the value of properties xPos and yPos to 200 and 50, respectively



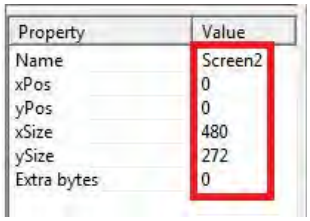
6. From the widget selection bar, add a button widget by clicking **Button**. The Button widget appears in the left top corner of the HomeScreen screen/dialog, as shown in the following figure.



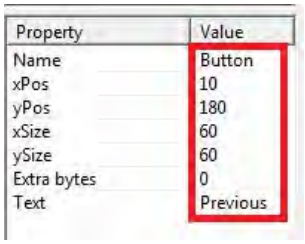
7. Edit the properties of Button widget, as follows:
   - Right click the button from the HomeScreen screen and select **Set Text**
   - Set the properties of the button widget xPos, yPos, xSize, ySize, and Text to 400, 180, 60, 60, and Next, respectively.
8. Save the screen design by selecting *File* > *Save* from the Menu bar. Alternately, you can also use the keyboard shortcut Ctrl + S to save the project. By default, this will save the screen/dialog design as the C file `HomeScreenDLG.c` at same location as GUIBuilder. The following figure shows the designed screen/dialog.



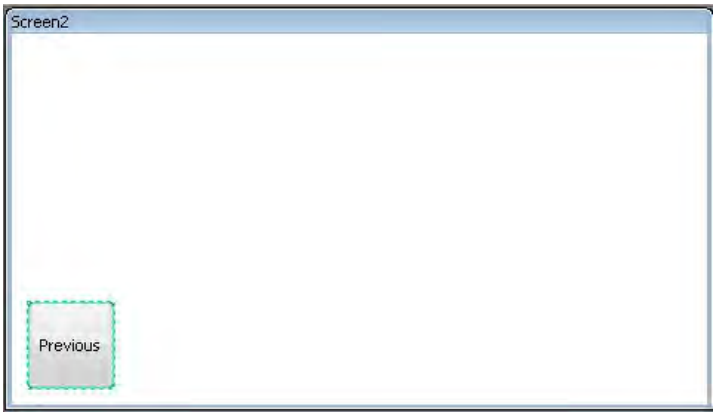
9. Next, create another screen using two widgets: Framewin and Button with the properties of each widget as follows:
   - Framewin widget properties



- Button widget properties



The designed screen will look like the following figure.



By default, saving this screen will generate a file named `Screen2DLG.c` at the same location as GUIBuilder.

Creating a New SEGGER emWin Application Within MPLAB Harmony

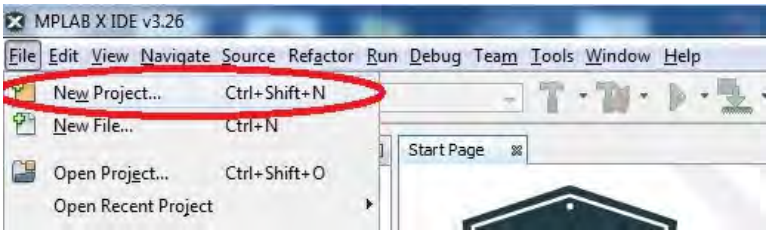
This section describes the process of creating a new SEGGER emWin application within MPLAB Harmony

Description

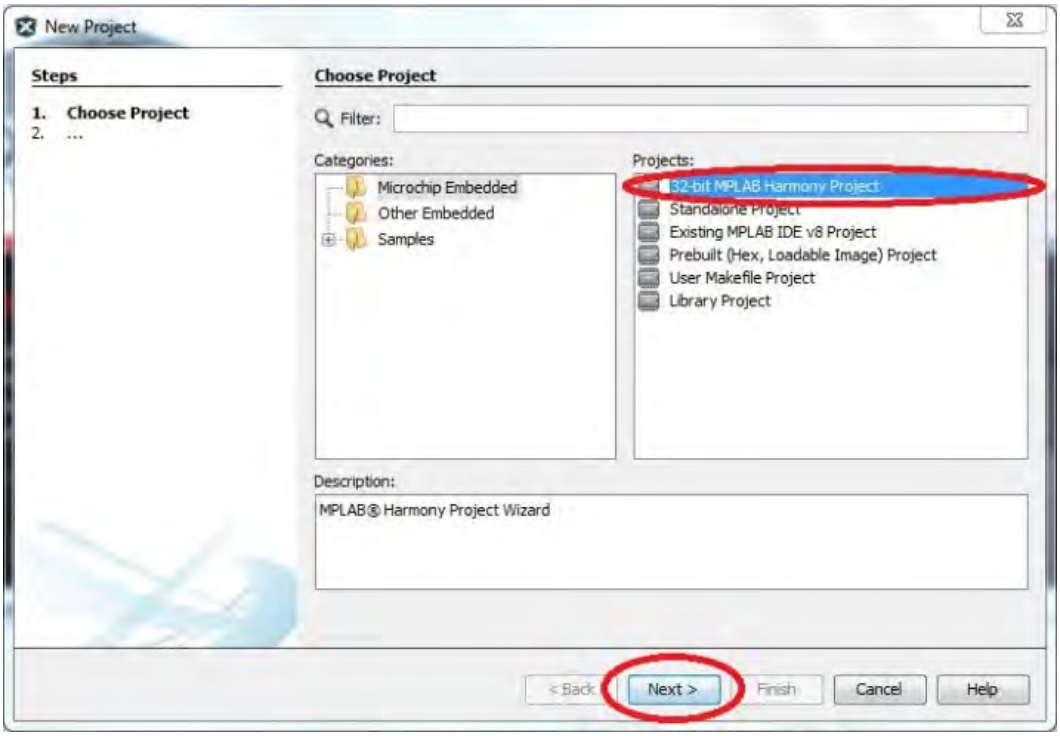
Creating the Application

The following steps describe how to create a new SEGGER emWin Graphics application within MPLAB Harmony. This MPLAB Harmony application uses the screen/dialog C files that were created in GUIBuilder Example.

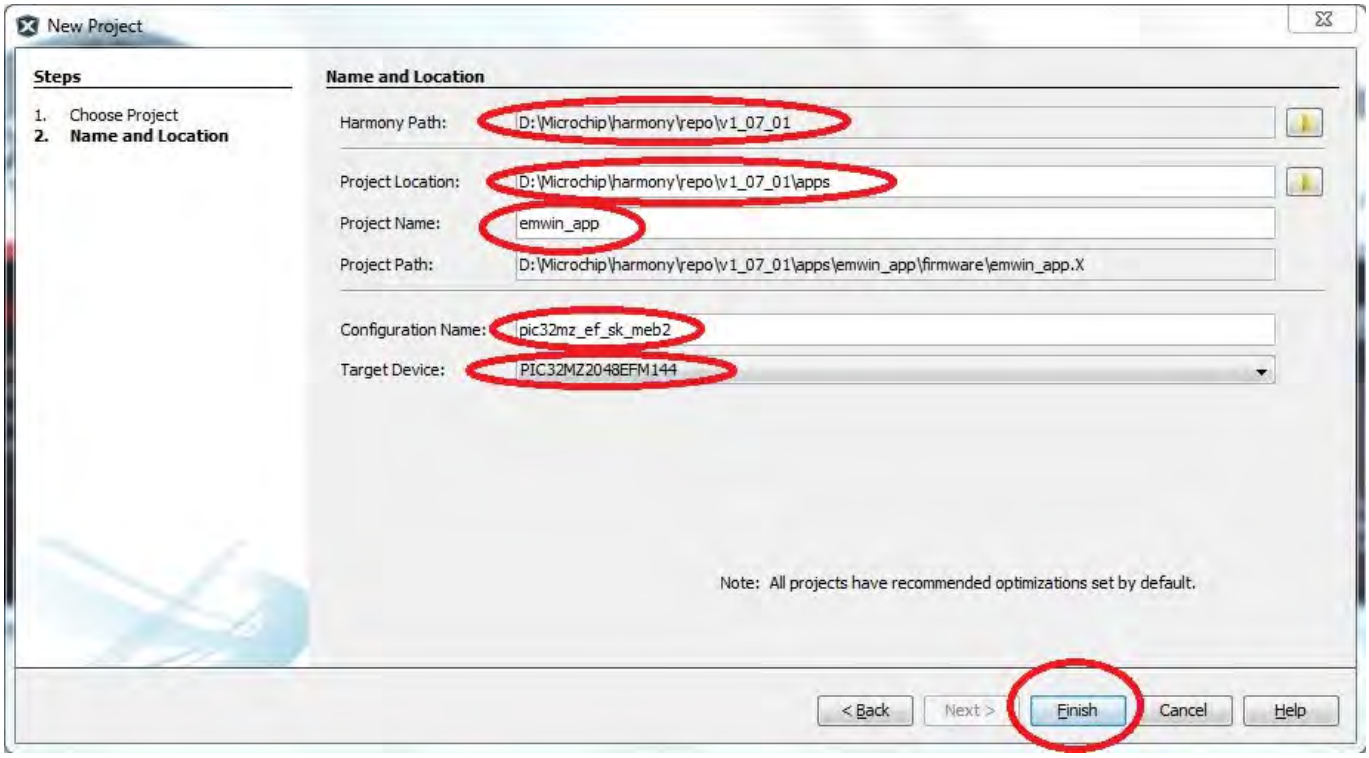
1. Open MPLAB X IDE and create a new project by selecting *File* > *New Project*.



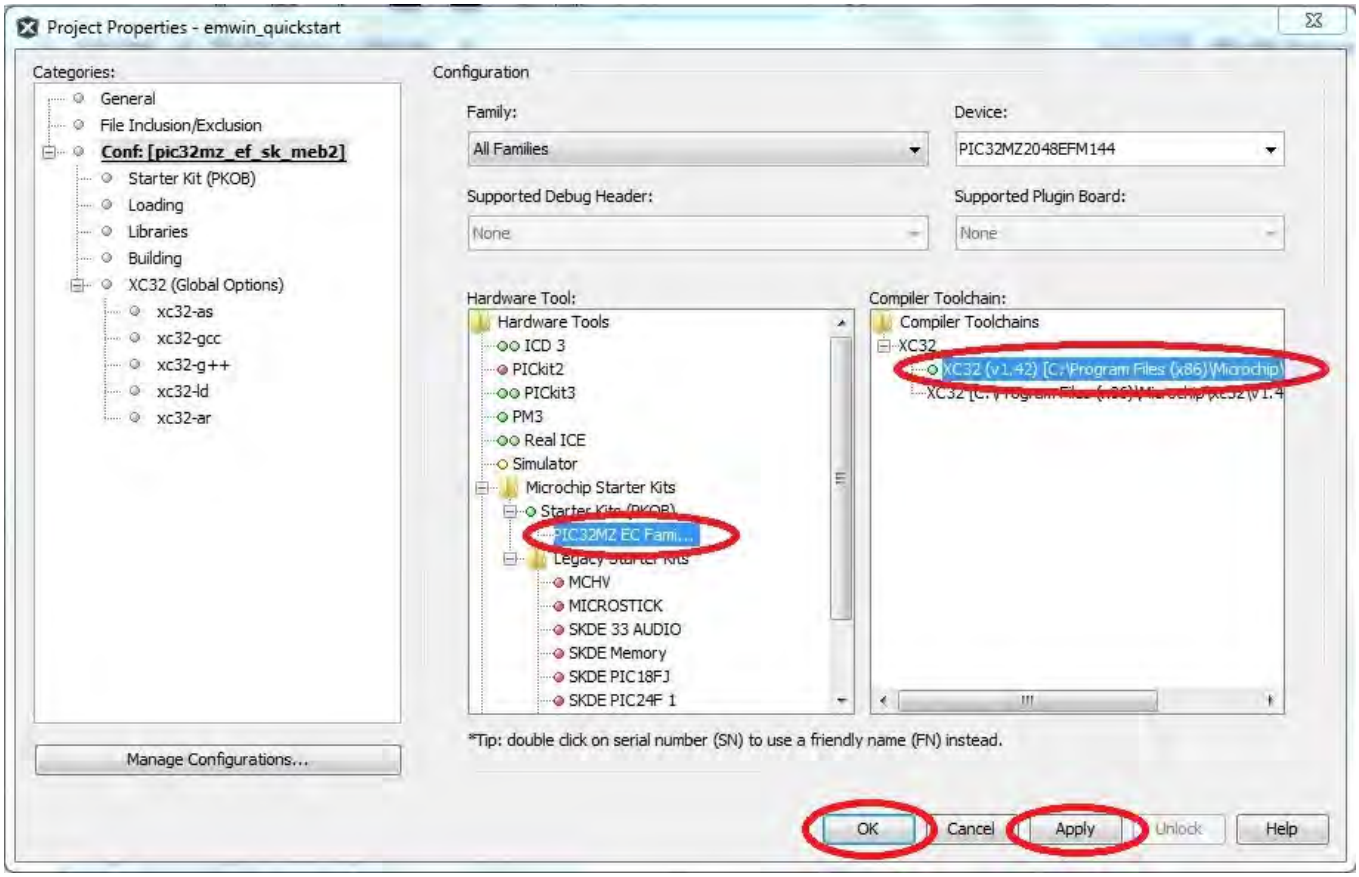
2. Choose the project type **32-bit MPLAB Harmony Project**, and then click **Next**.



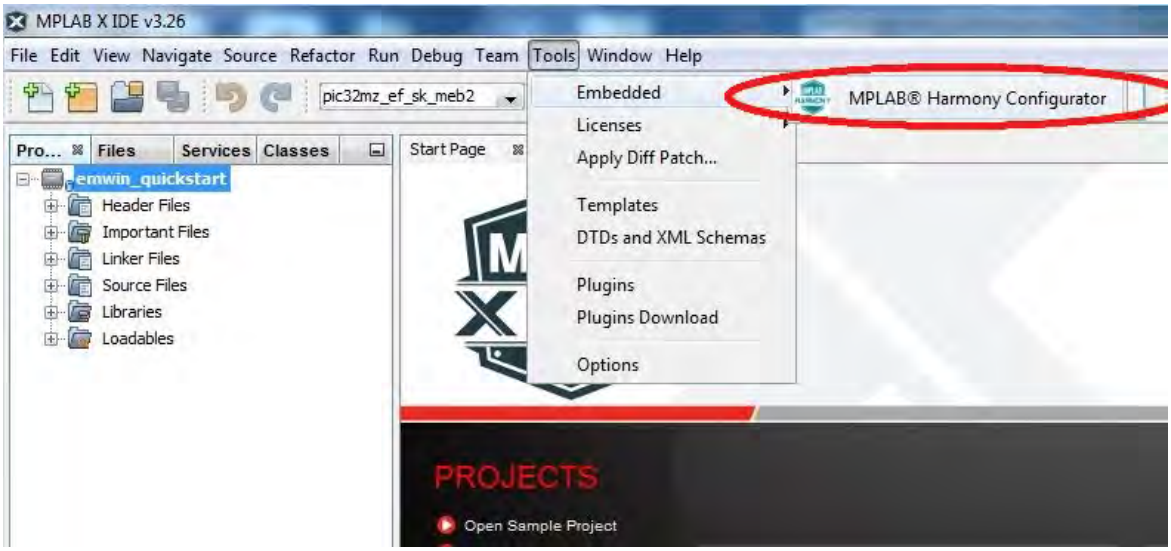
3. Specify the Name and Location project details, as shown in the following figure:
   - Harmony Path: Specify the path to your installation of MPLAB Harmony (i.e., `<install-dir>`)
   - Project Location: `<install-dir>\apps\gfx`
   - Project Name: **emwin_app**
   - Configuration Name: **pic32mz_ef_sk_meb2**
   - Target Device: **PIC32MZ2048EFM144**
   - Click **Finish**



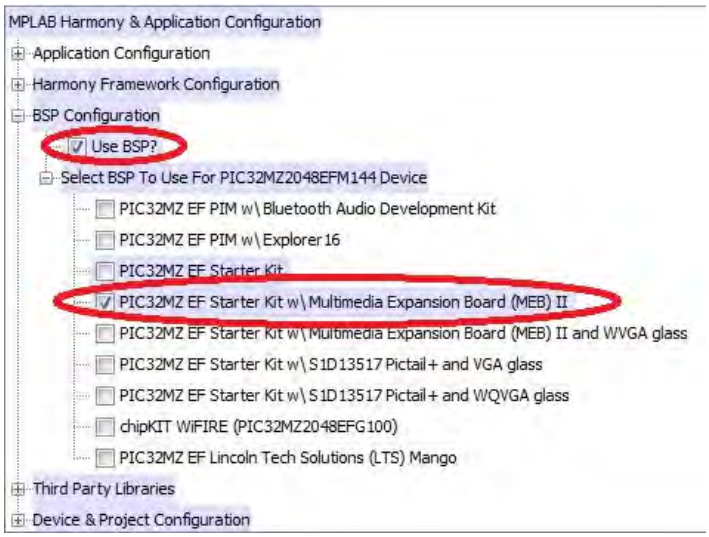
4. Once the new project is created in MPLAB X IDE, right click the project and select **Set as Main Project**. This will apply all future actions to this application in case multiple applications are open.
5. Project properties such as Hardware Tools and Compiler Toolchains need to be set, as shown in the following figure:
   - Right click the project and select **Properties**
   - Select the desired Hardware Tools and Compiler Toolchains
   - Click **Apply**, and then click **OK**



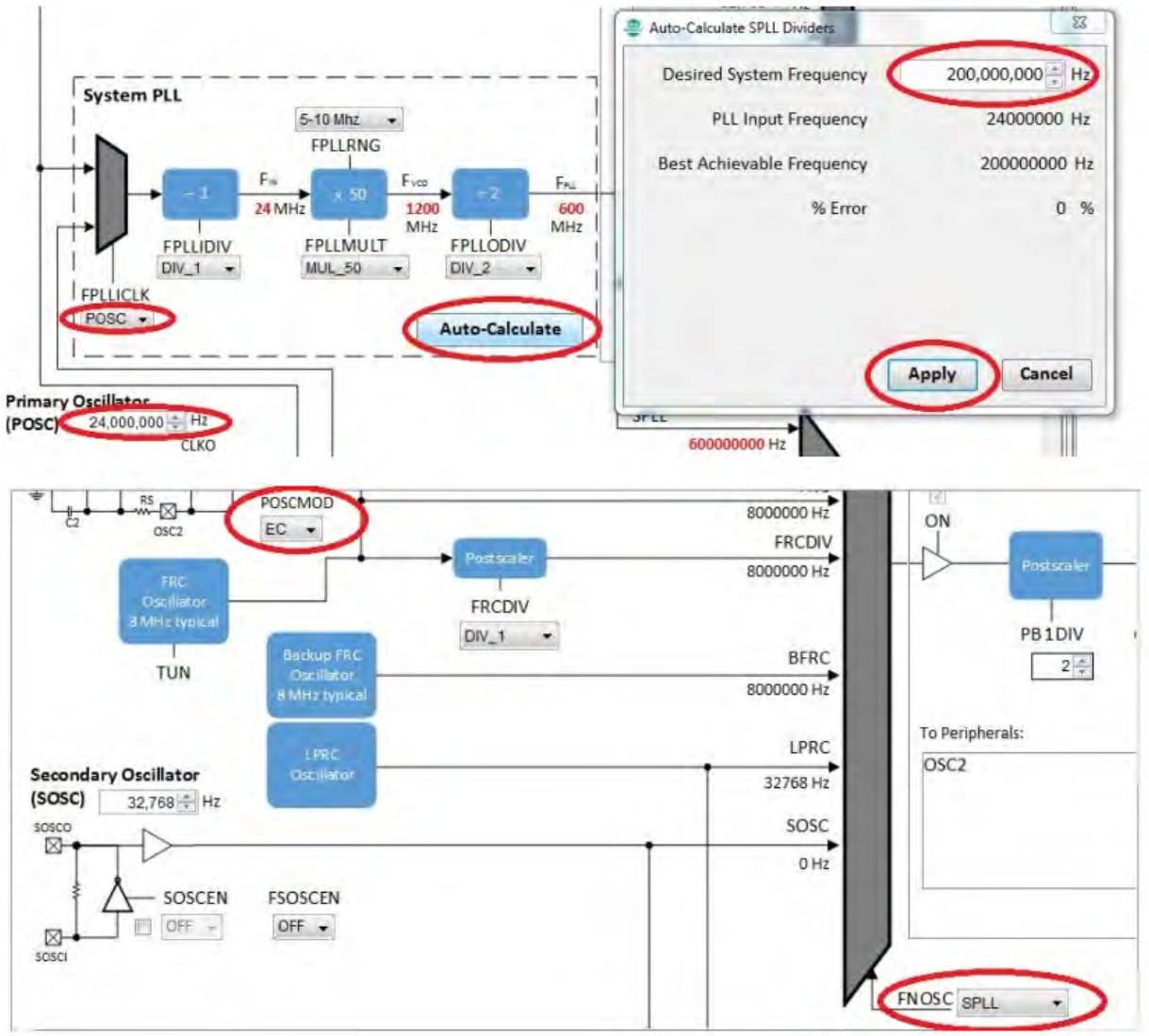
6. Start MHC by selecting *Tools* > *Embedded MPLAB Harmony Configurator*. It is assumed that the MHC plug-in is installed in MPLAB X IDE from the location <install-dir>\utilities\MHC. MHC will run only if at least one of the projects is set as the Main Project.



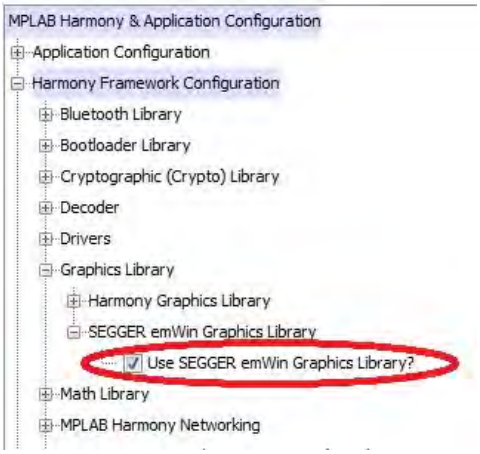
7. In MHC, select the desired Board Support Package (BSP) for the hardware platform, as shown in the following figure:
   - Expand BSP Configuration and select **Use BSP?**
   - Expand Select BSP to Use For PIC32MZ2048EFM144 Device and select **PIC32MZ EF Starter Kit w\ Multimedia Expansion Board (MEB) II**



8. Select the Clock Diagram tab and configure the following clock settings:
   - Primary Oscillator Frequency
   - PLL Clock Source
   - PLL Divider values
   - Primary Oscillator Mode

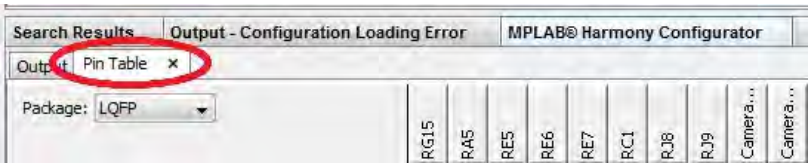


9. From the Options tab expand *Harmony Framework Configuration* > *Graphics Library* > *SEGGER emWin Graphics Library* and select **Use SEGGER emWin Graphics Library?**.

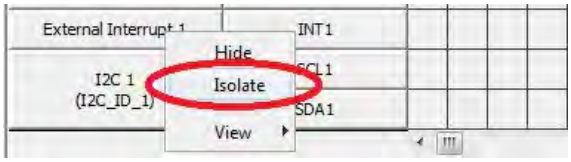


10. Based on the BSP selected, the required drivers are selected for the devices including display timing controller, display controller, Touch driver and Touch Bus (I2C) driver.
11. For the MEB II, the touch device (MTCH6301) reports available touch input by raising an external interrupt. The external interrupt pin needs to be mapped from the list of available pins. In this case, RE8 is the correct pin. To map the RE8 pin as an external interrupt pin do the following:

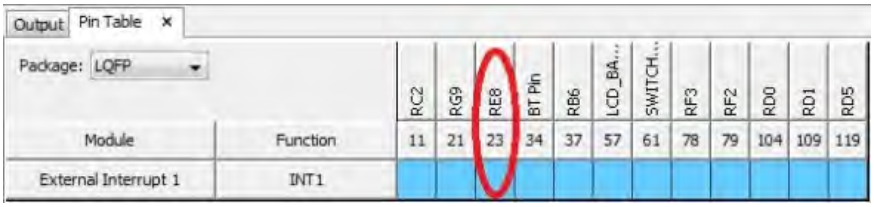
- Select the “Pin Table” tab and scroll down the pin table until the external interrupt tab appears



- Right click the External Interrupt 1 tab and select **Isolate**



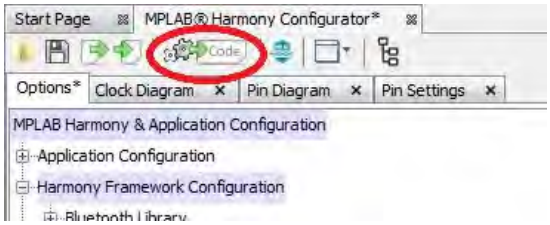
- In the External Interrupt 1 row select the RE8 pin to map it as External Interrupt 1 by clicking the blue block beneath RE8



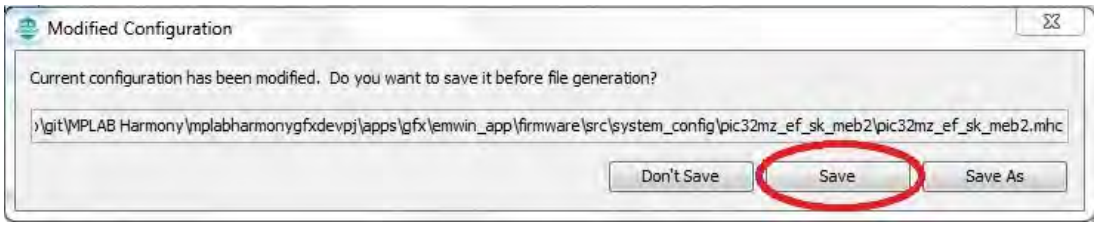
Generate the Code

All of required hardware configuration for our demonstration has been selected. It is now time to generate the demonstration hardware related application code. Use the following steps to generate the code:

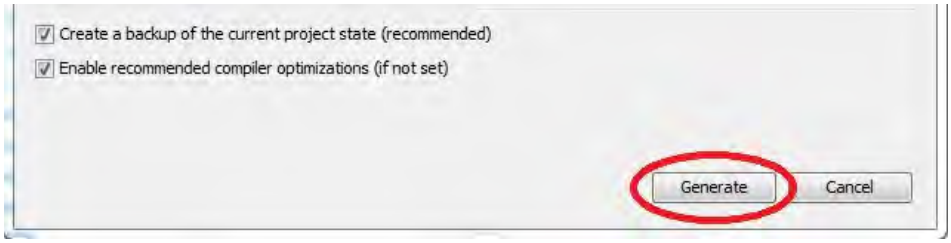
1. In MHC, click **Generate Code**.

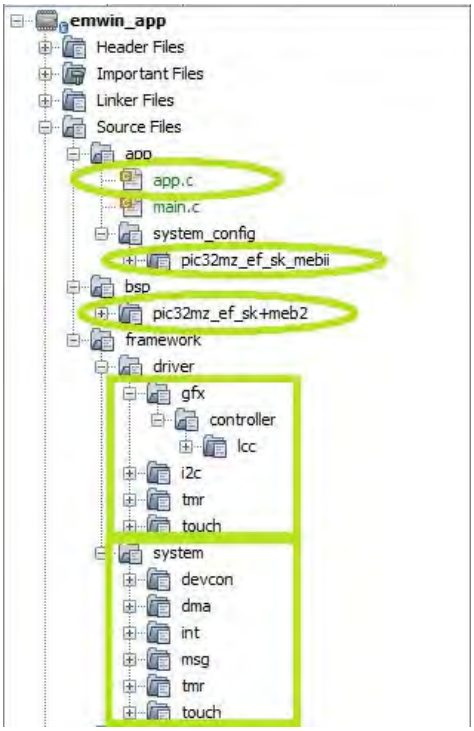


2. Next, in the Modified Configuration dialog, click **Save** to save the modified configuration.

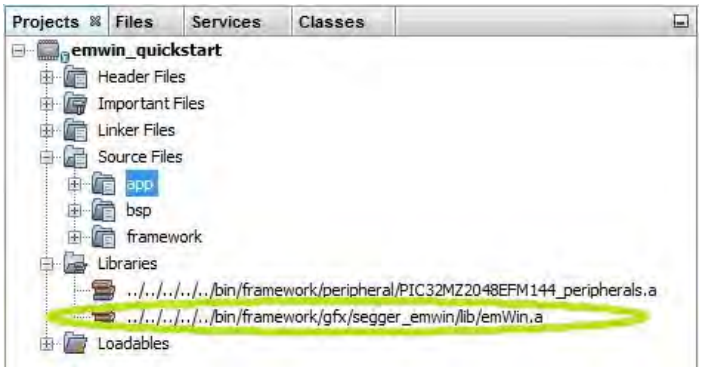


3. Then, in the Generate Project dialog, click **Generate**. This will generate the code for BSP, Drivers, System Services, Configuration, and the application template at the corresponding branch of the application directory tree.





4. MHC will also add the SEGGER emWin Library binary file, `emwin.a`. to the Libraries folder.



Loading the GUIBuilder Output into the MPLAB Harmony Project

This section describes the loading of the GUIBuilder output (C files) into your MPLAB Harmony project.

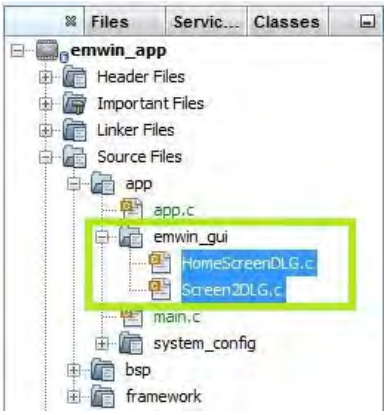
Description

Perform the following steps to load the previously created screens to your MPLAB Harmony project.

1. Create a folder named `emwin_gui` with the project path: `<install-dir>\ apps\gfx\emwin_app\firmware\src`.

2. Copy the GUIBuilder generated files, `HomeScreenDLG.c` and `Screen2DLG.c`, that you created in Using GUIBuilder to Create Screens/Dialogs from the location of the GUIBuilder utility to the path: `<install-dir>\ apps\gfx\emwin_app\firmware\src\emwin_gui.`

3. Create a logical folder named `emwin_gui` with the project tree location `Source Files > app`, as follows:
   - Expand `Source Files`
   - Expand `app`
   - Right click on the logical folder named `app`
   - Click **New Logical Folder**
   - Right click the newly created logical folder
   - Click **Rename…**
   - Replace the default name with `emwin_gui`
   - Click **OK**
4. In MPLAB X IDE, add the GUIBuilder generated `HomeScreenDLG.c` and `Screen2DLG.c` files to the project, as follows:
   - Right click the new logical folder `emwin_gui`
   - Click **Add Existing Item…**
   - Navigate to the folder `<install-dir>\ apps\gfx\emwin_app\firmware\src\emwin_gui` using the Look in: tab from the Select Item window
   - Select the files `HomeScreenDLG.c` and `Screen2DLG.c`
   - Click **Select**



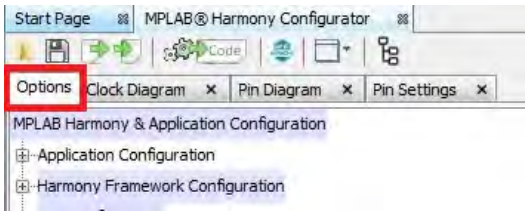
Integrating the GUIBuilder Output with MPLAB Harmony

Describes the integration of SEGGER emWin application code with MPLAB Harmony using the MPLAB Harmony GUI and Touch Wrapper Library.

Description

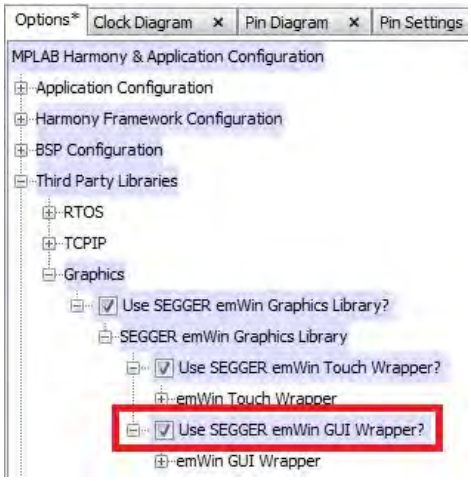
The emWin Wrapper Library consist of wrapper for GUI and Touch. Each wrapper need to be enabled and configured using MHC. Use the following steps to enable and configure the wrapper library:

1. In MHC, select the Options tab.

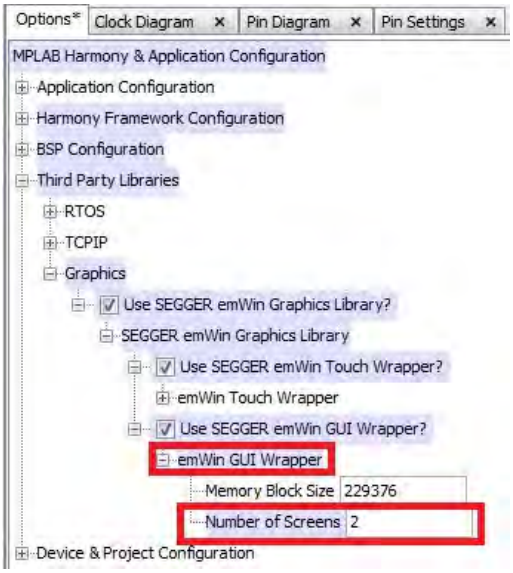


2. Enable the SEGGER emWin Touch Wrapper Library by expanding *Third Party Libraries* > *Graphics* and selecting **Use SEGGER emWin Graphics Library**. Then, expand *SEGGER emWin Graphics Library* and select **Use SEGGER emWin Touch Wrapper?**.

3. Enable the GUI Wrapper Library by selecting **Use SEGGER emWin GUI Wrapper?**.

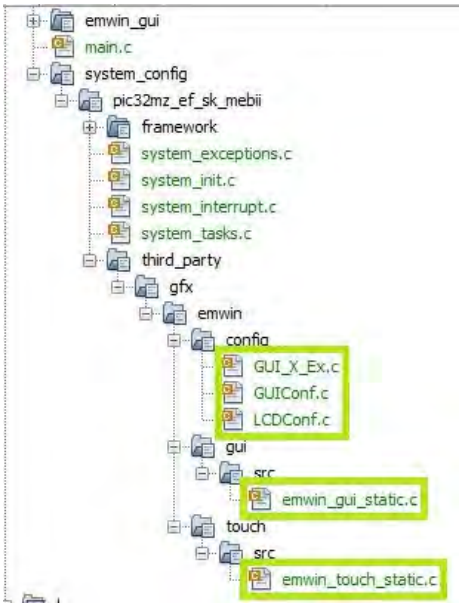


4. Configure the GUI Wrapper Library by expanding emWin GUI Wrapper and setting the Number of Screens to 2.



5. Generate the code by using steps similar to those from Loading the GUIBuilder Output into the MPLAB Harmony Project. The code generation will add the following files:
   - `emwin_gui_static.c`
   - `emwin_touch_static.c`
   - `GUI_X_Ex.c`
   - `GUIConf.c`
   - `GUIConf.h`
   - `LCDConf.c`
   - `LCDCong.h`

6. The generated files will be added to the project at: `Source Files\app\system_config\pic32mz_ef_sk_mebii\third_party\gfx\emwin`.



7. In MPLAB X IDE, edit the `app.c` file from `Source Files\app`, as follows:
   - Add a global variable:

```
/*
    Add create screens APIs from HomeScreenDLG.c and Screen2DLG.c to
    array of screens
*/
EMWIN_GUI_SCREEN_CREATE emWinScreenCreate [ EMWIN_GUI_NUM_SCREENS ]
                                        = { CreateHomeScreen,
                                            CreateScreen2 };
```

- Add a local function:

```
/* Screen Initialize function */
void APP_ScreenInitialize ( void )
{
    /* Set the Background to black */
    GUI_SetBkColor( GUI_BLACK );

    /* Clear the screen */
    GUI_Clear();
}
```

- Add initialization code to the APP_Initialize function:

```
void APP_Initialize ( void )
{
    /* Place the App state machine in its initial state. */
    appData.state = APP_STATE_INIT;
    int32_t screenCount = 0;

    /* Create MailBox for Touch Input */
    emWin_TouchMailBoxCreate();

    /* Register Screen Initialization */
    emWin_GuiScreenInitializeRegister( APP_ScreenInitialize );

    /* Register Screens from screen array */
    for( screenCount = 0; screenCount < EMWIN_GUI_NUM_SCREENS; screenCount++ )
    {
        emWin_GuiScreenRegister( screenCount,
emWinScreenCreate[screenCount]);
    }

    /* set the start screen to 0 */
    emWin_GuiStartScreenSet ( 0 );
}
```

8. Add the following code to `app.h`, which is located in `Header Files\app`:

```
// *****************************************************************************
// *****************************************************************************
// Section: Application Callback Routines
// *****************************************************************************
// *****************************************************************************
WM_HWIN CreateHomeScreen(void);
WM_HWIN CreateScreen2(void);
```

9. Make the following additions to `HomeScreenDLG.c`, which is located in `Source Files\app\emwin_gui`:
   - Add the following code with additional includes:

```
// USER START (Optionally insert additional includes)
#include "system_config.h"
#include "system_definitions.h"
// USER END
```

- Add the following code within case WM_NOTIFICATION_CLICKED for the ID_BUTTON_0:

```
emWin_GuiScreenChange(1);
```

10. The updated code will appear as follows:

```
switch(Id) {
    case ID_BUTTON_0: // Notifications sent by 'Button'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        emWin_GuiScreenChange(1);
        // USER END
        break;
```

11. Make the following additions to `Screen2DLG.c`, which is located in `Source Files\app\emwin_gui`:
   - Add the following code within additional includes:

```
// USER START (Optionally insert additional includes)
#include "system_config.h"
#include "system_definitions.h"
// USER END
```

- Add the following code within case WM_NOTIFICATION_CLICKED for the ID_BUTTON_0:

```
emWin_GuiScreenChange(0);
```

12. The updated code will appear as follows:

```
switch(Id) {
    case ID_BUTTON_0: // Notifications sent by 'Button'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        emWin_GuiScreenChange(1);
        // USER END
        break;
```

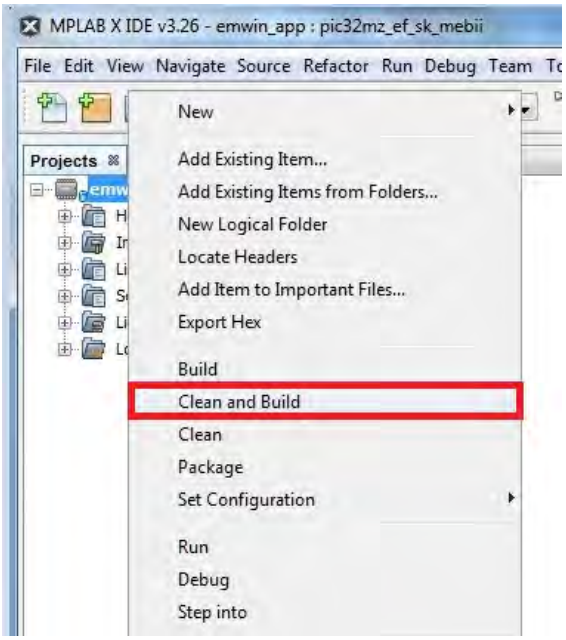
Build and Program the Application

Describes the process to build the application and program the application binary to the development hardware.

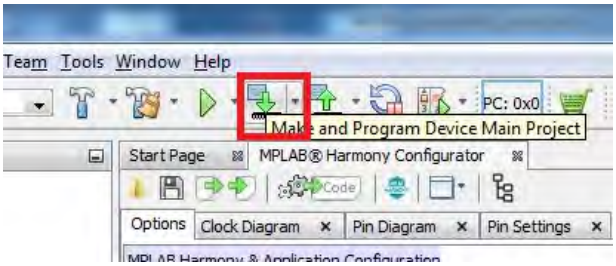
Description

Use the following steps to build and run the application on the development board:

1. To build the project, in MPLAB X IDE, right click the project and click **Clean and Build**.



2. Once the project is successfully built it needs to be run on the target device by programming the target device with the compiled binary file. Click **Make and Program Device Main Project**.



Demonstration Output

Describes the output of the demonstration from the development hardware.

Description

Once the device is programmed with the application binary, the output can be observed on the WQVGA PCAP Display board from the MEB II. The demonstration will display the HomeScreen dialog at the beginning.

To Navigate from the HomeScreen to Screen2 press the Next button. Similarly, to navigate from Screen2 to the HomeScreen, press the Previous button.

Using the Library

This topic describes the basic architecture of the SEGGER emWin Graphics Library and provides information and examples on how to use it.

Description

Library File:

The SEGGER emWin Graphics Library archive file, `emWin.a`, file is installed with MPLAB Harmony in the `<install-dir>/bin/framework/gfx/segger_emwin/lib` directory.

Please refer to the What is MPLAB Harmony? for information on how the SEGGER emWin Graphics Library interacts with the MPLAB Harmony Integrated Software Framework.

Abstraction Model

This library provides an abstraction of the SEGGER emWin Graphics Library.

Description

The SEGGER emWin Graphics Library interface defines a superset abstraction of the functionality provided by any specific implementation or configuration of the library. This topic describes how that abstraction is modeled in software.

emWin operates between the user application program and the hardware layer (i.e., BSP). The following layers are included in emWin:

Widget Layer

This layer includes the *optional* Widget Library.

Window Layer

This layer includes the *optional* Window Manager.

Rendering Layer

This layer includes the *required* GUI Core, which consists of the Graphic Library, Basic Fonts, and Touch/Mouse support.

Output Layer

This layer includes the *optional* memory devices and VNC server, as well as the *required* driver.

Refer to the SEGGER emWin website for more information: https://www.segger.com/emwin.html

Library Overview

Provides an overview of the library.

Description

The SEGGER emWin Graphics Library is provided in binary format; therefore, no API source is included in the installation of MPLAB Harmony.

As part of MPLAB Harmony, the SEGGER emWin Graphics Library now includes the v5.32 emWin header files and the `emWin.a` library, as well as utility tools from SEGGER for graphics application development.

The header files, which are SEGGER originals with only the license in the header modified to reflect Microchip's contract with SEGGER, are installed in: `<install-dir>\bin\framework\gfx\segger_emwin\inc`.

The `emWin.a` file for the library is located in: `<install-dir>\bin\framework\gfx\segger_emwin\lib`.

The latest tools for software development utilizing the library are available in: `<install-dir>\utilities\segger\emwin`.

Refer to Library Interface for information on obtaining documentation that describes the APIs included with SEGGER emWin.

How the Library Works

This topic describes the basic architecture of the SEGGER emWin Graphics Library and provides information and examples on its use.

Description

In addition to the display driver, the SEGGER emWin Graphics Library consists of basic and optional components, as follows:

emWin Basic Components

- Graphic Library
- Basic Fonts
- Simulation Library
- emWinView
- Bitmap Converter
- Color Conversion
- Touch/Mouse Support

Optional Components

- Window Manager
- Memory Devices
- Anti-aliasing
- VNC Server
- Font Converter
- Multi-layer/Multi-display

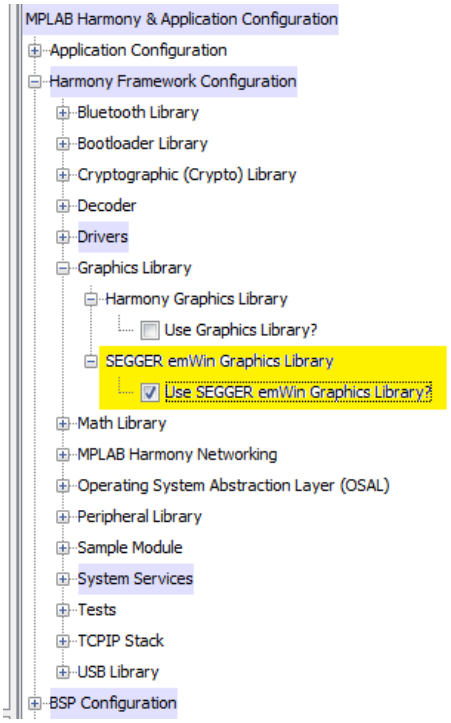
Refer to the SEGGER emWin website for more information: https://www.segger.com/emwin.html

Setup (Initialization)

This topic provides information on setup/initialization.

Description

The SEGGER emWin Graphics Library can be selected in any MPLAB Harmony project by selecting the SEGGER emWin option in the MPLAB Harmony Configurator (MHC) through *MPLAB Harmony & Application Configuration > Harmony Framework Configuration > Graphics Library > SEGGER emWin Graphics Library > Use SEGGER emWin Graphics Library*, as shown in the following figure. This option includes the `emWin.a` file within the Libraries tab of the application project.



Configuring the Library

The SEGGER emWin Graphics Library includes the `system_config.h` file. This file is generated by the MPLAB Harmony Configurator (MHC). It defines the user-selected configuration options necessary to build the library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Library Interface

This section provides information on the Application Programming Interface (API) functions provided in the SEGGER emWin Library.

Description

The SEGGER emWin Library is provided in binary format; therefore, no API source is provided in the installation of MPLAB Harmony. For information on the APIs in SEGGER emWin, refer to the latest *"emWin Graphic Library with Graphical User Interface User & Reference Guide"*, which is available for download from SEGGER by visiting: www.segger.com.

Once on the main site, download the document, by clicking **Downloads,** and then selecting **emWin**. In *Manuals and software*, select the link for the *emWin Manual*.

At the time of this release of MPLAB Harmony, the manual revision was v5.30, Rev.0:
https://www.segger.com/admin/uploads/productDocs/UM03001_emWin5.pdf

GUI and Touch Wrapper Library for SEGGER emWin

This section provides information on the GUI and Touch Wrapper for SEGGER emWin.

Introduction

This topic provides an introduction to the GUI and Touch wrapper for MPLAB Harmony compatibility with SEGGER emWin.

Description

The GUI and Touch Wrapper Library provides a wrapper to SEGGER emWin application code for the purpose of integrating the application code with MPLAB Harmony. The GUI and Touch Wrapper Library provides separate wrappers for GUI and Touch interface for the SEGGER emWin Graphics Library.

GUI Wrapper

The GUI Wrapper can be used for integrating the dialog code generated by the emWin GUIBuilder utility with MPLAB Harmony. This wrapper also initializes the SEGGER emWin Graphics Library and maintains the state machine of the wrapper. It generates the templates for GUI and LCD

configuration files required by the emWin application.

Touch Wrapper

The Touch Wrapper can be used for integrating the SEGGER emWin Touch interface with MPLAB Harmony. This wrapper interfaces the Messaging System Service Library with the emWin Touch interface and decodes the touch data from the Messaging System Service and encodes it as required by the emWin Touch interface.

Using the Library

This topic describes the basic architecture of the GUI and Touch Wrapper Library and provides information and examples on how to use it.

Description

Interface Header File: `emwin_gui_static.h` and `emwin_touch_static.h`

The interface to the GUI and Touch Wrapper Library is defined in the `emwin_gui_static.h` and `emwin_touch_static.h` header files. Any C language source (`.c`) file that uses the library should include `emwin_gui_static.h` and `emwin_touch_static.h`.

The SEGGER emWin Graphics Library includes other header files located at `<install-dir>/bin/framework/gfx/segger_emwin/inc` and `<install-dir>/apps/<app-dir>/src/system_config/<configuration>/third_party/gfx/emwin/config`. These two paths are required to be included into the project include path.

Library Source Files

The GUI and Touch Wrapper Library template files are provided in the `<install-dir>/third_party/gfx/emwin/gui/templates` and `<install-dir>/third_party/gfx/emwin/touch/templates` directory. The MPLAB Harmony Configurator (MHC) generates C source files using these template files within the application `system_config` folder. The templates folder may contain optional files and alternate implementations. Please refer to Configuring the Library for instructions on how to select optional features and to Building the Library for instructions on how to build the library.

Abstraction Model

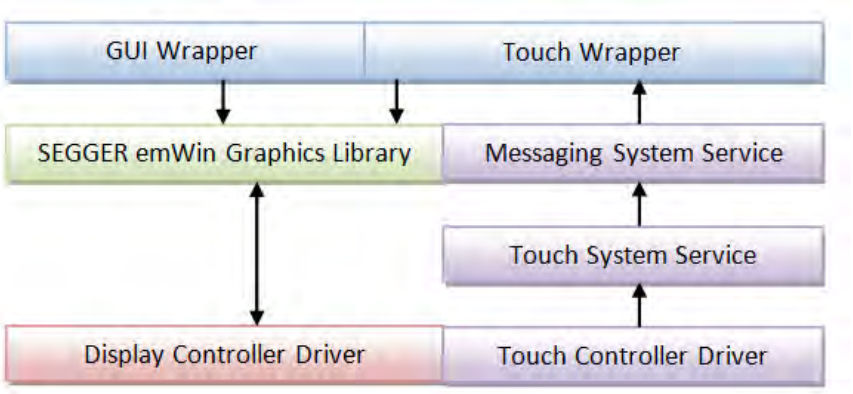
This library provides an abstraction of the GUI and Touch Wrapper Library. This topic describes how that abstraction is modeled in the software and introduces the library interface.

Description

The GUI and Touch Wrapper Library interface defines a superset abstraction of the functionality provided by any specific implementation or configuration of the library. This topic describes how that abstraction is modeled in software and introduces the library's interface. Refer to Configuring the Library to determine the actual set of features that are supported for each configuration option.

As shown in the following diagram, the GUI and Touch Wrapper Library uses services of the third-party SEGGER emWin Graphics Library and the MPLAB Harmony Messaging System Service Library. The GUI and Touch Wrapper Library writes to the display using the display controller driver. The display controller driver can be from the SEGGER emWin Graphics Library or from MPLAB Harmony. The Messaging System Service services the messages sent by the Touch System Service in First In First Out (FIFO) order. The Touch System Service Library encodes the touch commands from the touch data received from the Touch Controller Driver in MPLAB Harmony.

GUI and Touch Wrapper Library Software Abstraction Model



Library Overview

The Library Interface functions are divided into various sub-sections, each of which interacts with one or more of the items identified in the abstraction model.

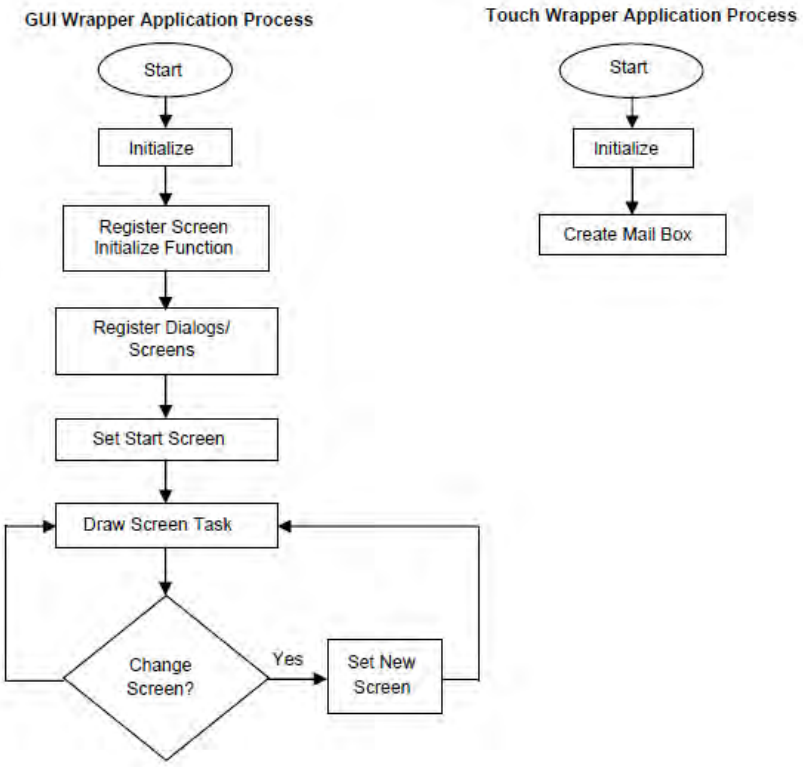
Library Interface Section	Description
GUI Wrapper Functions	Provides functions to initialize the GUI Wrapper as well as tasks and screen functions.
Touch Wrapper Functions	Provides functions for initializing the Touch Wrapper and creating a mail box.

How the Library Works

This section provides information on how the GUI and Touch Wrapper Library works.

Description

The following figure highlights the processes that the application must follow to use the GUI and Touch Wrapper Library.



Initializing the GUI and Touch Wrappers

Describes how to initialize the GUI Wrapper and Touch Wrapper.

Description

The application needs to initialize the GUI and Touch Wrapper Library. The library is initialized successfully by selecting the correct MHC configuration values.

The GUI Wrapper requires the MHC configuration values for “Memory Block Size” and “Number of Screens” to be set.

The Touch Wrapper requires the MHC configuration values for “System Message Service Instance” to be set.

The following code shows an example of designing the data structure EMWIN_TOUCH_INIT and also shows how example usage of the emWin_TouchInitialize and emWin_GuiInitialize functions.

```
EMWIN_TOUCH_INIT emWinTouchInitData;

/* Set the Messaging System Service instance */
 emWinTouchInitData.iSysMsg = SYS_MSG_0;

/* Initialize the emWin Touch Wrapper */
 emWin_TouchInitialize( (SYS_MODULE_INIT *)&emWinTouchInitData );

/* Initialize the emWin GUI Wrapper */
emWin_GuiInitialize();
```

Touch Wrapper Setup

Describes how to set up the Touch Wrapper.

Description

After initializing the Touch Wrapper, the Messaging System Service Mail Box required by the Touch System Service needs to be created.

To create the mail box, the user application needs to call the emWin_TouchMailBoxCreate function from the GUI and Touch Wrapper Library.

The following code shows an example of creating the mail box using emWin_TouchMailBoxCreate.

```
emWin_TouchMailBoxCreate();
```

GUI Wrapper Setup

Describes how to set up the GUI Wrapper.

Description

After initializing the GUI Wrapper, it must be set up with the screen or dialog and resources code generated by the emWin tools. This may require initializing the screens parameters such as background color or font by registering the Screen Initialize function using the emWin_GuiScreenInitializeRegister function. After registering the Screen Initialize function, emWin screens or dialogs need to be registered using the emWin_GuiScreenRegister function. Please note the all register functions only perform the registration. The registered function is called at the appropriate occurrence of the state of the GUI Tasks state machine. To start drawing the screens or dialog a starting screen or dialog needs to be set using the emWin_GuiStartScreenSet function.

The following code shows an example of setting up the GUI Wrapper.

```
/* Create Screen/dialog function pointer array */
EMWIN_GUI_SCREEN_CREATE emWinScreenCreate[ EMWIN_GUI_NUM_SCREENS ]
                                        = { CreateScreen1,
                                            CreateScreen2,
                                            CreateScreen3 };

/* Screen Initialize Function */
void APP_ScreenInitialize ( void )
{
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
}

/* Register Screen Initialize Function */
emWin_GuiScreenInitializeRegister( APP_ScreenInitialize );

/* Register Screen/dialog Initialize */
for( screenCount = 0; screenCount < EMWIN_GUI_NUM_SCREENS; screenCount++ )
{
    emWin_GuiScreenRegister( screenCount, emWinScreenCreate[screenCount]);
}

/* Set the start Screen/dialog as first screen */
emWin_GuiStartScreenSet( 0 );
```

GUI Wrapper Screen Change

Describes how to change a screen using the GUI Wrapper.

Description

Once the GUI is set up, emWin_GuiTasks will call APIs to draw the screen, and based on events, the SEGGER emWin Graphics Library will redraw the screen. To draw a new screen based on the event, call the emWin_GuiScreenChange function.

The following code shows an example of changing screen on event.

```
/* Abstract of event code from GUIBuilder generated Screen/dialog */
   switch(Id) {
      case ID_BUTTON_0: // Notifications sent by 'Next'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
         // USER START (Optionally insert code for reacting on notification message)
         // On click of next button set the next screen/dialog
         emWin_GuiScreenChange(SCREEN_2);
         // USER END
```

```
      break;
   case WM_NOTIFICATION_RELEASED:
      // USER START (Optionally insert code for reacting on notification message)
      // USER END
         break;
      // USER START (Optionally insert additional code for further notification handling)
      // USER END
      }
   break;
      // USER START (Optionally insert additional code for further IDs)
      // USER END
      }
```

Configuring the Library

The GUI and Touch Wrapper Library includes the `system_config.h` file. This file is generated by the MPLAB Harmony Configurator (MHC). It defines the user-selected configuration options necessary to build the library.

This header can be placed anywhere; however, the path of this header needs to be present in the include search path for a successful build. Refer to the Applications Help section for more details.

Building the Library

This section lists the files that are available with the GUI and Touch Wrapper Library. It lists the files need to be included in the build based on configuration option selected by the system.

Description

The following three tables list and describe the header (`.h`) and source (`.c`) files that implement this library. The parent folder for these files is `<install-dir>/third-party/gfx/emwin`.

Required File(s)

All of the required files listed in the following table are automatically added into the MPLAB X IDE project by the MHC when the library is selected for use.

This table lists and describes the source and header files that must *always* be included in the MPLAB X IDE project to build this library.

Source File Name	Description
`gui/emwin_gui_static.h`	This file should be included by any `.c` file that accesses the GUI Wrapper API. This file contains the prototypes for the GUI Wrapper APIs.
`touch/emwin_touch_static.h`	This file should be included by any `.c` file that accesses the Touch Wrapper API. This file contains the prototypes for the Touch Wrapper APIs

Optional File(s)

This table lists and describes the source and header files that may optionally be included if required for the desired implementation.

Source File Name	Description
`touch/src/emwin_touch_static.c`	This file should always be included in the project when using the Touch Wrapper.
`gui/src/emwin_gui_static.c`	This file should always be included in the project when using the GUI Wrapper.
`config/GUI_X_Ex.c`	This file should always be included in the project when using the GUI Wrapper.
`config/GUIConf.c`	This file should always be included in the project when using the GUI Wrapper.
`config/LCDConf.c`	This file should always be included in the project when using the GUI Wrapper.

Module Dependencies

The GUI and Touch Wrapper Library is dependent upon the following modules:

- SEGGER emWin Graphics Library
- Messaging System Service Library
- Touch System Service Library

Library Interface

GUI Wrapper Data Types and Constants

	Name	Description
	EMWIN_GUI_SCREEN_CREATE	Pointer to emWin Screen Create function.
	EMWIN_GUI_SCREEN_INITIALIZE	Pointer to emWin Screen Initialize Function.

GUI Wrapper Functions

	Name	Description
	emWin_GuiInitialize	Initializes the emWin GUI Wrapper.
	emWin_GuiScreenChange	Sets the ID of the next screen to be drawn.
	emWin_GuiScreenInitializeRegister	Registers the Screen Initialize function.
	emWin_GuiScreenRegister	Register the GUIBuilder generated screen.
	emWin_GuiStartScreenSet	Sets the start screen.
	emWin_GuiTasks	Maintains the emWin GUI Wrapper's state machine.
	emWin_GuiScreenEnd	Ends the already created screen.
	emWin_GuiScreenGet	Gets handle of the screen.

Touch Wrapper Data Types and Constants

	Name	Description
	EMWIN_TOUCH_INIT	Defines the data required to initialize the emWin Touch Wrapper.

Touch Wrapper Functions

	Name	Description
	emWin_TouchInitialize	Initializes the emWin Touch Wrapper.
	emWin_TouchMailBoxCreate	Create Mail Box for the touch input messages.

Description

This section provides information on the Application Programming Interface (API) functions provided in the GUI and Touch Wrapper Library.

GUI Wrapper Functions

emWin_GuiInitialize Function

Initializes the emWin GUI Wrapper.

File

emwin_gui.h

C

```
void emWin_GuiInitialize();
```

Returns

None.

Description

This function initializes the emWin GUI wrapper.

Remarks

None.

Preconditions

None.

Example

```
emWin_GuiInitialize();
```

Function

void emWin_GuiInitialize(void)

emWin_GuiScreenChange Function

Sets the ID of the next screen to be drawn.

File

emwin_gui.h

C

```
void emWin_GuiScreenChange(int32_t screenId);
```

Returns

None.

Description

This function sets the ID of the next screen to be drawn.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called. All screens must have been registered using emWin_GuiScreenRegister.

Example

```
    typedef enum
{
  EMWIN_APP_SCREEN_1 = 0,
  EMWIN_APP_SCREEN_2,
  EMWIN_APP_SCREEN_3,

} EMWIN_APP_SCREEN_ID;

//Custom code addition to the Guibuilder Generated Screen 1 code
static void _cbDialog(WM_MESSAGE * pMsg) {
    const void * pData;
    WM_HWIN      hItem;
    U32          FileSize;
    int          NCode;
    int          Id;
    // USER START (Optionally insert additional variables)
    // USER END

    // Intermediate code.

    switch(Id) {
        case ID_BUTTON_0: // Notifications sent by 'ButtonNext'

            switch(NCode) {
                case WM_NOTIFICATION_CLICKED:
                // USER START (Optionally insert code for reacting on notification message)
                // USER END
                break;

                case WM_NOTIFICATION_RELEASED:
                // USER START (Optionally insert code for reacting on notification message)
        // Change to new screen with ID EMWIN_APP_SCREEN_2
        emWin_GuiScreenChange(EMWIN_APP_SCREEN_2);
                // USER END
                break;
                // USER START (Optionally insert additional code for further notification handling)
                // USER END
                }
```

```
            break;
            // USER START (Optionally insert additional code for further notification handling)
            // USER END
}
```

Parameters

Parameters	Description
screenId	Index to the array of GUIBuilder generated screens.

Function

void emWin_GuiScreenChange(int32_t screenId)

emWin_GuiScreenInitializeRegister Function

Registers the Screen Initialize function.

File

emwin_gui.h

C

```
void emWin_GuiScreenInitializeRegister(EMWIN_GUI_SCREEN_INITIALIZE screenInit);
```

Returns

None.

Description

This function is used to register the Screen Initialize function.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called.

Example

```
void APP_ScreenInitialize( void )
{
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
}

emWin_GuiScreenInitializeRegister( APP_ScreenInitialize );
```

Parameters

Parameters	Description
screenInit	Pointer to the Screen Initialize function.

Function

void emWin_GuiScreenInitializeRegister

(

EMWIN_GUI_SCREEN_INITIALIZE screenInit

)

emWin_GuiScreenRegister Function

Register the GUIBuilder generated screen.

File

emwin_gui.h

C

```
void emWin_GuiScreenRegister(int32_t screenId, EMWIN_GUI_SCREEN_CREATE screen);
```

Returns

None.

Description

This function registers the GUIBuilder generated screens.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called.

Example

```
EMWIN_GUI_SCREEN_CREATE emWinScreenCreate[ EMWIN_GUI_NUM_SCREENS ]
                                    = { Screen_1,
                                        Screen_2,
                                        Screen_3 };

int32_t screenCount = 0;

for( screenCount = 0; screenCount < EMWIN_GUI_NUM_SCREENS; screenCount++ )
{
    emWin_GuiScreenRegister( screenCount, emWinScreenCreate[screenCount]);
}
```

Parameters

Parameters	Description
screenId	Index to the array of the screens.
screen	pointer to the GUIBuilder generated screen.

Function

void emWin_GuiScreenRegister

(

int32_t screenId,

EMWIN_GUI_SCREEN_CREATE screen

)

emWin_GuiStartScreenSet Function

Sets the start screen.

File

emwin_gui.h

C

```
void emWin_GuiStartScreenSet(int32_t screenId);
```

Returns

None.

Description

This functions sets the start screen.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called.

Example

```
typedef enum
{
  EMWIN_APP_SCREEN_1 = 0,
  EMWIN_APP_SCREEN_2,
  EMWIN_APP_SCREEN_3,

} EMWIN_APP_SCREEN_ID;

EMWIN_GUI_SCREEN_CREATE emWinScreenCreate[ EMWIN_GUI_NUM_SCREENS ]
                                      = { Screen_1,
                                          Screen_2,
                                          Screen_3 };

int32_t screenCount = 0;

for( screenCount = 0; screenCount < EMWIN_GUI_NUM_SCREENS; screenCount++ )
{
    emWin_GuiScreenRegister( screenCount, emWinScreenCreate[screenCount]);
}

emWin_GuiStartScreenSet( EMWIN_APP_SCREEN_2 );
```

Parameters

Parameters	Description
screenId	Index to the array of GUIBuilder generated screens.

Function

void emWin_GuiStartScreenSet(int32_t screenId);

emWin_GuiTasks Function

Maintains the emWin GUI Wrapper's state machine.

File

emwin_gui.h

C

```
void emWin_GuiTasks();
```

Returns

None.

Description

This function is used to maintain the emWin GUI Wrapper's internal state machine. This function should be called from the SYS_Tasks function.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called. All screens must have been registered using emWin_GuiScreenRegister.

Example

```
while (true)
{
   emWin_GuiTasks();

   //Do other tasks

}
```

Function

void emWin_GuiTasks(void)

emWin_GuiScreenEnd Function

Ends the already created screen.

File

emwin_gui.h

C

```
void emWin_GuiScreenEnd(int32_t screenId);
```

Returns

None.

Description

This function ends the already created screen. The handle of the ended screen will no longer be valid. This function is used when screen to be end during transition to from the current screen to another screen.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called.

Example

```
    typedef enum
{
  EMWIN_APP_SCREEN_1 = 0,
  EMWIN_APP_SCREEN_2,
  EMWIN_APP_SCREEN_3,

} EMWIN_APP_SCREEN_ID;

//Custom code addition to the Guibuilder Generated Screen 1 code
static void _cbDialog(WM_MESSAGE * pMsg) {
    const void * pData;
    WM_HWIN      hItem;
    U32          FileSize;
    int          NCode;
    int          Id;
    // USER START (Optionally insert additional variables)
    // USER END

    // Intermediate code.

    switch(Id) {
        case ID_BUTTON_0: // Notifications sent by 'ButtonNext'

            switch(NCode) {
                case WM_NOTIFICATION_CLICKED:
                // USER START (Optionally insert code for reacting on notification message)
                // USER END
                break;

                case WM_NOTIFICATION_RELEASED:
                // USER START (Optionally insert code for reacting on notification message)
            // Change to new screen with ID EMWIN_APP_SCREEN_2
            emWin_GuiScreenChange(EMWIN_APP_SCREEN_2);
            emWin_GuiScreenEnd(EMWIN_APP_SCREEN_1);
                // USER END
                break;
                // USER START (Optionally insert additional code for further notification handling)
                // USER END
                }

            break;
```

```
            // USER START (Optionally insert additional code for further notification handling)
            // USER END
}
```

Parameters

Parameters	Description
screenId	Index to the array of GUIBuilder generated screens.

Function

void emWin_GuiScreenEnd(int32_t screenId);

emWin_GuiScreenGet Function

Gets handle of the screen.

File

emwin_gui.h

C

```
WM_HWIN emWin_GuiScreenGet(int32_t screenId);
```

Returns

Handle of the already created screen. Returns 0 if the screenId is invalid.

Description

This functions gets the handle of the screen with screen Id provided by screenId variable. This function is used when objects from one screen to be accessed in another screen.

Remarks

None.

Preconditions

The emWin_GuiInitialize function must have been called.

Example

```
typedef enum
{
  EMWIN_APP_SCREEN_1 = 0,
  EMWIN_APP_SCREEN_2,
  EMWIN_APP_SCREEN_3,

} EMWIN_APP_SCREEN_ID;

extern GUI_CONST_STORAGE GUI_BITMAP bmMyImage;
    WM_HWIN hScreen2;

// Custom code addition to the Guibuilder Generated Screen 1 code
switch(Id) {
        case ID_BUTTON_0: // Notifications sent by 'ButtonNext'

        switch(NCode) {
                case WM_NOTIFICATION_CLICKED:
                // USER START (Optionally insert code for reacting on notification message)
            // Set BMP Image in screen 2 based on event on screen 1
            hScreen2 = emWin_GuiScreenGet( EMWIN_APP_SCREEN_2 );
        hItem = WM_GetDialogItem( hScreen2, ID_SCREEN2_IMAGE_0);
                    IMAGE_SetBitmap( hItem, &bmMyImage );
                // USER END
                break;

                case WM_NOTIFICATION_RELEASED:
                // USER START (Optionally insert code for reacting on notification message)
                // USER END
                break;
                // USER START (Optionally insert additional code for further notification handling)
```

```
                // USER END
                }

            break;
            // USER START (Optionally insert additional code for further notification handling)
            // USER END
}
```

Parameters

Parameters	Description
screenId	Index to the array of GUIBuilder generated screens.

Function

WM_HWIN emWin_GuiScreenGet(int32_t screenId)

GUI Wrapper Data Types and Constants

EMWIN_GUI_SCREEN_CREATE Type

Pointer to emWin Screen Create function.

File

emwin_gui.h

C

```
typedef WM_HWIN (* EMWIN_GUI_SCREEN_CREATE)(void);
```

Returns

Handle of the created dialog.

Description

emWin Screen Create Function Pointer

This data type defines the function signature for the emWin Screen Create function. The application must register the pointers to the different Screen create functions with signature matching to the types specified by this function pointer.

Remarks

The array of type EMWIN_GUI_SCREEN_CREATE is initialized by Screen_1, Screen_2 and Screen_3. Screen_1 to Screen_3 are functions auto generated by the emWin tool: GUIBuilder.

Example

```
EMWIN_GUI_SCREEN_CREATE emWinScreenCreate[ EMWIN_GUI_NUM_SCREENS ]
                                    = { Screen_1,
                                        Screen_2,
                                        Screen_3 };

int32_t screenCount = 0;

for( screenCount = 0; screenCount < EMWIN_GUI_NUM_SCREENS; screenCount++ )
{
    emWin_GuiScreenRegister( screenCount, emWinScreenCreate[screenCount]);
}
```

EMWIN_GUI_SCREEN_INITIALIZE Type

Pointer to emWin Screen Initialize Function.

File

emwin_gui.h

C

```
typedef void (* EMWIN_GUI_SCREEN_INITIALIZE)(void);
```

Returns

None.

Description

emWin Screen Initialize Function Pointer

This data type defines the function signature for the emWin Screen Create function. The application can register this function in case any emWin initialization functions to be called before creation of the screens.

Remarks

The screen initialize if registered will be called once before screen create.

Example

```
void APP_ScreenInitialize( void )
{
    GUI_SetBkColor(GUI_BLACK);
    GUI_Clear();
}

EMWIN_GUI_SCREEN_INITIALIZE screenInitialize = &APP_ScreenInitialize;

emWin_GuiScreenInitializeRegister( screenInitialize );
```

Touch Wrapper Functions

emWin_TouchInitialize Function

Initializes the emWin Touch Wrapper.

File

emwin_touch.h

C

```
void emWin_TouchInitialize(const SYS_MODULE_INIT * const init);
```

Returns

None.

Description

This function initializes the emWin Touch Wrapper.

Remarks

None.

Preconditions

None.

Example

```
EMWIN_TOUCH_INIT touchInit;

touchInit.iSysMsg = SYS_MSG_0;

emWin_TouchInitialize( &touchInit );
```

Parameters

Parameters	Description
init	Pointer to a data structure containing any data necessary to Initialize the emWin Touch Wrapper.

Function

void emWin_TouchInitialize(const SYS_MODULE_INIT * const init)

emWin_TouchMailBoxCreate Function

Create Mail Box for the touch input messages.

File

emwin_touch.h

C

```
void emWin_TouchMailBoxCreate();
```

Returns

None.

Description

This function creates Mail Box for the touch input messages.

Remarks

None.

Preconditions

The emWin_TouchInitialize function must have been called.

Example

```
void APP_Initialize ( void )
{
   emWin_TouchMailBoxCreate();
}
```

Function

void emWin_TouchMailBoxCreate(void)

Touch Wrapper Data Types and Constants

EMWIN_TOUCH_INIT Structure

Defines the data required to initialize the emWin Touch Wrapper.

File

emwin_touch.h

C

```
typedef struct {
  SYS_MSG_INSTANCE iSysMsg;
} EMWIN_TOUCH_INIT;
```

Members

Members	Description
SYS_MSG_INSTANCE iSysMsg;	message system service instance

Description

emWin Touch Wrapper Initialization Data

This data type defines the data required to initialize the emWin Touch Wrapper.

Remarks

None.

Using SEGGER emWin with MPLAB Harmony

This section describes using the SEGGER emWin Graphics Library and its utilities with MPLAB Harmony on Microchip development hardware equipped with PIC32 devices.

Description

emWin is a third-party graphics library from SEGGER. The graphics library is used to create a GUI application on PIC32-based hardware from Microchip. The library binary and the associated utilities are part of the SEGGER emWin PRO tool suite and are shipped for free with MPLAB Harmony version 1.07 or later under PIC32 license.

Important! It is the responsibility of the user to understand and comply with any third party license terms or requirements applicable to the use of such third party software, specifications, systems, or tools. Please consult the MPLAB Harmony Software License Agreement for full details. A PDF copy of the software license agreement is provided in the `<install-dir>/doc` folder of your installation of MPLAB Harmony.

Library Binary

The SEGGER emWin Graphics Library binary is located in `<install-dir>\bin\framework\gfx\segger_emwin\lib`.

Linking Applications

To link the library binary file, applications require the header files, which are located in `<install-dir>\bin\framework\gfx\segger_emwin\inc`.

Using the Library Binary in an Application

To use the emWin library binary in the application, link the library binary to the MPLAB Harmony application and call the library APIs.

Supported Features

The SEGGER emWin Graphics Library supports the following features:

- Bitmaps of different color depths supported
- Variety of different fonts are shipped
- Ability to define and link new fonts
- Ability to show and edit values in decimal, binary, hexadecimal, any font
- Window manager support
- Widgets support
- Input device support (keyboard, mouse, touch)
- Any display with any controller supported
- Cache support for slower display controllers
- Configurable display size
- Fast drawing of point, line, circle, and polygon
- Different drawing mode

Configuration Files

The SEGGER emWin Graphics Library is Configurable. The library supports run-time configuration using library APIs and compile-time configuration using library macros. The precompiled library binary is provided in your installation of MPLAB Harmony with certain compile-time configurable macros set. In this case, only run-time configurable parameters can be modified. The configuration macros and APIs need to be defined or called in C header or source files. The configuration C header or source files are either to be manually created or will be generated by MPLAB Harmony Configurator (MHC), based on selection of MHC parameters. The compile-time parameters are located in the `GUIConf.h` and `LCDConf.h` header files. The run-time configuration of the library must be done in the files `GUIConf.c`, `LCDConf.c` and GUI_Ex_X.c C files. If the SEGGER emWin Wrapper Library is selected in MHC, the configuration related files will be generated by the Wrapper Library.

SEGGER emWin Utilities

Your installation of MPLAB Harmony also provides the following SEGGER emWin utilities from the PRO tool suite, which are located in `<install-dir>\ utilities\segger\emwin`.

- Binary to C Converter
- BMP Converter
- emWin VNC Client
- emWin SPY
- emWin Windows View
- GUI Builder
- JPEG to Movie Converter
- Font Converter (Demonstration Setup Executable)
- UTF-8 Text to C Converter

Refer to SEGGER emWin Utilities for more information.

The utilities provided in your installation of MPLAB Harmony assist users to:

- Design the GUI

- Create the graphics resources
- Optimize the graphics resources
- Monitor the application parameters

The output of most of the tools is either a binary file or a C file to be linked with the application code. Please note that the tools can be run only on a host machine with the Windows Operating System.

For more information on the SEGGER emWin Graphics Library, its configuration and tools, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

SEGGER emWin and MPLAB Harmony Integration

To run a SEGGER emWin application using MPLAB Harmony on a PIC32 device hardware platform, the following SEGGER emWin modules and MPLAB Harmony modules need to be integrated:

- SEGGER emWin application code (generated by SEGGER emWin utilites)
- SEGGER emWin Configuration
- SEGGER emWin Graphics Library
- MPLAB Harmony Display Controller Driver
- MPLAB Harmony Messaging System Service
- MPLAB Harmony Touch System Service
- MPLAB Harmony Touch Controller Driver

There are two ways to integrate these modules. One way is to write custom integration wrapper code. Another way is to use the SEGGER emWin Wrapper Library. The Wrapper Library provides a mechanism to integrate these modules. The Wrapper Library is divided into two parts, the GUI Wrapper and the Touch Wrapper.

GUI Wrapper

The GUI Wrapper code integrates GUI-related SEGGER emWin application code with MPLAB Harmony. It has its own state machine to maintain the GUI states and integrates the following SEGGER emWin and MPLAB Harmony modules:

- SEGGER emWin application code
- SEGGER emWin configuration
- SEGGER emWin Graphics Library
- MPLAB Harmony Display Controller Driver

Touch Wrapper

The Touch Wrapper code integrates touch-related SEGGER emWin application code with MPLAB Harmony and integrates the following SEGGER emWin and MPLAB Harmony modules:

- SEGGER emWin Application code
- SEGGER emWin Graphics Library
- MPLAB Harmony Messaging System Service
- MPLAB Harmony Touch System Service
- MPLAB Harmony Touch Controller Driver

Refer to GUI and Touch Wrapper Library for SEGGER emWin for more information on SEGGER emWin Graphics Library integration with MPLAB Harmony using the Wrapper Library.

Getting Started

Refer to the Getting Started section to get started with designing a GUI using the SEGGER emWin utilities and integrating the GUI with MPLAB Harmony.

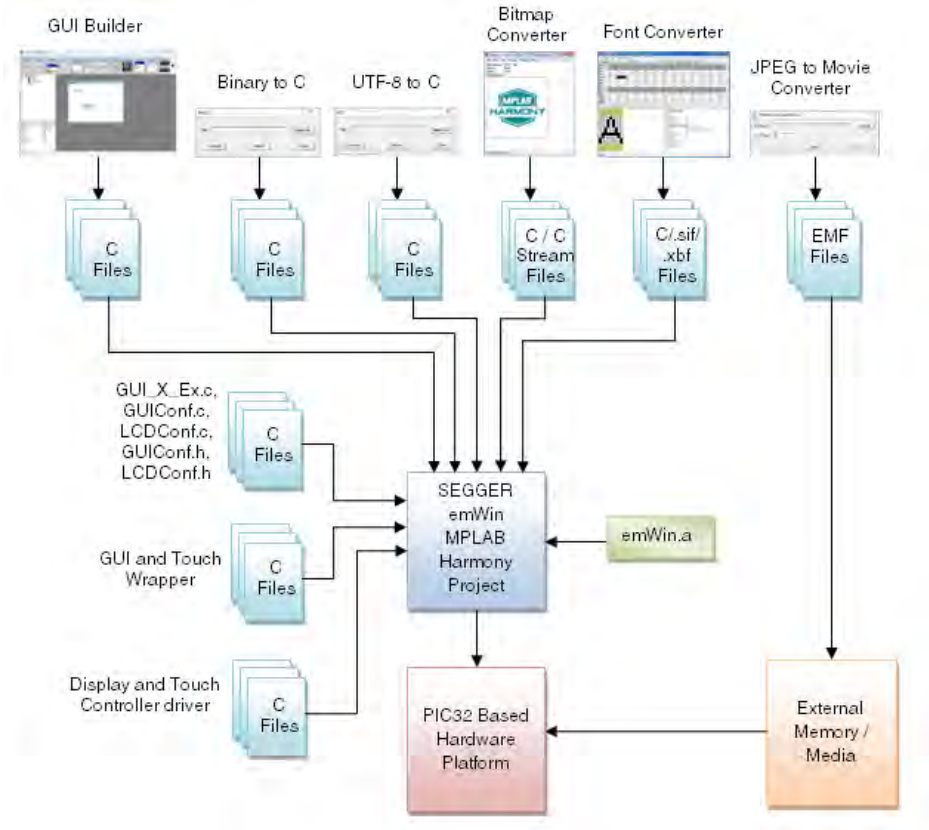
SEGGER emWin Utilities

This section describes the SEGGER emWin utilities and their usage.

Description

There are multiple vital utilities provided with the SEGGER emWin release package within MPLAB Harmony. These utilities are located within your installation of MPLAB Harmony in `<install-dir>\utilities\segger\emwin`.

All of the utilities run on a host computer. The following diagram shows the SEGGER emWin utilities provided in MPLAB Harmony, their output and the integration of the output with the MPLAB Harmony project.



GUI Builder (GUIBuilder)

This topic describes the GUI Builder (GUIBuilder) utility.

Description

GUIBuilder is a GUI designer utility. A designer having no knowledge of C programming also can use this utility. GUIBuilder also conserves the effort of writing source code by generating the C file that will call the SEGGER emWin Graphics Library APIs. The generated files need further modifications under user-defined code to add more functionality.

Binary Name

GUIBuilder.exe

Usage

- Project Path: Set the project path by modifying the `ProjectPath` variable from `GUIBuilder.ini` located in the same location as `GUIBuilder.exe`
- Parent widget: Add a parent widget to each dialog. Available parent widgets are: Frame Window and Window
- Child widget: Add required child widgets
- Widget properties: Modify widget properties such as size, position, etc.
- Save: Save the loaded dialog as C files at the project path
- Modify C files: Modify the C files as required within user code sections
- Integrate: Integrate the generated C files with emWin application. For more information on integration please refer to GUI and Touch Wrapper Library for SEGGER emWin.

Input

- Widgets
- A C file (generated by GUIBuilder)

Output

A C file.

Binary to C Converter (Bin2C)

This topic describes the Binary to C Converter (Bin2C) utility.

Description

The Bin2C utility converts the binary data from a file to a C file, which is linked to the emWin application, and is read using emWin Library API. Please note that the binary data from a file is converted to a C file "as is". To use the data from the C file by the emWin application, further processing may be required.

Binary Name

Bin2C.exe

Usage

- Load the binary file into the application
- Convert the file
- The converted C file will be located at the same location as of the binary file and is primarily used to convert JPEG or TTF files

Input

Any binary file.

Output

A C file.

UTF-8 Text to C Converter (U2C)

This topic describes the UTF-8 to C Converter (U2C) utility.

Description

The U2C utility converts UTF-8 text to C code by reading a UTF-8 text file and creating a C file with C strings.

Binary Name

U2C.exe

Usage

- UTF-8 text file**:** To create UTF-8 encoded text file using notepad, load the text under notepad, select *File* > *Save As* and select the UTF-8 encoding format while saving the file
- UTF-8 to C: Run `U2C.exe`. Select the UTF-8 encoded file using **Select file…**. Click **Convert** to convert the UTF-8 encoded file to a C file.
- Using the string: Copy the converted string from the C file to an array of characters in the application code. Set the text encode format to UTF-8 using the GUI_UC_SetEncodeUTF8 function. Display the string by passing the array pointer to the GUI_DispString function.

Output

A C file.

Bitmap Converter (BmpCvt)

This topic describes the Bitmap Converter (BmpCvt) utility.

Description

The Bitmap Converter utility is used to convert BMP, PNG or GIF files into the desired emWin bitmap format. The utility supports color conversion, which is used as a tool to reduce memory consumption. The Bitmap Converter also supports other simple functions such as:

- Scaling the bitmap size,
- Flipping the bitmap horizontally or vertically,
- Rotating the bitmap
- Inverting the bitmap indices or colors

Binary Name

BmpCvt.exe

Usage

- Load the image into the application
- Modify the color format, size, flip, rotate
- Save the image in the appropriate format

Input

- BMP file: 1/4/8 bpp (with palette), 16/24/32 bpp, RLE4/8
- GIF file: Please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.
- PNG file: Images with or without alpha channel

Output

- C file: Link to application code
- C stream file: Load on any media
- BMP file: Use for further processing
- GIF file: Use for further processing
- PNG file: Use for further processing

Font Converter (SetupFontCvtDemo_V532)

This topic describes the Font Converter (SetupFontCvtDemo_V532) utility.

Description

The binary provided is a Microsoft Windows setup utility for the demonstration version of the font converter. The demonstration utility will be available in the host computer once the setup binary is successfully installed into the host computer. The font converter utility converts installed PC fonts into an emWin (bitmap) font.

Binary Name

SetupFontCvtDemo_V532.exe

JPEG to Movie Converter (JPEG2Movie)

This topic describes the JPEG to Movie Converter (JPEG2Movie) utility.

Description

The JPEG to Movie Converter utility creates emWin Movie Files (EMF) from multiple JPEG images. The EMF format acts as a container format and can be used to play a movie using the existing JPEG decoding capability of the SEGGER emWin Graphics Library.

Binary Name

JPEG2Movie.exe

Usage

- Frame encoding: If the frame image is raw or has a format other than JPEG, the image need to be encoded into JPEG format. `FFmpeg` is an open source utility available under LGPL or GPL License, which can be used to convert any movie file to JPEG files for each frame.
- Collection: Collect all frames encoded into JPEG format into one folder
- Movie encoding: From the command line, change to the directory containing JPEG2Movie and run the command: `JPEG2Movie.exe <folder path> <ms>`. The output file with EMF format will be created in the same folder containing the JPEG files.

Input

A folder containing JPEG files.

Output

A EMF file.

emWin VNC Client (emVNC)

This topic describes the emWin VNC Client (emVNC) utility.

Description

The emWin Virtual Network Computing (VNC) Client is part of the VNC client server system. The emWin VNC Server runs on the embedded target. The emWin VNC server running on an embedded device allows the display of content from an embedded device to the emWin VNC Client running on the host computer. The emWin VNC Server support is available as a separate package and is not included with your installation of MPLAB Harmony. the emWin VNC server requires a multi-tasking environment with TCP/IP stack support and the TCP/IP stack is not part of the emWin.

Binary Name

emVNC.exe

For more information, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

emWin SPY (emWinSPY)

This topic describes the emWin SPY (emWinSPY) utility.

Description

The emWin SPY utility is used to show run-time information of the embedded target on a PC. The utility shows information about the currently connected emWin application, such as: memory status, information about existing widows, and a list of user input. The utility can also take captures of the current screen. The embedded device and the host computer communicate through a socket connection (TCP/IP) or SEGGER's Real-Time Transfer (RTT).

Binary Name

emWinSPY.exe

For more information, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

emWin Windows View (emWinView)

This topic describes the emWin Windows View (emWinView) utility.

Description

While debugging the emWin application and stepping through the application source code, the display output cannot be seen. The emWin Window View utility solves this problem by showing the simulation display while debugging the simulation. The utility also provides the following additional capabilities:

- Multiple windows for each layer
- Watching the whole virtual layer in one window
- Magnification of each layer window
- Composite view if using multiple layers

Binary Name

emWinView.exe

For more information, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

Integrating SEGGER emWin and MPLAB Harmony

This section describes the process of integrating SEGGER emWin application code with MPLAB Harmony.

Description

The SEGGER emWin application code consists of GUI and graphics resources source code or binary files. The application code needs to be integrated with MPLAB Harmony to run PIC32-based hardware. The SEGGER emWin application code calls SEGGER emWin Graphics library APIs. The SEGGER emWin Graphics Library is a software library and is independent of the underlying hardware. With the help of SEGGER emWin Configuration Source code, MPLAB Harmony performs the I/O task of graphics data generated by SEGGER emWin application code. MPLAB Harmony also passes the pointer input device (PID) data to the SEGGER emWin Graphics Library. The SEGGER emWin Graphics Library recommends implementation of various configuration files to configure the library and LCD interface. The following lists and describes the configuration files:

GUI_EX_X.c

This C file contains the placeholders of the required timing, logging, and multi-tasking related functions. These functions are called by the SEGGER emWin Graphics Library.

GUIConf.c

This C file consists of the GUI_X_Config function, which is called by the SEGGER emWin Graphics Library. The GUI_X_Config function is responsible for assigning a memory block to the memory management system. The memory block needs to be accessible 8-, 16-, and 32-bitwise. To pass the memory block to the SEGGER emWin internal memory management system, the GUI_X_Config function must call the GUI_ALLOC_AssignMemory function with the necessary arguments. The memory device uses the memory block passed by the GUI_ALLOC_AssignMemory function.

GUIConf.h

This header file consists of GUI configuration macros. Please note that these macros are fixed at compile-time and and change in the macro values requires recompilation of the SEGGER emWin Graphics Library. `GUIConf.h` is used to set the following parameters, which are required by the SEGGER emWin Graphics Library during compilation of the library:

- Default font
- Default color
- Default background color
- Memory device support
- Multi-display or Multi-layer support
- Number of layers
- Multi-tasking support
- Number of tasks
- Mouse support
- Default skinning support
- Window manger support
- Window manager transparency support
- Rotation support
- Cursor support
- emWin SPY support
- Debug level
- Default Memory Copy function
- Default Memory Set function
- Maximum number of the PID events managed by the input buffer
- Maximum number of the key events managed by the input buffer

LCDConf.c

This C file consists of the LCD_X_Config display configuration function and the display driver callback function, LCD_X_DisplayDriver. These functions are called by the SEGGER emWin Graphics Library. The LCD_X_Config function sets the configuration of the display driver support and sets the run-time configurations for the LCD controller, as follows:

- Set the number of buffers for multi-buffer support
- Set the display driver parameters and callback functions
- Set the color conversion callback functions
- Set the custom callback routine for copy operation

The LCD_X_DisplayDriver callback function is called by the driver for different tasks. The display driver passes commands for corresponding tasks and a pointer to the command parameters data. Typical commands for the display driver callback are:

- LCD controller initialization
- LCD ON/OFF
- Set layer alpha
- Set Color Lookup Table (CLUT) entries
- Set virtual screen origin
- Set layer position
- Set layer size
- Set layer visibility
- Set video RAM address
- Show buffer with given index

LCDConf.h

This header file consists of general configuration options required for compiling the display driver(s). Please note that these macros are fixed at compile-time and any change in the macro values requires recompilation of the SEGGER emWin Graphics Library. `LCDConf.h` is used to set the following parameter, which is required by drivers from the SEGGER emWin Graphics Library during compilation of the library:

- Display orientation

For more information on adding content to the template, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

Wrapper Libraries

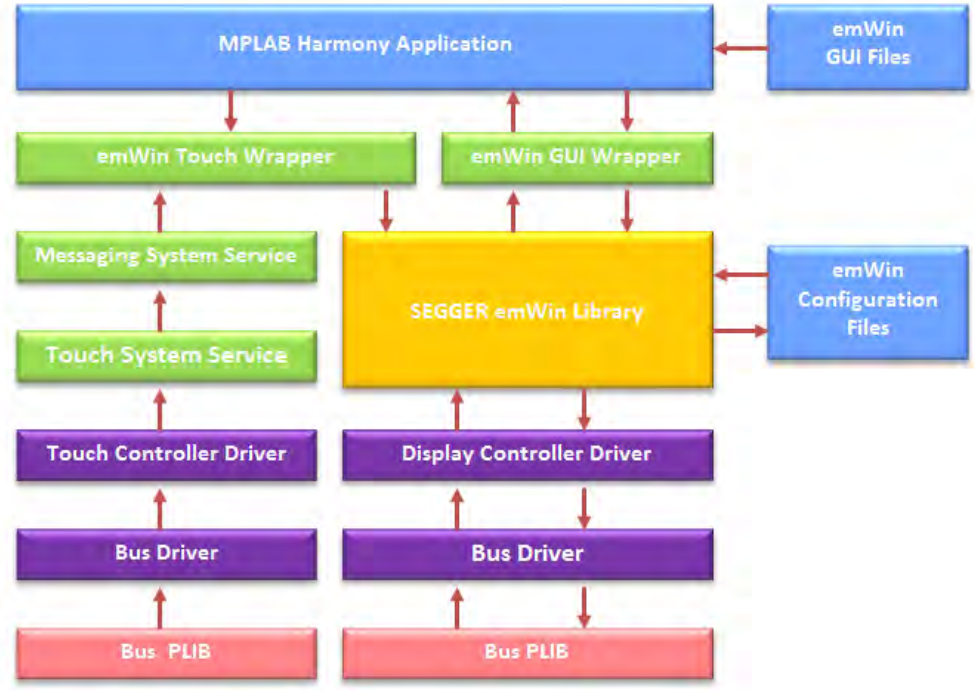
There are multiple ways to design and implement a GUI using the SEGGER emWin Graphics Library. One way is to use the GUIBuilder utility. GUIBuilder can be used to design a GUI in form of dialogs. This utility also generates the C code implementing the designed dialogs. To integrate the generated dialog code and to maintain the state machine calling the dialog, MPLAB Harmony provides the GUI Wrapper Library. The GUI Wrapper library state machine performs:

- Initialization of the SEGGER emWin Graphics Library
- Initialization of the dialog
- Calls the dialog based on events or external input

Similarly, a Touch Wrapper Library is provided by MPLAB Harmony to:

- Initialize the touch interface
- Decode the touch input
- Encode the touch input
- Pass the encoded touch input to the SEGGER emWin Graphics Library.

The following diagram shows the work flow of SEGGER emWin and MPLAB Harmony integration using the GUI and Touch Wrapper Libraries:



As shown in the diagram, the emWin GUI files are passed to MPLAB Harmony application. The emWin GUI files may consist of dialog files generated by GUIBuilder and graphics resource files generated by other SEGGER emWin utilities. Please note that these files need to be manually copied and added to the MPLAB Harmony project. The C code from these files is integrated by calling the appropriate GUI wrapper API. The templates for emWin configuration files such as `GUI_Ex_X.c`, `GUIConf.c`, `LCDConf.c`, `GUIConf.h`, and `LCDConf.h`, are generated by the GUI Wrapper Library. These configuration files may need further editing to configure the GUI and LCD based on application requirements.

The Touch Wrapper Library can be integrated with the MPLAB Harmony application by calling the appropriate Touch Wrapper API.

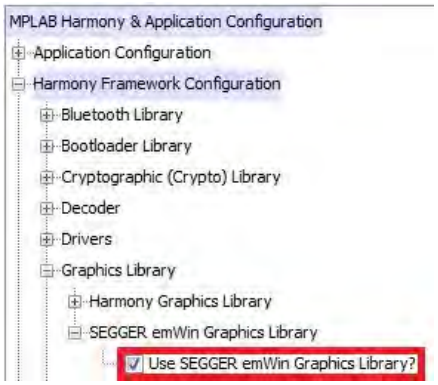
Other MPLAB Harmony modules are integrated by selecting the corresponding BSP in MHC or by selecting required modules separately.

MPLAB Harmony Configurator (MHC)

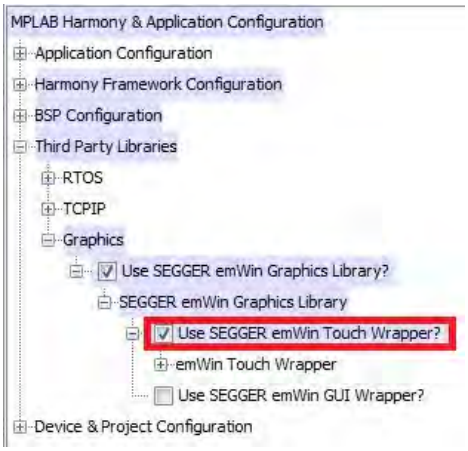
This topic provides information on configuring MHC for the SEGGER emWin Graphics Library.

Description

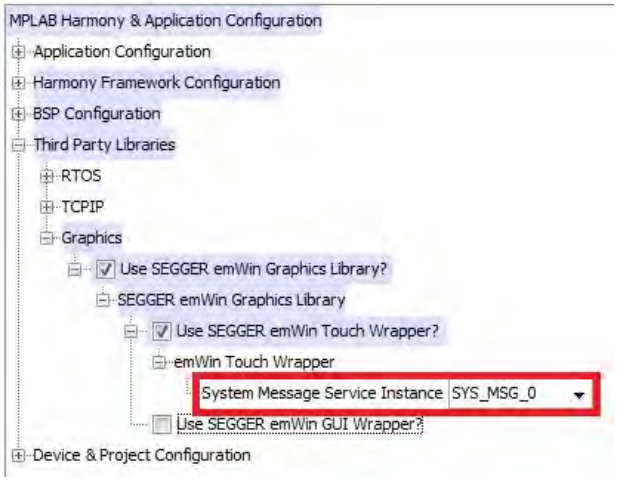
To add the SEGGER emWin Library binary, `emWin.a`, to the MPLAB Harmony application, select “Use SEGGER emWin Graphics Library?” in MHC.



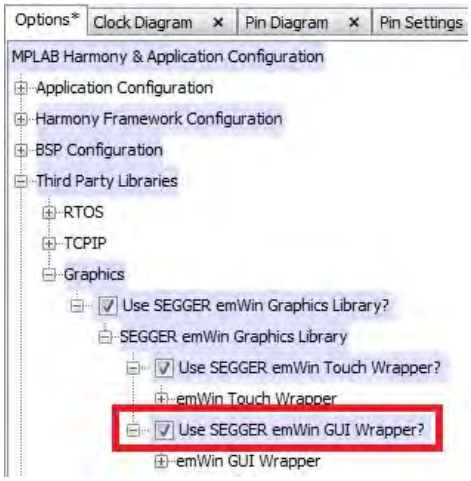
To add the Touch wrapper Library, select “Use SEGGER emWin Touch Wrapper?” in MHC.



To further configure the Touch Wrapper Library, set a suitable value for “System Message Service Instance” from the list of values in MHC. The value of the “System Message Service Instance” must be same as the instance of System Message Service selected in MHC for Touch input.



To add the GUI wrapper Library, select “Use SEGGER emWin GUI Wrapper?” in MHC.



To further configure the GUI Wrapper Library, set suitable value for “Memory Block Size” and “Number of Screens”.

The “Memory Block Size” configuration value represents the size of the memory block assigned to the SEGGER emWin internal memory management system. The memory block is created by defining a static array of a size defined by the “Memory Block Size” configuration value. This memory block is passed to the GUI_ALLOC_AssignMemory function within the `GUIConf.c` file. Please note the size is bytes and must be 8-, 16-, or 32-bit accessible.

The “Number of Screens” configuration value is used to define the array size of dialog creation function. This array is used by the GUI Wrapper to call the appropriate dialog creation function based on the array index.

LCD Integration

This topic provides information on LCD integration.

Description

LCD integration requires the SEGGER emWin Graphics Library to interact with the display controller driver driving the LCD. Generally, there are two types of display controllers:

- Display controller with direct interface
- Display controller with indirect interface

In both cases, the display controller driver is integrated with the SEGGER emWin Graphics Library through the `LCDConf.c` C file and the `LCDConf.h` header file. The method of integrating the display controller driver varies with the type of display controller driver.

For a display controller with a direct interface, the SEGGER emWin Graphics Library can directly write to and read from the video memory. The video memory can be within the system memory or within the display controller (CPU addressable using address bus). The information required when configuring the direct interface type display controller is about the address range and bus width to the display controller. The interface of the direct interface drivers needs to be created and linked with the SEGGER emWin Graphics Library using the GUI_DEVICE_CreateAndLink function. Further configured is needed by specifying the address of the video memory using the LCD_SetVRAMAddrEx function. If the driver supports more operations, these operations can be registered using the GUI_DEVICE_CreateAndLink function. To achieve this, pass the supported operations through the GUI_DEVICE_API parameter of the GUI_DEVICE_CreateAndLine function.

The LCDConf.c code shows a simple example of the configuration of Microchip's Low-Cost Controllerless (LCC) driver from MPLAB Harmony for PIC32 devices. The code example also shows that GUIDRV_LIN_16 driver functions and GUICC_M565 color conversion functions are registered using the GUI_DEVICE_CreateAndLink function. The GUIDRV_LIN_16 driver API structure is used as the MPLAB Harmony LCC driver structure, which is different from the driver structure required by the SEGGER emWin Graphics Library. The LCC driver returns the video memory address, which is further passed to the SEGGER emWin Graphics Library using the LCD_SetVRAMAddrEx function. Other functions, such as LCD_SetSizeEx and LCD_SetVSizeEx, are used to configure the size of the visible area and virtual area in the vertical and horizontal direction.

Configuring the display controller with indirect interface is more complex than with the direct interface.

```
/*********************************************************************
*
*       LCD_X_Config
*
* Purpose:
*   Called during the initialization process in order to set up the
*   display driver configuration.
*
*/
void LCD_X_Config(void)
{
    uintptr_t bufferAddr;

    GUI_DEVICE_CreateAndLink( GUIDRV_LIN_16, GUICC_M565, 0, 0);

    if (LCD_GetSwapXY())
    {
        LCD_SetSizeEx (0, DISP_VER_RESOLUTION, DISP_HOR_RESOLUTION);
        LCD_SetVSizeEx(0, DISP_VER_RESOLUTION, DISP_HOR_RESOLUTION);
    }
    else
    {
        LCD_SetSizeEx (0, DISP_HOR_RESOLUTION, DISP_VER_RESOLUTION);
        LCD_SetVSizeEx(0, DISP_HOR_RESOLUTION, DISP_VER_RESOLUTION);
    }

    bufferAddr = (uintptr_t) DRV_GFX_LCC_GetBuffer();
    LCD_SetVRAMAddrEx( 0, ( void * )bufferAddr );

    return;
}

/*********************************************************************
*
*       LCD_X_DisplayDriver
*
* Purpose:
*   To support the according task the routine needs to be adapted to
*   the display controller.
*
* Parameter:
*   LayerIndex - Index of layer to be configured
*   Cmd        - Please refer to the details in the switch statement below
*   pData      - Pointer to a LCD_X_DATA structure
*
* Return Value:
*   < -1 - Error
*     -1 - Command not handled
*      0 - Ok
*/
int LCD_X_DisplayDriver(unsigned LayerIndex, unsigned Cmd, void * pData)
{
    int retVal = -1;

    switch( Cmd )
    {
        case LCD_X_INITCONTROLLER:
        {
            retVal = 0;
            break;
        }
```

```
            default:
            {
                retVal = -1;
                break;
            }
        }

        return retVal;
    }
```

Touch Integration

This topic provides information on touch integration.

Description

The SEGGER emWin widgets respond to the pointer input device (PID) event based on the area where the event occurs. If the area falls within the widget area, the widget will react by modifying the predefined or custom widget properties. For example, if the user touches the touch screen within the area of an on-screen button, the button will change its properties, such as color or image. To do this, the location of touch needs to be registered with the SEGGER emWin Graphics Library on the touch event. The MPLAB Harmony Touch System Service will send a message on the touch event. The message will contain the description of the event and the coordinates of touch input. The messages from the Touch System Service need to be decoded and passed to the SEGGER emWin Graphics Library. The implementation of decoding the touch message and registering the touch input with the SEGGER emWin Graphics Library is already provided in the SEGGER emWin Touch Wrapper Library. The following code example shows how the Touch Wrapper Library initializes the Messaging System Service by creating the mailbox, decodes the touch message from the Touch System Service, and registers the decoded touch input with the SEGGER emWin Graphics Library.

```
/* Initialize Message System Service by creating a Mail box */
void emWin_TouchMailBoxCreate( void )
{
    emWinTouchData.hMsgType = SYS_MSG_TypeCreate ( emWinTouchData.iSysMsg,
                                                   TYPE_TOUCHSCREEN ,
                                                   0 );

    emWinTouchData.hMailbox = SYS_MSG_MailboxOpen( emWinTouchData.iSysMsg,
                      (SYS_MSG_RECEIVE_CALLBACK) &_emWin_TouchMessageCallback );

    SYS_MSG_MailboxMsgAdd( emWinTouchData.hMailbox, emWinTouchData.hMsgType );

    return;
}
```

In the following code example, iSysMsg is the instance of the Messaging System Service selected in MHC. The _emWin_TouchMessageCallback function is a callback function that decodes the touch message and also registers the touch input using the GUI_TOUCH_StoreStateEx function.

```
/* Decode Touch Message and
   register the touch input with SEGGER emWin Library */
static void _emWin_TouchMessageCallback( SYS_MSG_OBJECT *pMsg )
{
    GUI_PID_STATE * pidState = NULL;

    if( NULL == pMsg )
    {
        return;
    }

    if( TYPE_TOUCHSCREEN != pMsg->nMessageTypeID )
    {
        return;
    }

    pidState = &emWinTouchData.pidState;

    if( EVENT_INVALID == pMsg->param0 ||
        EVENT_MOVE    == pMsg->param0 )
    {
        return;
    }

    if( EVENT_PRESS      == pMsg->param0 ||
        EVENT_STILLPRESS == pMsg->param0 )
    {
        pidState->Pressed = 1;
```

```
    }
    else
    {
        pidState->Pressed = 0;
    }

    pidState->Layer = 0;
    pidState->x     = pMsg->param1;
    pidState->y     = pMsg->param2;

    GUI_TOUCH_StoreStateEx( pidState );

    return;
}
```

For more information on configuring the driver using `LCDConf.c`, please refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

SEGGER emWin Event Handling

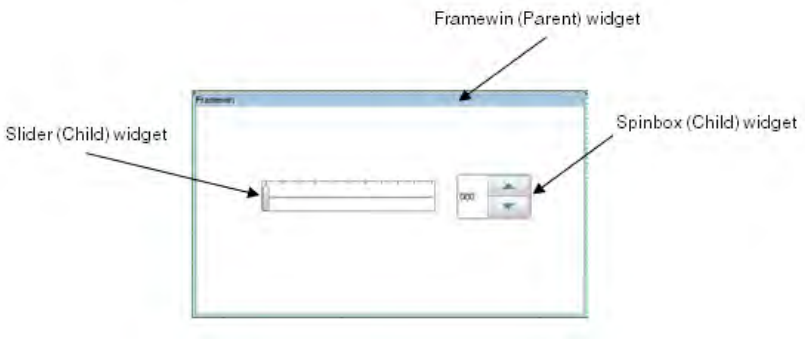
This topic describes the different ways of event handling between MPLAB Harmony and a SEGGER emWin application.

Description

There are different ways in which an application may need to react to different events. Events can be internal events generated by the SEGGER emWin Graphics Library on touch input, or it can be an external event triggered by a button press from the demonstration hardware.

Event Handling Under Same Parent Widget

This is a common scenario of a GUI, where an event generated by one widget, changes the behavior of another widget from the same screen. Both widgets are under the common parent dialog widget. To do this, on the notification of event from one widget, get the handle of the other widget and use relevant APIs to modify the behavior of the other widget. For example, we will take into account a parent widget as the Framewin widget. There are two widgets under the Framewin widget: Spinbox and Slider. The following figure shows the example screen of this scenario.



After designing this GUI using GUIBuilder and saving it as C file, the generated C file needs to be edited to add the corresponding events. To change the value of the spin box by moving the slider, do the following:

1. Open the C file generated by GUIBuilder in MPLAB X IDE.
2. Navigate to the _cbDialog function.
3. Below the notification message for the slider, WM_NOTIFICATION _VALUE_CHANGED, add the following code:

```
/* Get the handle of the Slider */
hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_0);

/* Get the current value of the Slider */
value = SLIDER_GetValue(hItem);

/* Get the handle of the Spinbox */
hItem = WM_GetDialogItem(pMsg->hWin, ID_SPINBOX_0);

/* Set the spinbox value with slider value */
SPINBOX_SetValue(hItem, value);
```

The following code shows the updated dialog file.

```
case ID_SLIDER_0: // Notifications sent by 'Slider'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
```

```
    break;
  case WM_NOTIFICATION_RELEASED:
    // USER START (Optionally insert code for reacting on notification message)
    // USER END
    break;
  case WM_NOTIFICATION_VALUE_CHANGED:
    // USER START (Optionally insert code for reacting on notification message)
        /* Get the handle of the Slider */
        hItem = WM_GetDialogItem(pMsg->hWin, ID_SLIDER_0);

        /* Get the current value of the Slider */
        value = SLIDER_GetValue(hItem);

        /* Get the handle of the Spinbox */
        hItem = WM_GetDialogItem(pMsg->hWin, ID_SPINBOX_0);

        /* Set the spinbox value with slider value */
        SPINBOX_SetValue(hItem, value);
    // USER END
    break;
  // USER START (Optionally insert additional code for further notification handling)
  // USER END
  }
  break;
```

Event Handling Under Different Parent Widgets

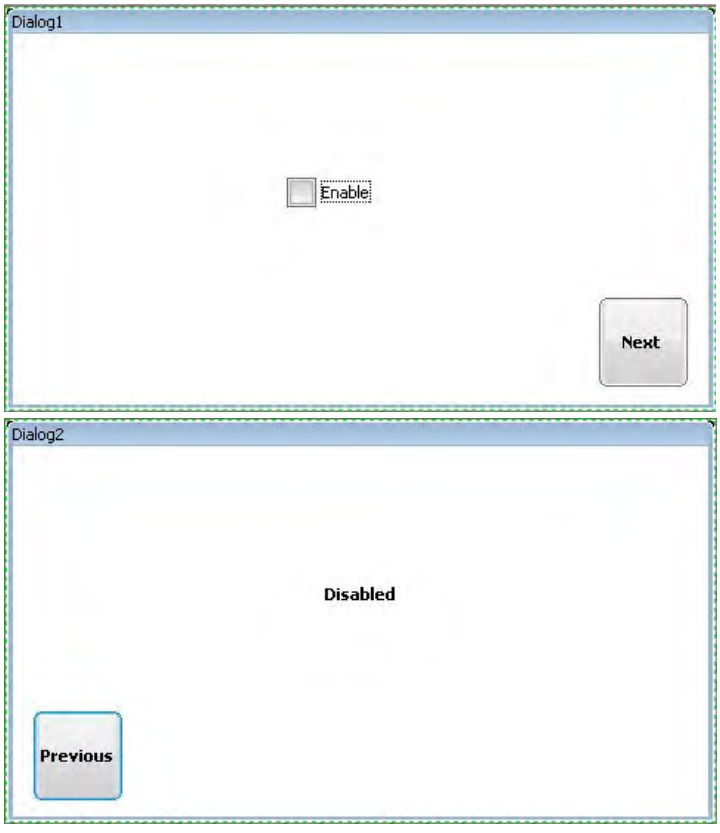
This scenario occurs when a widget under one parent widget changes behavior of a widget under another parent widget. The two parent widgets are referred to as P1 and P2. The child widgets under parent widgets P1 and P2, are referred to as C1 and C2, respectively. There are two possible cases based on whether parent widget P2 exists at the time of an event generated by C1:

- A) P2 parent widget does not exists
- B) P2 parent widget exists

For case A, the properties of child widget C2 need to be initialized based on the state of child widget C1. For case B, the properties of the child widget C2 can be changed on an event generated by child widget C1. The following section discusses both case A and case B in further detail.

Parent Widget P2 Does Not Exist

In this example, we have two parent widgets, Dialog1 widget and Dialog2 widget. The Dialog1 parent widget has two child widgets: a check box and a button. Similarly, the Dialog2 parent widget has two child widgets: text and a button. The following figures show examples of both dialogs:



In this example, to navigate from Dialog1 to Dialog2 the Next button from Dialog1 must be pressed. Similarly, to navigate from Dialog2 to Dialog1 the Previous button from Dialog2 must be pressed. The example starts with Dialog1. Dialog2 will be created only after pressing the Next button in Dialog1. The goal of this example is to change the text from Dialog2 using the check box from Dialog1. By default, the check box from Dialog1 will be clear (i.e., not checked). If the check box remains clear and demonstration is navigated from Dialog1 to Dialog2, the text will have the value value “Disabled”. If the check box from Dialog1 is selected (i.e., checked), on navigating from Dialog1 to Dialog2, the text will have the value “Enabled”. To achieve these results, the text widget from Dialog2 needs to be initialized with a value based on the state of the check box widget from Dialog1. During initialization of the text widget from Dialog2, if the state of the check box widget from Dialog1 is selected, set the value of the text from Dialog2 as “Enabled”; otherwise, set it as “Disabled”. The following code example shows the Dialog2 file generated by GUIBuilder. The file is further edited to add custom initialization code for the text widget.

```
    //
    // Initialization of 'Text_1'
    //
    hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_0);
    TEXT_SetTextAlign(hItem, GUI_TA_HCENTER | GUI_TA_VCENTER);
    TEXT_SetTextColor(hItem, GUI_MAKE_COLOR(0x00000000));
    TEXT_SetFont(hItem, GUI_FONT_13B_1);
    // USER START (Optionally insert additional code for further widget initialization)
    /*
    Get handle of checkbox widget from Dialog1
    using Dialog1 handle and checkbox widget id.
    */
    hItem = WM_GetDialogItem(hDialog1, 0x802);
    /*
    If checkbox is checked set text value to "Enabled"
    */
    if(CHECKBOX_IsChecked(hItem))
    {
        hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_0);
        TEXT_SetText(hItem, "Enabled");
    }
    /*
    else set text value to "Disabled"
    */
    else
    {
        hItem = WM_GetDialogItem(pMsg->hWin, ID_TEXT_0);
        TEXT_SetText(hItem, "Disabled");
    }
    // USER END
```

Parent widget P2 exists

To describe this scenario, we will continue to use the previous example of Dialog1 and Dialog2. In the previous example, we have seen initialization of the properties widget C2 based on an event generated by widget C1, while P2 was not in existence at the time C1 generated the event. Both widgets, C1 and C2, are from different parent widgets, P1 and P2, respectively. Once the demonstration navigates from Dialog1 to Dialog2 for the first time, both parent widgets, Dialog1 and Dialog2, will come into existence and will be persistent until the widgets are destroyed. In such a case, to change the property of widget C2 based on an event generated by widget C1, the code required to change the property of widget C2 needs to be added under the widget C1 event handling code. In the previous example, at first, the check box from Dialog1 was selected to change the text value from Dialog2. To demonstrate the scenario where P2 now exists, the demonstration must navigate back to Dialog1 using the “Previous” button from Dialog2, clear the checked check box from Dialog1, and navigate back to Dialog2 using the “Next” button from Dialog1.

The text value from Dialog2 will still be “Enabled”. Please note that both Dialog1 and Dialog2 were already created and persistent. Navigating from Dialog1 to Dialog2 will not reinitialize the widgets, as Dialog2 and its child widgets are persistent. To change the value of existing widget text from Dialog2, the TEXT_SetText function must be called under event handler code of the widget check box from Dialog1. Once the suitable code is added, the text from Dialog2 will follow the check box selection from Dialog1. Anytime the check box from Dialog1 is cleared, the text from Dialog2 will reflect the value as “Disabled”. If the check box is selected, the text will reflect the value as “Enabled”. The following code example shows the required code under check box event handler from Dialog1.

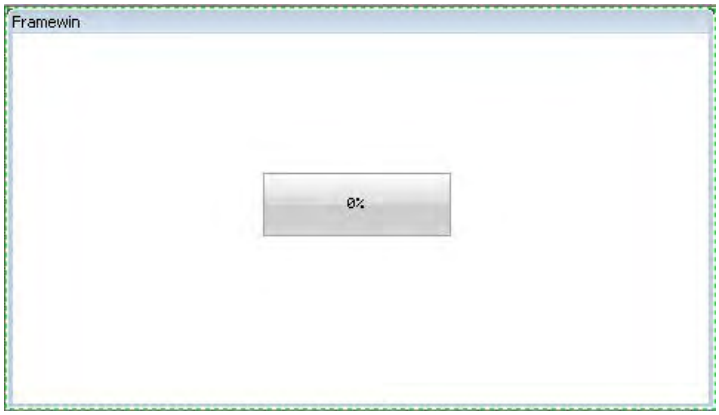
```
case WM_NOTIFY_PARENT:
    Id    = WM_GetId(pMsg->hWinSrc);
    NCode = pMsg->Data.v;
    switch(Id) {
    case ID_CHECKBOX_0: // Notifications sent by 'Checkbox_1'
      switch(NCode) {
      case WM_NOTIFICATION_CLICKED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
      case WM_NOTIFICATION_RELEASED:
        // USER START (Optionally insert code for reacting on notification message)
        // USER END
        break;
```

```
  case WM_NOTIFICATION_VALUE_CHANGED:
    // USER START (Optionally insert code for reacting on notification message)
      /*
      If the Dialog2 handle is valid
      change the properties text widget from Dialog2
      */
      if(hDialog2)
      {
          hItem = WM_GetDialogItem(pMsg->hWin, ID_CHECKBOX_0);
          /*
          If checkbox from Dialog1 is checked
          Set the text value from Dialog2 to Enabled
          */
          if(CHECKBOX_IsChecked(hItem))
          {
              hItem = WM_GetDialogItem(hDialog2, 0x808);
              TEXT_SetText(hItem, "Enabled");
          }
          /*
          Otherwise set the text value from Dialog2 to Disabled
          */
          else
          {
              hItem = WM_GetDialogItem(hDialog2, 0x808);
              TEXT_SetText(hItem, "Disabled");
          }
      }

    // USER END
```

External Event Handling

This section describes the method of changing widget properties based on external events. External events can be a button press or an interrupt generated by a peripheral, etc. Based on the external event, properties of targeted widgets are changed. For example, consider a dialog widget as a parent widget with a progress bar as the child widget. The goal of the demonstration is to increment the progress bar value with a hardware button press. The following figure shows the demonstration screen.



It can be observed that at the center of the screen a progress bar is drawn. To change the value of the progress bar on an external event, such as a hardware button press, use the PROGBAR_SetValue function. In this example. to detect the button press of a button from demonstration hardware, the MPLAB Harmony BSP function, BSP_SwitchStateGet, is used. The following code example shows usage of these functions.

```
static int32_t value = 0;

/*
  Verify if button is pressed and
  valid handle of Dialog containing progress bar is available
*/
if( BSP_SWITCH_STATE_PRESSED == BSP_SwitchStateGet( BSP_SWITCH_S1 ) &&
    hDialog)
{
    /*
      Get the handle of Progress bar
    */
    hItem = WM_GetDialogItem( hDialog, 0x801 );
    /*
      Update the value of Progress bar
```

```
    */
    PROGBAR_SetValue( hItem, value++);
}
```

GUI Resource Management

This section describes effective management of GUI resources of the SEGGER emWin Graphics Library.

Description

The various GUI resources are images, fonts, video streams, etc. The images can be of different types, such as uncompressed bitmap images, images compressed using lossless compression methods, or images compressed using lossy compression methods. Bitmap images can have different types of pixel color formats, such as raw BGR (Blue, Green, and Red) 888, or palletized formats. The bitmap formats containing the raw pixel color format BGR888 will take 1 byte for each color, and eventually, will take 3 bytes for each pixel. Based on the image size, the memory requirement will vary. If image size grows, the memory size required to store the image will also grow. The following example shows the calculation of memory size required for the image.

Considering image dimensions in pixels with Image Width = 480 and Image Height = 272, if the color format is BGR888, the size of memory required to store the pixel data will be: Image Size (in bytes) = Image Width x Image Height x Bytes Per Pixel = 480 x 272 x 3 = 391680 bytes. Please note that in this calculation the image header size is ignored.

The size of Image is accountable considering the RAM memory size of an embedded system device, which could be in form of 10s of KBytes. There are different methods to handle this scenario, as follows:

a) Reduce the size of Image by changing the color format

b) Reduce the size of Image by applying compression

c) Moving Image to Program Flash memory

d) Moving Image to external Flash memory

Reducing Image Size by Changing the Color Format

In this method, the color format of Image is modified to reduce the size of the image. As seen in the previous formula to calculate the Image size, one of the contributing factors to the image size is: Bytes Per Pixel. To be able to reduce the number of bytes required for storing one pixel, Image size can be reduced. For example, with color format BGR888, the image size in bytes is calculated as 391680 bytes. The same image with a color format of RGB565 will take 261120 bytes. Please note for color format RGB565 it takes 2 bytes for each pixel. Similarly, further reduction in the number bytes required for each pixel, the image size will reduce in proportion. Also note for palletized formats, a few more bytes are required to store the color table.

Please note that a reduction in the number of bytes required per pixel will reduce the number of colors from image and will affect the image quality, as well. The reduction of the number of colors can be decided based on the number of different colors available in the image and size of the image. For smaller images, the reduction in the number of colors from Image may not be visible, allowing reduction in memory requirements for storing such an image.

SEGGER emWin provides the Bitmap Converter utility to achieve the change in color format and saving the image as a C file. The C file can be further linked with the application code. The SEGGER emWin Graphics Library supports drawing of images edited by the Bitmap Converter using the library APIs.

For more information on using the Bitmap Converter to change the image format and on the APIs to access the image data, refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

Reduce Image Size by Applying Compression

There are various compression algorithms available, which can help reduce the size of the image. The size of the image is reduced by compressing the image data. The compression can be lossless compression or lossy compression. The Bitmap Converter utility can apply the lossless compression algorithm, such as RLE, and the lossy compression algorithm, such as JPEG. The original image size to compressed image size ratio (compression ratio) varies with the image pixel data and applied algorithm. The JPEG algorithm will be more efficient in terms of the compression ratio, but will be less efficient in terms of image quality, as the JPEG algorithm is a lossy compression algorithm. The SEGGER emWin Graphics Library supports drawing of RLE or JPEG compressed images by using suitable library APIs.

For more information on using the Bitmap Converter utility to change the image format, refer to the *"emWin Graphic Library with Graphical User Interface User and Reference Guide"*, document UM03001, which is available for download from the SEGGER website at: https://www.segger.com/downloads/emwin.

Moving an Image to Program Flash Memory

The static image or font data can be placed in the directly accessible memory such as program Flash. The data is placed in directly accessible memory such as program Flash using compiler attribute commands or in some cases by defining static data as const.

The XC32 compiler for PIC32 devices places the data into directly accessible program flash memory if the data is defined as const. The only disadvantage in this case is that the program flash size is now reduced by the size of image or font data. By default, SEGGER emWin tools define the image or font data as const or as GUI_CONST_STORAGE. The SEGGER emWin library for PIC32 defines the GUI_CONST_STORAGE as const.

wolfSSL Library Help

This section provides information on the wolfSSL Library.

Introduction

This topic provides an overview of the wolfSSL Library in MPLAB Harmony.

Description

The wolfSSL embedded SSL library (formerly CyaSSL) is a lightweight SSL/TLS library written in ANSI C and targeted for embedded, RTOS, and resource-constrained environments - primarily because of its small size, speed, and feature set. It is commonly used in standard operating environments as well because of its royalty-free pricing and excellent cross platform support. wolfSSL supports industry standards up to the current TLS 1.2 and DTLS 1.2 levels, is up to 20 times smaller than OpenSSL, and offers progressive ciphers such as ChaCha20, Curve25519, NTRU, and Blake2b. User benchmarking and feedback reports dramatically better performance when using wolfSSL over OpenSSL.

More Information

For known issues and additional details about this release, please see the README file in `<install dir>/third_party/tcpip/wolfssl`.

Product information is available from the wolfSSL website: https://www.wolfssl.com/wolfSSL/Products-wolfssl.html

For technical documentation, visit: https://www.wolfssl.com/wolfSSL/Docs.html

Additional information is also available from the Microchip Third-Party Software Stacks web page: http://www.microchip.com/devtoolthirdparty/

Index