MPLAB® Harmony Help - MPLAB Harmony Configurator Developer's Guide

MPLAB Harmony Integrated Software Framework v1.11

MPLAB Harmony Configurator Developer's Guide

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

CONFIG_DEVICE

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

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

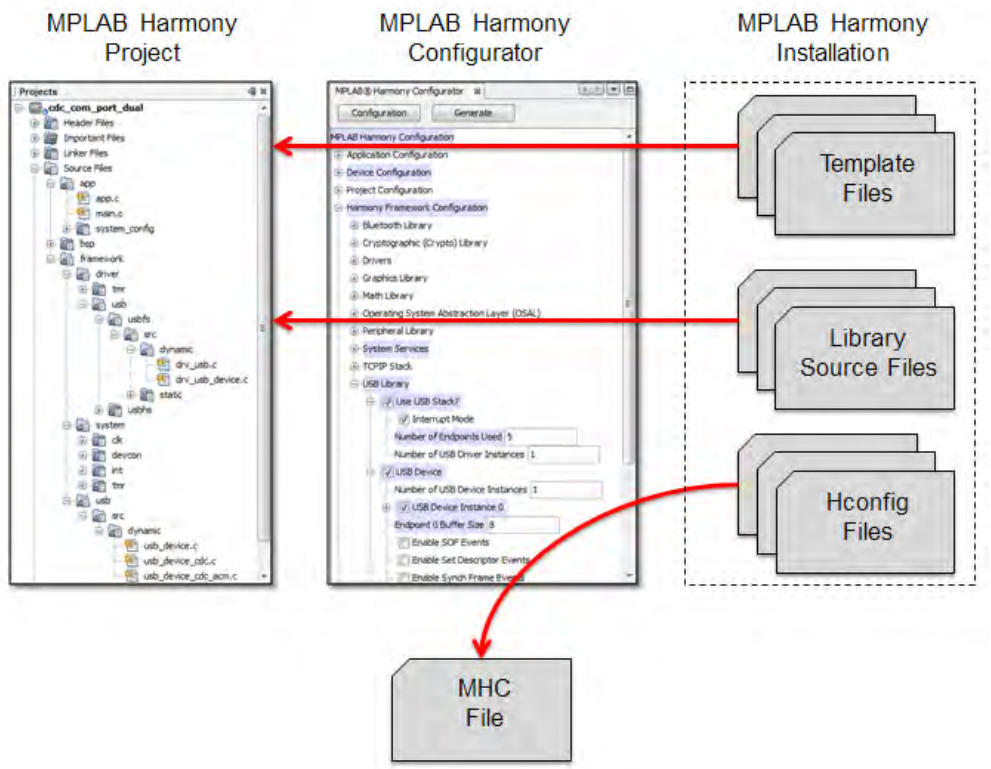
```
<#if CONFIG_DEVICE == “PIC32MZ2028ECM144”>
  … perform device-specific code …
<#/if>
```

Introduction

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

Description

When installed into MPLAB X IDE, the MHC plug-in provides a "New MPLAB Harmony" project wizard and a graphical user interface for configuration of MPLAB Harmony projects. When used, it generates (or updates) a project outline, including the C-language main function and system configuration files and stores the project configuration selections for later retrieval, modification, and sharing. To do this, the MHC utilizes a completely data driven method for defining the configuration options presented to the user and a template driven method for generating the source code, as illustrated in the following diagram.



Libraries are primarily provided in the MPLAB Harmony installation in source form. Each library provides an hconfig file and a set of template (`.ftl`) files. The hconfig files are text files that use an extended version of the Linux Kconfig grammar to define the configuration options available for the associated library and to identify source files, dependencies, and help content. When launched from within MPLAB X IDE, the MHC reads the hconfig files and presents the libraries and options to the user for selection and configuration in a graphical tree-based format similar to the Linux Xconfig utility.

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

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

Adding New Libraries

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

Description

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

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

These steps are described in detail in the following sections.

Developing a Library That is Compatible With MPLAB Harmony

This provides information on compatibility.

Description

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

Developing a New hconfig File

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

Description

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

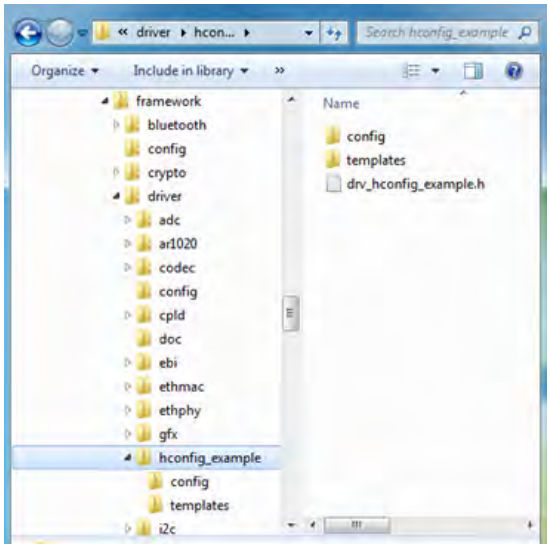
- Step 1: Create the File and Insert it into the hconfig Hierarchy
- Step 2: Create a Menu Item for the Module in the Driver Framework Tree
- Step 3: Creating Configuration Options
- Step 4: Use Dependencies
- Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another
- Step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

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

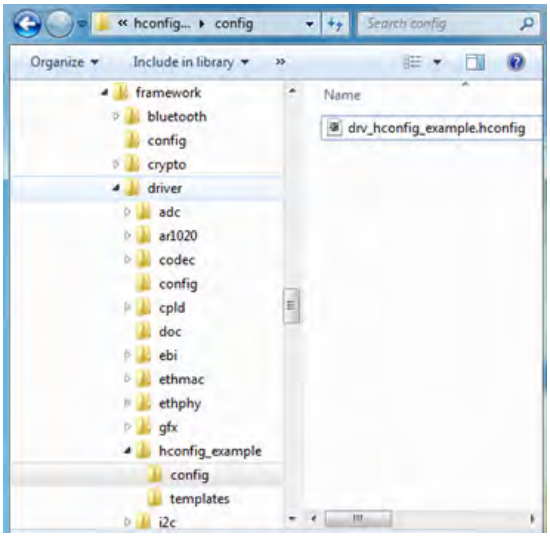
This topic describes how to create the hconfig file and insert it into the hconfig hierarchy.

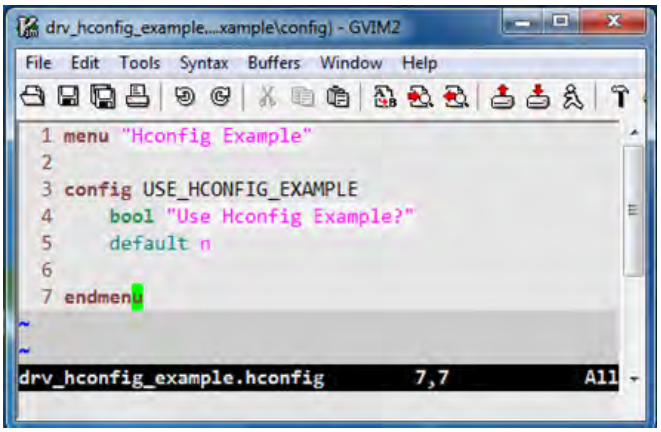
Description

Our module example will be a MPLAB Harmony driver named "hconfig_example", and will be inserted into the `<$HARMONY_VERSION_PATH>/framework/driver` directory.



Let's create an hconfig file for the hconfig_example driver, and put it in the `config` folder. For now, all it will do is create a menu entry and a single Boolean config item "Use Hconfig Example?". Note that by default, the driver is not selected.





Now we need to insert our hconfig file into the hconfig tree hierarchy so it will be invoked when we run MHC.

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

```
menu "Drivers"

config DRIVER
    bool

source "$HARMONY_VERSION_PATH/framework/driver/ar1020/config/drv_ar1020.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/ebi/config/drv_ebi.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/ethmac/config/drv_ethmac.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/ethphy/config/drv_ethphy.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/gfx/config/drv_gfx.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/i2s/config/drv_i2s.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/ic/config/drv_ic.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/nvm/config/drv_nvm.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/oc/config/drv_oc.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/pmp/config/drv_pmp.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/rtcc/config/drv_rtcc.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/sdcard/config/drv_sdcard.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/spi/config/drv_spi.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/tmr/config/drv_tmr.hconfig"
source "$HARMONY_VERSION_PATH/framework/driver/usart/config/drv_usart.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/usb/config/drv_usb.hconfig"
#source "$HARMONY_VERSION_PATH/framework/driver/wifi/config/drv_wifi.hconfig"
endmenu

ifblock DRIVER
file DRIVER_H "$HARMONY_VERSION_PATH/framework/driver/driver.h"
endif
```

Added

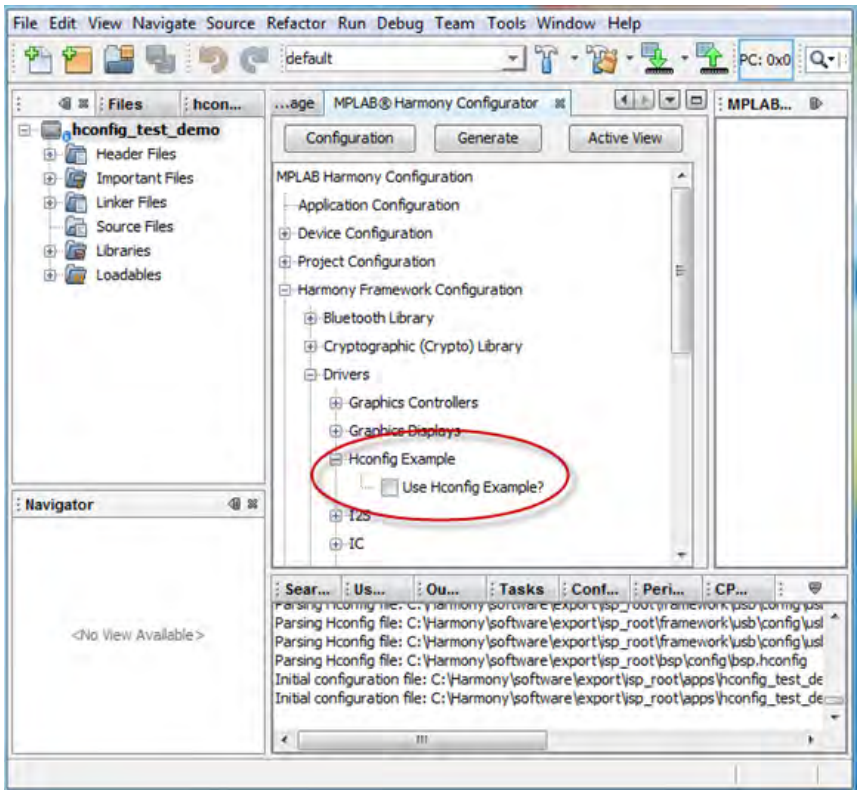
Note that all hconfig files are included recursively by the top-level hconfig file in the application's firmware directory. The entire hconfig tree is parsed when MHC is invoked and when a configuration change is made, so the relative placement of configuration options only affects the menu structure.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

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

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

Description

Let's create a demonstration application and see if our driver config appears.



Step 3: Creating Configuration Options

This topic describes adding menu configuration options

Description

Now let's add some config options.

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

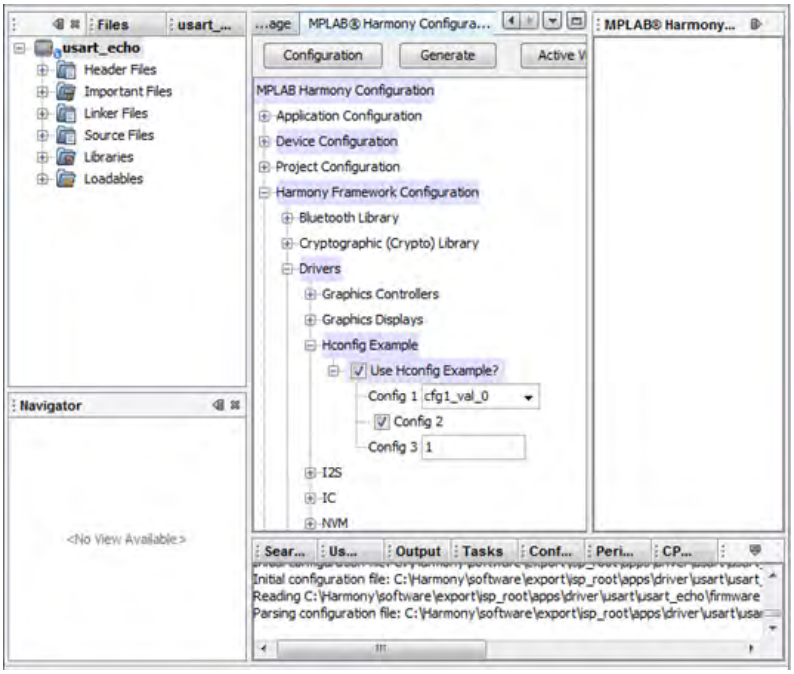
```
menu "Hconfig Example"

config USE_HCONFIG_EXAMPLE
    bool "Use Hconfig Example?"
    default n

# 1. A config option selected from a dropdown menu.
enum CFG1_VAL
    "cfg1_val_0"
    || "cfg1_val_1"
    || "cfg1_val_2"

config CFG1
    depends on USE_HCONFIG_EXAMPLE
    string "Config 1"
    range CFG1_VAL
    default "cfg1_val_0"

# 2. A boolean config value
config CFG2
    depends on USE_HCONFIG_EXAMPLE
    bool "Config 2"
    default y

# 3. An integer config whose default value depends on CFG1.
config CFG3
    depends on USE_HCONFIG_EXAMPLE
    int "Config 3"
    range 1 3
    default 1 if CFG1 = "cfg1_val_0"
    default 2 if CFG1 = "cfg1_val_1"
    default 3 if CFG1 = "cfg1_val_2"

endmenu
```

Step 4: Use Dependencies

This topic describes use dependencies.

Description

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



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

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

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

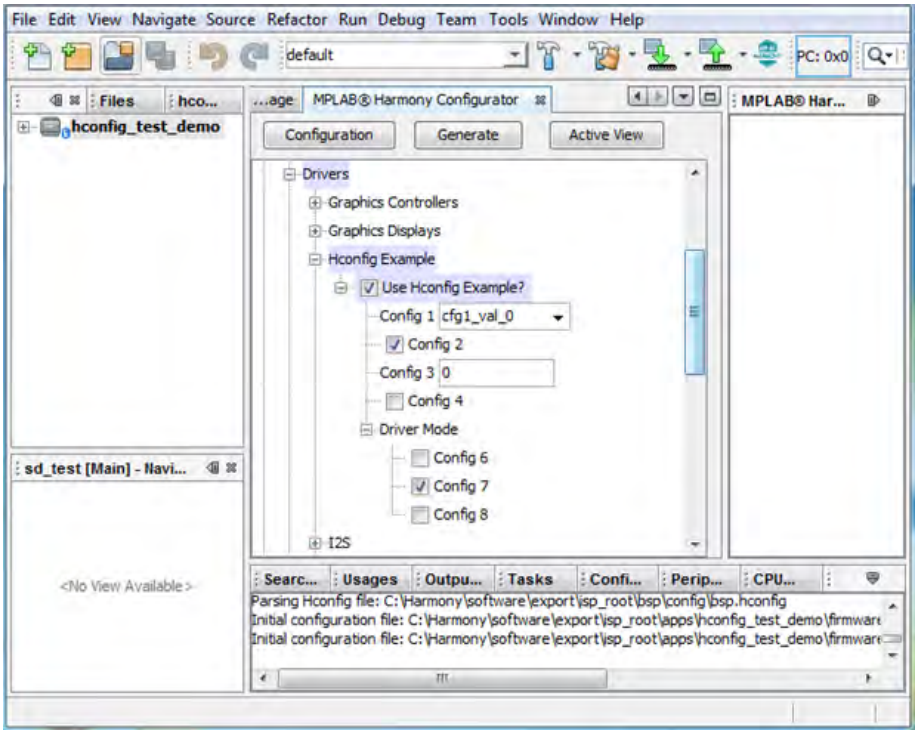
Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another

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

Description

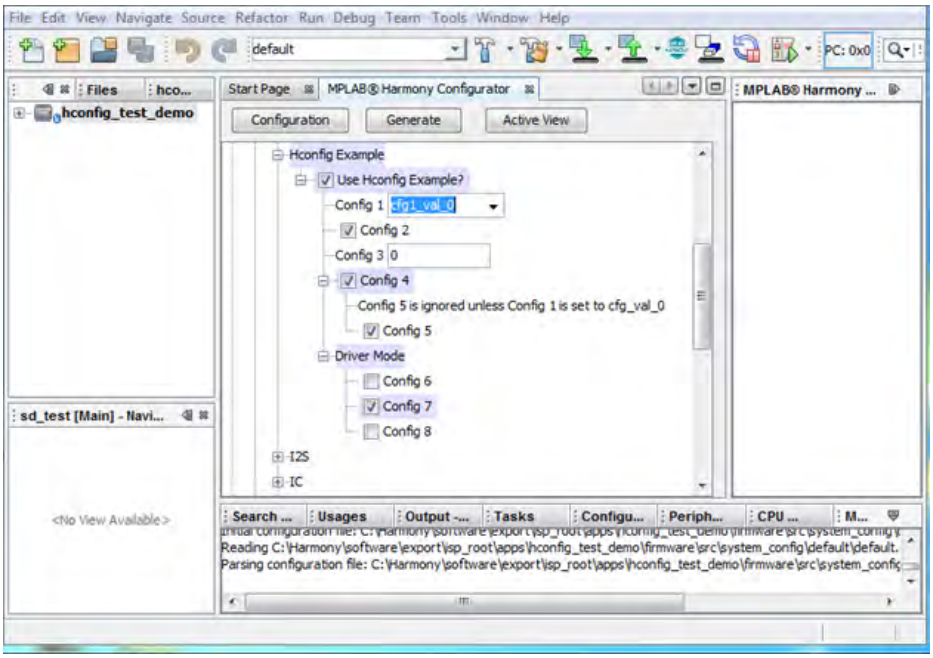
You can make config options mutually exclusive with the "choice" statement. This is useful for modules that can be configured to operate in different modes. A choice block requires a prompt, which is displayed in the hconfig tree. Choice blocks can optionally have a default option and dependencies. If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

```
45 choice
46     prompt "Driver Mode"
47     depends on USE_HCONFIG_EXAMPLE
48     default CFG7
49 config CFG6
50     bool "Config 6"
51
52 config CFG7
53     bool "Config 7"
54
55 config CFG8
56     bool "Config 8"
57 endchoice
```



Comments can be displayed in the menu with the "comment" statement. Comments can also have dependencies.

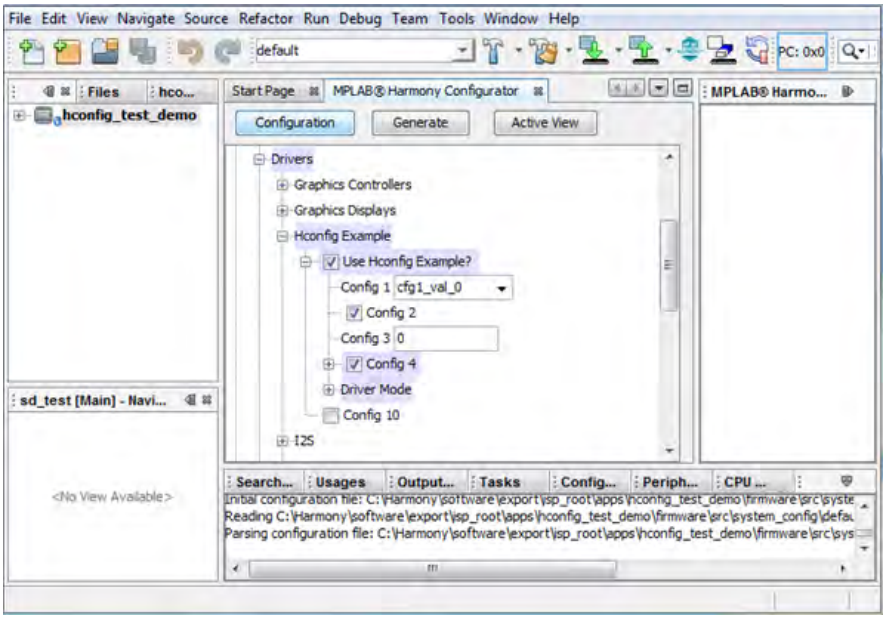
```
39 comment "Config 5 is ignored unless Config 1 is set to cfg_val_0"
40     depends on CFG4
41 config CFG5
42     depends on USE_HCONFIG_EXAMPLE && CFG2
43     depends on CFG4
44     bool "Config 5"
45     default y if (CFG1 = "cfg1_val_0") && CFG4
46
```

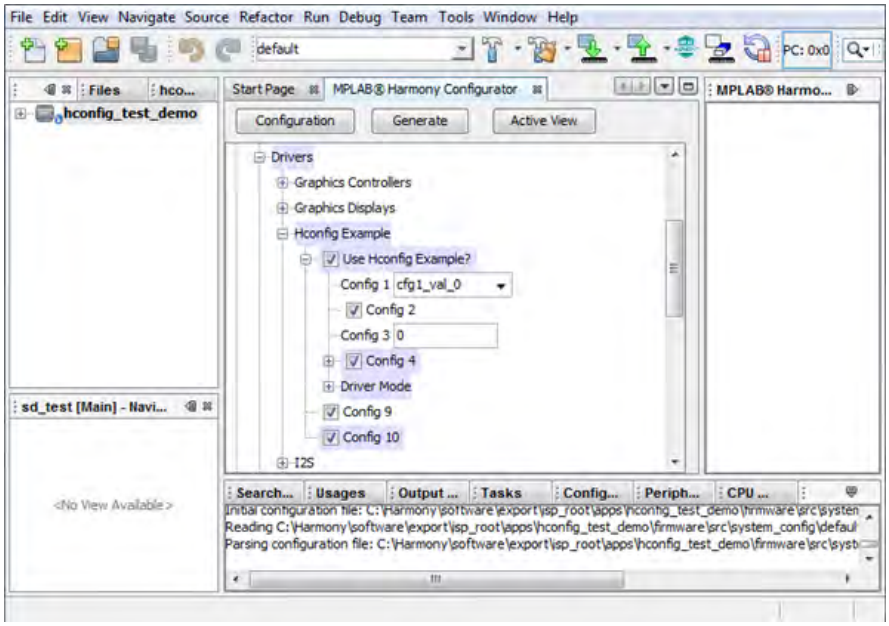


The "select" statement is used to select or enable a config option based on another config option. This is often used to enable a module that may be required by multiple modules. An example is the Interrupt System Service, which is used by many drivers and system services.

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

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





Step 6: Sourcing hconfig Files

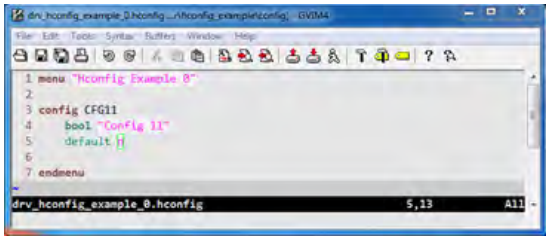
This topic describes how to source an hconfig file from another hconfig file.

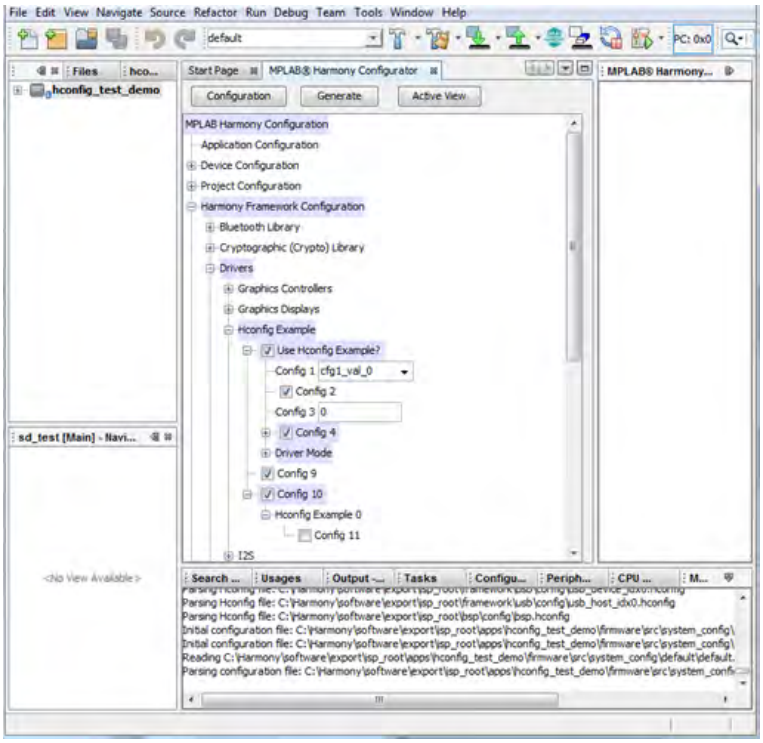
Description

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

Enclosing a source statement within an "ifblock" will apply the dependency to all config options within the sourced file. In the example shown below, all config options in the `drv_hconfig_example_0.hconfig` file are dependent on CFG10. All Ifblock statements must be terminated with `endif`.

```
74 ifblock CFG10
75 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_example_0.hconfig"
76 endif
77
```





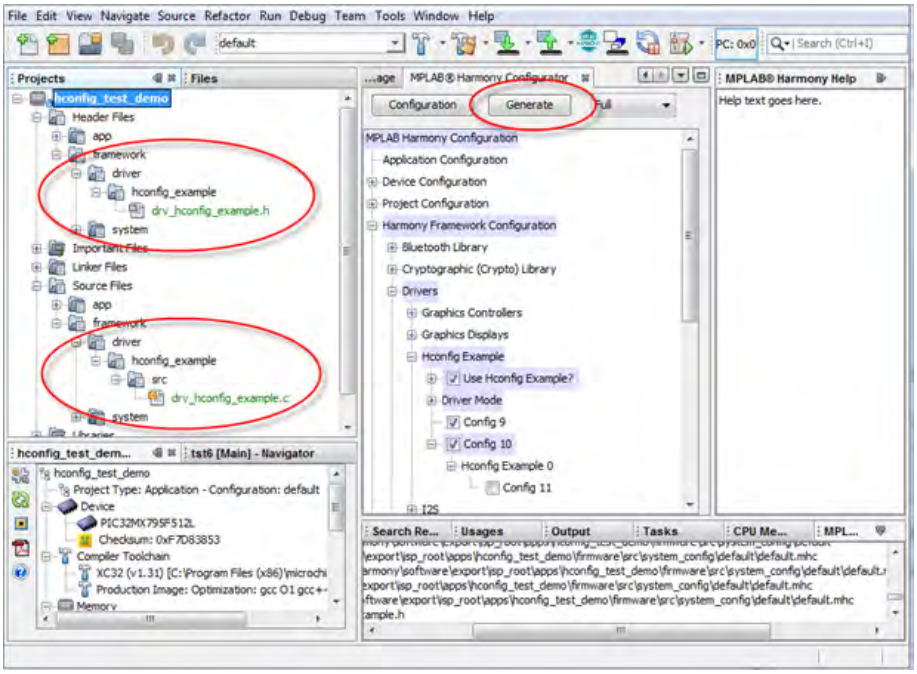
Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement

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

Description

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

```
82
83 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl" 3 instances
84
85 file DRV_HCONFIG_EXAMPLE_H "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/drv_hconfig_example.h" to
   "$PROJECT_HEADER_FILES/framework/driver/hconfig_example/drv_hconfig_example.h"
86 file DRV_HCONFIG_EXAMPLE_C "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/src/drv_hconfig_example.c"
   to "$PROJECT_SOURCE_FILES/framework/driver/hconfig_example/src/drv_hconfig_example.c"
87
```



Step 8: Add Help Links to Configuration Options

This topic provides an example for adding Help links.

Description

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

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

Step 9: Create Multiple Module Instances

This topic describes the creation of multiple instances.

Description

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

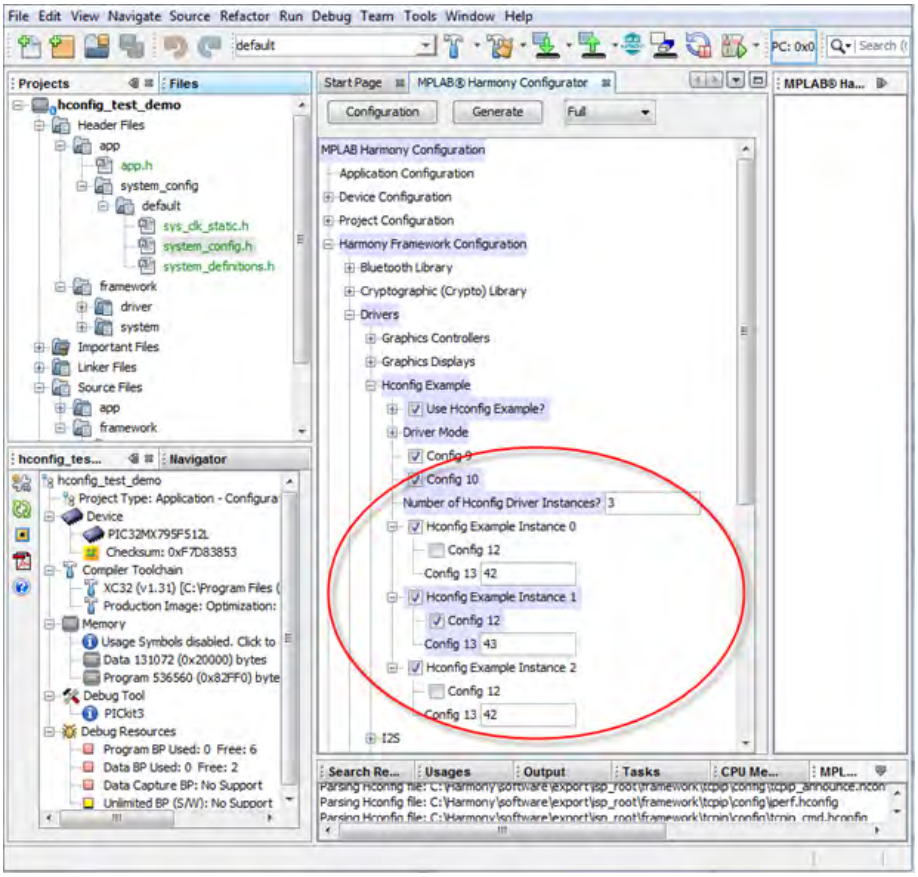
The instance template is sourced like a normal hconfig file, but with the keyword "instances" preceded by the maximum number of instances supported. A configuration option is added to allow the user to select the number of instances actually configured and instantiated.

```
77 config DRV_HCONFIG_INSTANCES_NUMBER
78     depends on USE_HCONFIG_EXAMPLE
79     int "Number of Hconfig Driver Instances?"
80     default 1
81     range 1 3
82
83 source "$HARMONY_VERSION_PATH/framework/driver/hconfig_example/config/drv_hconfig_idx.ftl" 3 instances
84
```

The FreeMarker template is a marked-up hconfig file that is processed through FreeMarker once for each instance. Each time it is processed, the `${INSTANCE}` variable is set to the instance number.

```
config DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE+1}
    bool
<#if INSTANCE != 0>
    default n if DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE} = n
</#if>
    default n if DRV_HCONFIG_INSTANCES_NUMBER = ${INSTANCE+1}
    default y

config DRV_HCONFIG_INST_IDX${INSTANCE}
    depends on USE_HCONFIG_EXAMPLE
<#if INSTANCE != 0>
                && DRV_HCONFIG_INSTANCES_NUMBER_GT_${INSTANCE}
</#if>
    bool "Hconfig Example Instance ${INSTANCE}"
    default y

ifblock DRV_HCONFIG_CFG12_IDX${INSTANCE}

config DRV_HCONFIG_IDX${INSTANCE}
    bool "Config 12"
    default n

config DRV_HCONFIG_CFG13_IDX${INSTANCE}
    int "Config 13"
    default 42

endif
```

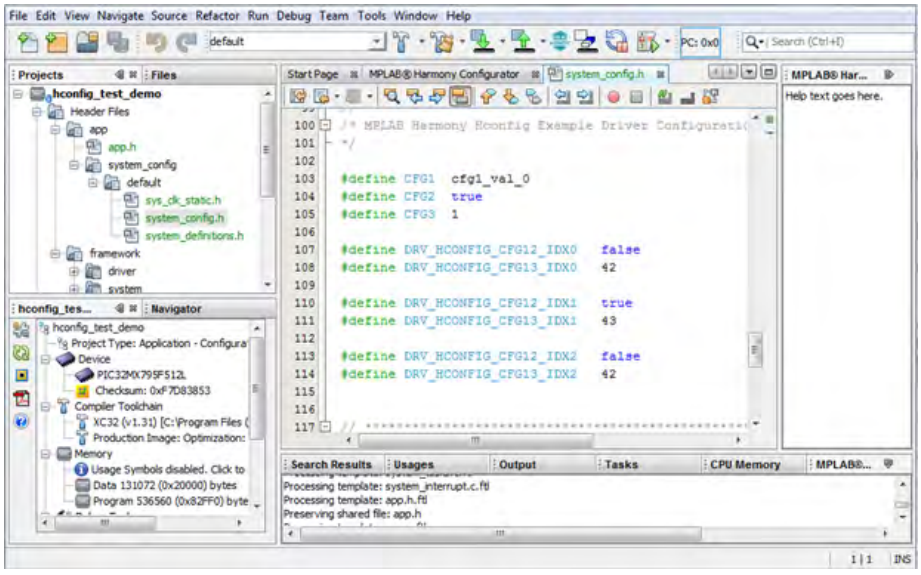
When MHC is run, the user is prompted for the number of instances. Configuration options for each instance are displayed in the menu.



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

```
<#if CONFIG_USE_HCONFIG_EXAMPLE == true>

// ******************************************************************************
/* MPLAB Harmony Hconfig Example Driver Configuration Options
*/

#define CFG1  ${CONFIG_CFG1}
<#if CONFIG_CFG2 == true>
#define CFG2  true
<#else>
#define CFG2  false
</#if>
#define CFG3  ${CONFIG_CFG3}

<#-- Instance 0 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX0 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX0 == true>
#define DRV_HCONFIG_CFG12_IDX0   true
<#else>
#define DRV_HCONFIG_CFG12_IDX0   false
</#if>
#define DRV_HCONFIG_CFG13_IDX0   ${CONFIG_DRV_HCONFIG_CFG13_IDX0}
</#if>

<#-- Instance 1 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX1 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX1 == true>
#define DRV_HCONFIG_CFG12_IDX1   true
<#else>
#define DRV_HCONFIG_CFG12_IDX1   false
</#if>
#define DRV_HCONFIG_CFG13_IDX1   ${CONFIG_DRV_HCONFIG_CFG13_IDX1}
</#if>

<#-- Instance 2 -->
<#if CONFIG_DRV_HCONFIG_INST_IDX2 == true>
<#if CONFIG_DRV_HCONFIG_CFG12_IDX2 == true>
#define DRV_HCONFIG_CFG12_IDX2   true
<#else>
#define DRV_HCONFIG_CFG12_IDX2   false
</#if>
#define DRV_HCONFIG_CFG13_IDX2   ${CONFIG_DRV_HCONFIG_CFG13_IDX2}
</#if>

</#if>
```

When the code is generated, code is generated for each instance.



Using the Set Statement

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

Description

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

Library C Selection and Configuration Menu Definition

```
# Library C Configuration
config USE_LIBRARY_C
    bool "Use Library C?"
    default n

menu "Configure Library C"
    depends on USE_LIBRARY_C

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

endmenu  # Configure Library C
```

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

Library A Selection and Configuration Menu Definition

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

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

menu "Configure Library A"
    depends on USE_LIBRARY_A

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

endmenu  # Configure Library A
```

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

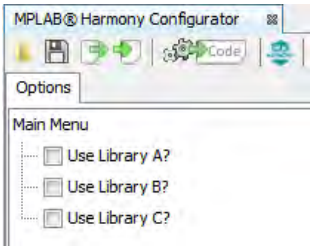
Library B Selection and Configuration Menu Definition

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

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

menu "Configure Library B"
    depends on USE_LIBRARY_B

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

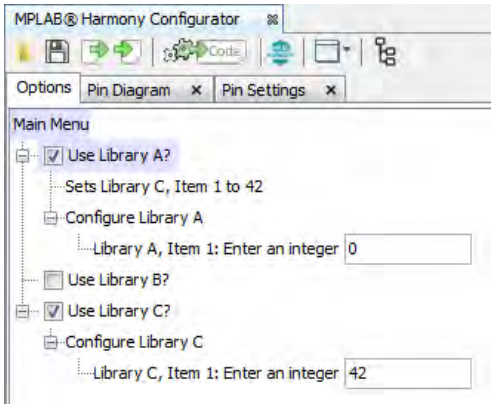
```
    default 0

endmenu  # Configure Library B
```

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

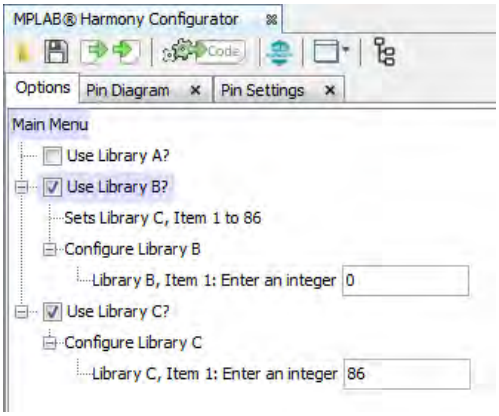
The following sequence of images illustrates the behavior of the MHC when the previous Hconfig code is used. Before any of these libraries have been selected, the MHC Options menu shows their Use Library options.



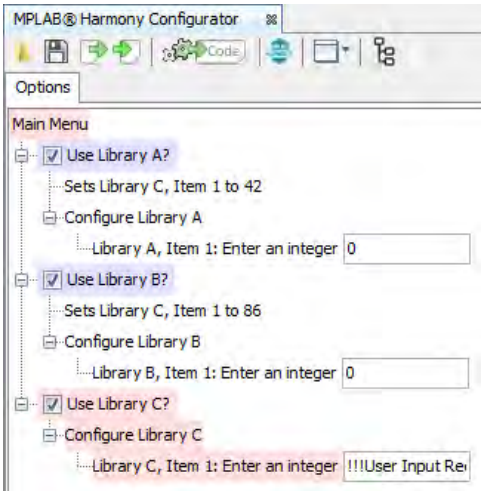
If library A is used (but not library B), the MHC automatically sets the value of library C's configuration item to 42.



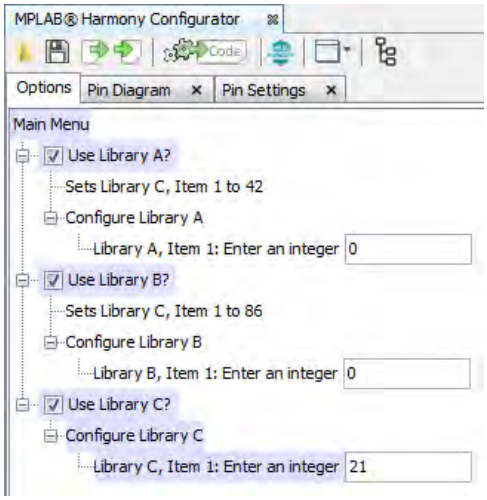
If library B is used (but not library A), the MHC automatically sets the value of library C's configuration item to 86.



However, if both library A and B are used, the MHC highlights the conflict in library C in red and requires the user to enter a value to resolve the conflict.



If the user then enters a value for library C's item 1, the MHC recognizes that the user has set that item's value and assumes that the conflict has been resolved.



It is important to understand that the MHC does not validate that the chosen value satisfies the requirements of both libraries A and B. It is up to the user to understand the requirements and select an appropriate configuration value.

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

hconfig Development Guidelines

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

Description

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

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

Driver hconfig contains:

```
config USE_DRV_TMR_NEEDED
    bool
config USE_DRV_TMR
    depends on HAVE_TMR
    bool "Use Timer Driver?"
    default y if USE_DRV_TMR_NEEDED
    default n
```

and the timer system service hconfig contains:

```
config USE_SYS_TMR
    bool "Use Timer System Service?"
    select USE_DRV_TMR_NEEDED
    default y if USE_SYS_TMR_NEEDED
    default n
```

Selecting the Timer System Service automatically selects the Timer Driver, by selecting `USE_DRV_TMR_NEEDED`, which (if selected) sets the `USE_DRV_TMR` default to 'y'.

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

Developing MPLAB Harmony FreeMarker Templates

This topic provides information on developing FreeMarker templates.

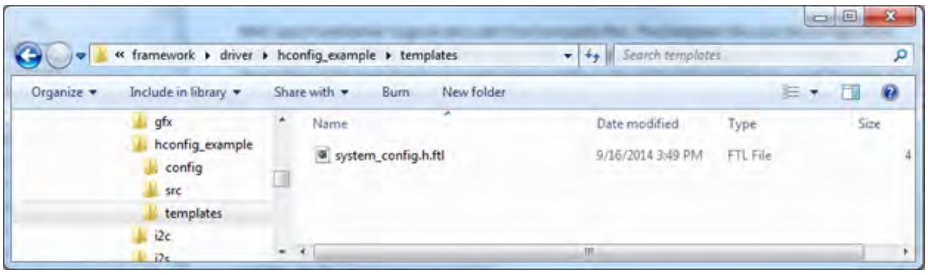
Description

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

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

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

First, we need to create the template for `system_config.h`. By convention, this file will be named `system_config.h.ftl`, and be placed in the `<module>/templates` directory.



Second, we need to implement the source code template, as shown by the following example.

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

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

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

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

Device Configuration

This topic describes the CONFIG_DEVICE hconfig symbol.

Description

CONFIG_DEVICE

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

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

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

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

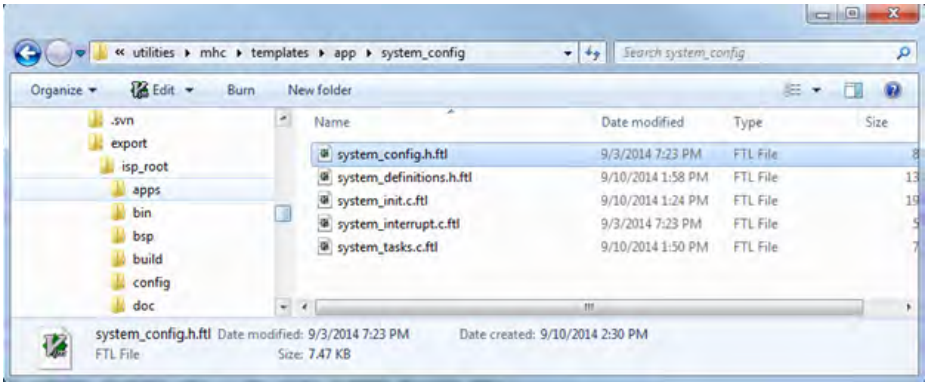
This topic describes how to insert a new FreeMarker template into the MPLAB Harmony top-level templates.

Description

To insert the template into MHC, we need to do one of two things:

- Use the "template" keyword in hconfig, or
- Include this template into another template

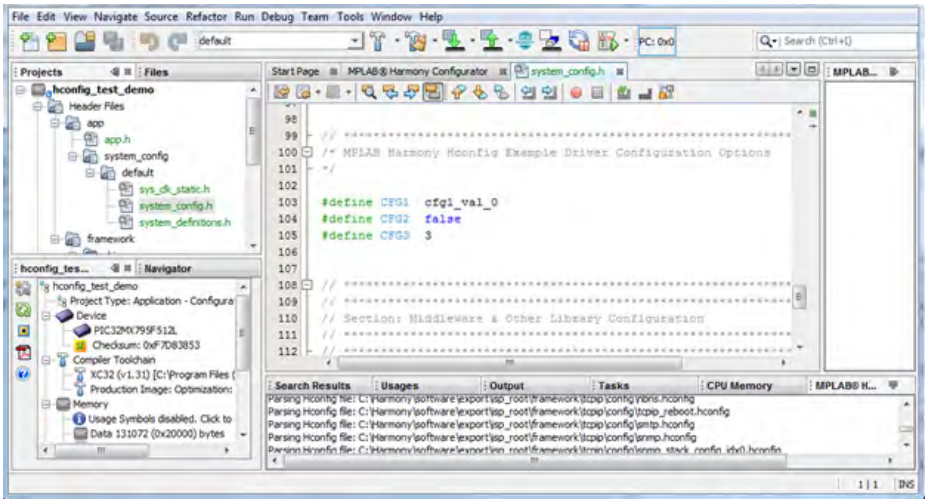
Since the `system_config.h` file draws config options from many templates, we will include our template in the top-level `system_config.ftl`. It is located in the `$HARMONY_VERSION_PATH/utilities/mhc/templates/app/system_config` directory.



This top-level template simply includes all of the module-specific templates that contribute to the `system_config.h` file. The included files are logically organized within the top-level template. For the following example, we will add our template in the driver configuration section.

```
118 // *****************************************************************************
119 // *****************************************************************************
120 // Section: Driver Configuration
121 // *****************************************************************************
122 // *****************************************************************************
123
124 <#if CONFIG_USE_DRV_TMR == true>
125 <#include "/framework/driver/tmr/templates/drv_tmr.h.ftl">
126 </#if>
127 <#if CONFIG_USE_DRV_USART == true>
128 <#include "/framework/driver/usart/templates/drv_usart.h.ftl">
129 </#if>
130 <#if CONFIG_USE_HCONFIG_EXAMPLE == true>
131 <#include "/framework/driver/hconfig_example/templates/system_config.h.ftl">
132 </#if>
```

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



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

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

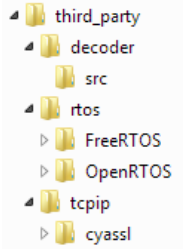
It will be necessary to modify each of the above templates to include the module-specific templates for any new libraries. It is also necessary to carefully review each top-level template to determine the appropriate location at which to include the module-specific templates and then test the code that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

Installing a New Library into MPLAB Harmony

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

Description

Within the MPLAB Harmony installation, you will find a `third_party` top-level folder, as shown in the follow figure. Within that folder, third-party code is organized by its purpose. If an appropriate sub-folder exists, create a directory named for your company or your product within that folder and copy your source installation files and folders into it. Your source tree should include the necessary hconfig and FreeMarker templates (as described previously) and Help content (described in the next section) to support your library in the MHC.



By default, MPLAB Harmony installs into a version-specific folder (`C:\microchip\harmony\<version>` on a Windows personal computer or `~/microchip/harmony/<version>` on a Mac or Linux computer). Therefore, when you install a newer version of MPLAB Harmony, it is very likely that you will need to reinstall your library. If your library is not part of the MPLAB Harmony installation, providing an installer that automates the process of copying your installation files into the new MPLAB Harmony installation folder and inserting the hconfig, FreeMarker templates, and help files into the new hierarchies will be a necessity.

Inserting New Library Help into the MPLAB Harmony Documentation Index

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

Description

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

HTML Browser Used by MHC

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

Description

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

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

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

Help Documentation Methods

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

Description

Two methods exist for creating Help content:

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

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

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

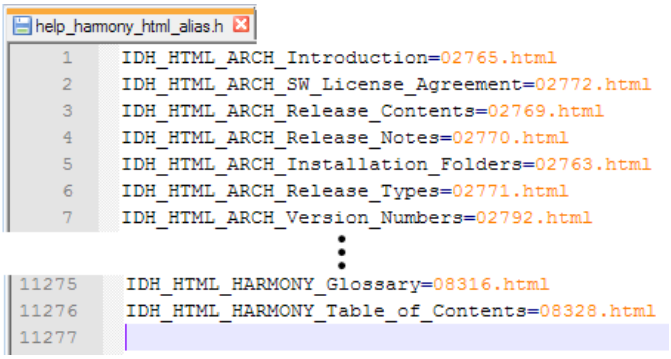
HTML Alias Header File

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

Description

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

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



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

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

Where:

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

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

`IDH_HTML_IBC_TOOL_New_Tool=IBC_new_tool_help.html`

Notes:

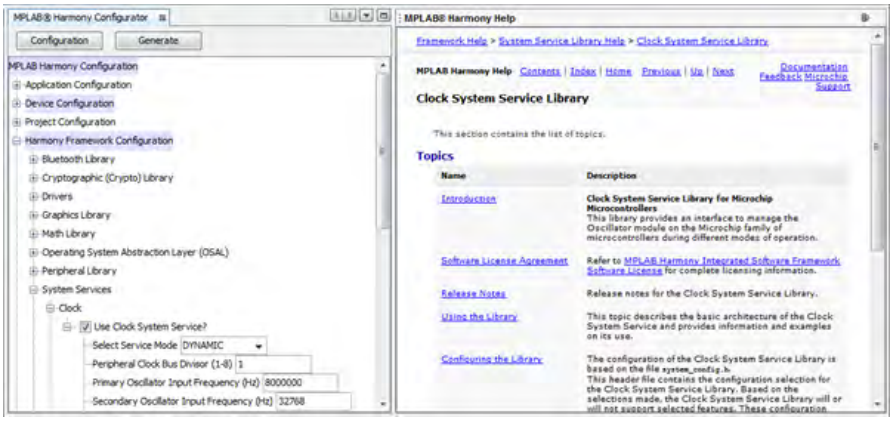
1. The content of your HTML files must be compatible with HTML Version 3.2 or older. The HTML browser used by MHC cannot process HTML tags that are newer than V3.2.
2. You must ensure that any entries added to the existing alias header file are unique from all other entries.
3. When choosing the TopicTitle,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

hconfig Files

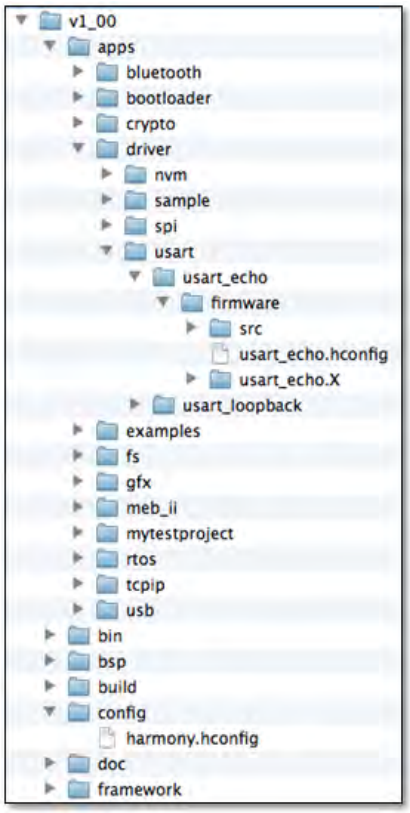
This topic provides information and the location of hconfig files.

Description

The hconfig file tree represents a hierarchy of configuration options presented, with associated Help documentation, by the MHC so that the user can select and configure the desired build options.



Within the MPLAB Harmony installation, hconfig files are kept in the `config` folder at each level in the installation hierarchy that requires them, with one exception. The root of the hconfig file tree is an application-specific file (`<application-name>.hconfig`) that is generated in the project's `firmware` folder. It is not a predefined file. This generated hconfig root file enables the creation of application-specific options if desired (see **Note**). The root file defines the "MPLAB Harmony configuration" main menu item and then includes (AKA "sources") the installation's top-level hconfig file (`<install-dir>/config/harmony.hconfig`) for the installed libraries and templates (as illustrated in the following figure). The top-level hconfig file then includes (sources) the next level of hconfig files in the hierarchy, each of which includes the next level, and so on, down to the individual library hconfig files, which form the "leaves" of the hconfig tree.



Important! The MHC does not currently provide a graphical method of creating application-specific configuration options. It is therefore necessary to manually edit the application-specific hconfig file to create application-specific configuration options that will appear in the MHC tree.

Kconfig Language Specification

This topic provides information on obtaining the Kconfig Language Specification.

Description

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

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

hconfig Language Extensions (Kconfig+)

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

"enum"

This topic describes the "enum" extension.

Description

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

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

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

Example:

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

"range"

This topic describes the "range" extension.

Description

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

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

Example:

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

"template"

This topic describes the "template" extension.

Description

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

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

Example:

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

"file"

This topic describes the "file" extension.

Description

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

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

Example:

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

"library"

This topic describes the "library" extension.

Description

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

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

Example:

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

"execute"

This topic describes the "execute" extension.

Description

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

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

Example:

```
execute GDDX_PLUGIN GDDX if USER_EXECUTES_GDDX
```

"persistent"

This topic describes the "persistent" extension.

Description

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

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

Example:

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

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

hconfig Environment Variables

This topic provides information on the hconfig environment variables.

Description

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

The hconfig environment variables include:

Variable Name	Description
`$HARMONY_VERSION_PATH`	Physical pathname to the MPLAB Harmony directory (i.e., `C:/microchip/harmony/<version>`).
`$PROJECT_NAME`	MPLAB X IDE main project name when the Generate option was selected.

$PROJECT_FIRMWARE_DIRECTORY	Physical path to the project's firmware directory.
$PROJECT_BSP_DIRECTORY	Physical path to the project's bsp directory.
$PROJECT_HEADER_FILES	Logical path to the project header files.
$PROJECT_SOURCE_FILES	Logical path to the project source files.
$CONFIGURATION	MPLAB X IDE project configuration name.
$DEVICE	MPLAB X IDE project device name.
$OS_NAME	Name of the operating system on the computer running MPLAB X IDE.

hconfig Configuration Variables

This topic provides information on the hconfig configuration variables.

Description

The hconfig configuration variables include:

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

Complete hconfig Grammar Definition

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

Description

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

Statement:
   CommonStatement
 | MainmenuStmt
 | MenuStmt
 | ChoiceStmt
;

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

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

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

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

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

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

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

MainmenuStmt:
    'mainmenu' value=STRING
;

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

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

Dependency:
    'depends on' depexpr = Expr
;

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

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

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

Orstring:
    '||' value=STRING
;

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

```
;

ChoiceOption:
    Optional | Prompt | Dependency | Default
;

Optional:
    Optional='optional'
;

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

NotLiteral:
    '!' (NotLiteral = ID)
;

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


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

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

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

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

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

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

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

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

MHC Files

This topic provides an example MHC file.

Description

The MHC stores the user's selections in an MHC file. An MHC file is created for each configuration, named using the configuration name provided by the MPLAB-IDE, and located (by default) in the configuration-specific `system_config` folder within the `src` folder in the default MPLAB Harmony project.

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

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

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

`CONFIG_<config option>=<value>`

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

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

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

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

MHC Configuration File

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

Description

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

The information that this file currently contains includes:

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

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

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

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

BSP XML Specification

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

Description

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

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

File Example

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

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

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

The function node must have these required attributes:

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

The function node may also have these attributes:

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

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

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

Adding New BSPs

This section provides information adding a new BSP.

Updating the BSP hconfig File

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

Description

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

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

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

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

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

```
config BSP_MYBOARD
   depends on USE_BSP
   depends on DS60001320    # Microchip document number for devices that can use this BSP
   select BSP_TRIGGER
   select USE_GFX_STACK
   bool "BSP for my board"  # Ensure you are using the correct quotation marks to prevent errors
```

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

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

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

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

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

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

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

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

MPLAB Harmony Configurator Plug-ins

Describes the clock screen system plug-in interface.

Description

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

System Requirements

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

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

NetBeans Project Setup

Java Dependencies

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

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

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

Plug-in Installation:

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

1. *Plug-in file*. NetBeans will produce a JAR file of your plug-in. This file must be copied to the MPLAB Harmony framework folder: `<install-dir>\utilities\mhc\plugins\clock`.
2. *HConfig* - MPLAB Harmony Configurator relies on the HConfig tree to tell it which clock plug-in file to load. Often this is processor-specific. To get your plug-in loaded, you must edit the MPLAB Harmony framework file: `<install-dir>\framework\config\framework.hconfig`.
   - The string symbol `SYS_CLK_MANAGER_PLUGIN_SELECT` must be defined. This symbol value must have the following format: `($JAR_FILE_NAME):($CLOCK_MODEL_CLASS)`, where:
     - `($JAR_FILE_NAME)` - the name of your plug-in JAR file without the `.jar` file extension
     - `($CLOCK_MODEL_CLASS)` - the name of the class in your plug-in that inherits from the ClockModel base class
   - For example, to load the MX1 clock screen the symbol would be set to: `mx1:MX1ClockModel`

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

Debugging Plug-ins

MPLAB X IDE Configuration

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

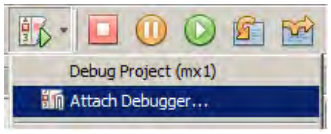
1. Open the following file in a text editor: `($MPLABX_ INSTALLATION_PATH)/($MPLABX_VERSION)/etc/mplab_ide.conf`.
2. Locate the configuration entry `default_options`. Add the following text to the line (without a line break):

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

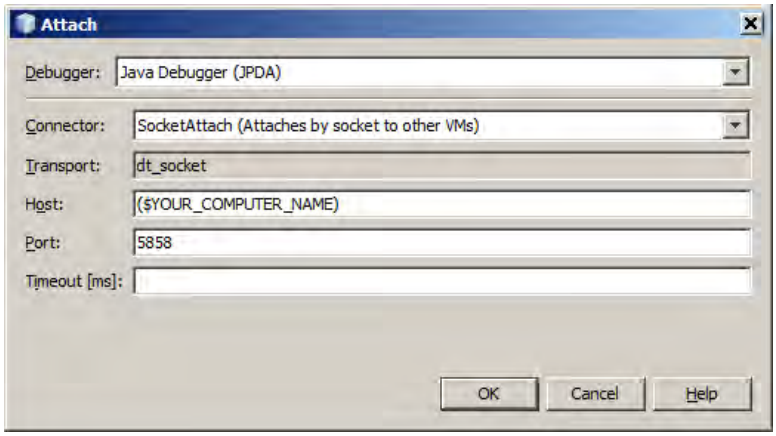
This instructs MPLAB X IDE to allow debugging over the socket 5858.

NetBeans Configuration

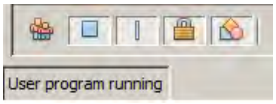
1. To attach to MPLAB X IDE, right-click the Debug Project drop-down menu and select **Attach Debugger**.



2. Configure the Attach dialog, as shown in the following figure.



3. If MPLAB X IDE is running and you configured everything properly, the message “User program running” should appear in the lower-left corner of NetBeans.



You can now set breakpoints in your plug-in code and debug as normal. If you receive the message *Connection Refused*, this indicates that something has been misconfigured.

Pin Manager Development

Provides details on pin manager development.

Description

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

The following table provides common terms and their descriptions.

Term	Description
Pin Manager	A system for configuring component I/O pins.
Pin Diagram	A visual representation of a component.
Pin Table	A matrix-based system for assigning functions to pins.
Pin	A single I/O interface on a component.
Package	A physical pin layout for a component. Components may come in several packages.
Function	A processing capability that a component supports (e.g., UTX1).
Module	Designates a set of related functions (e.g., UART1).
Component	A discreet part number (e.g., PIC32MX110F016B).
Family	Designates a superset of components (e.g., PIC32MZEF).

File Parsing

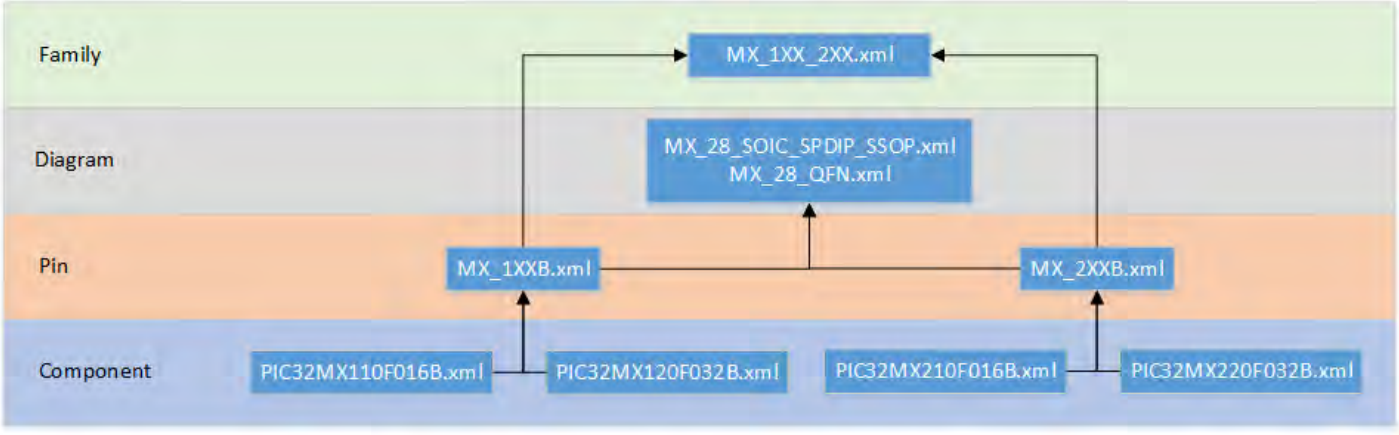
The pin manager is responsible for parsing through a set of XML files for the purpose of building a component pin state. These data files are located in the MPLAB Harmony framework within `<install-dir>/utilities/mhc/pin_xml`.

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

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

XML File Hierarchy

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



Detailed File Descriptions

Component

The component file only has one entry that maps the selected component to its pin map file.

```
<component device="PIC32MX110F016B" pins="MX_1XXB" />
```

Pin

The `pinfile` root node of the pin file maps the pin file to the family file:

```
<pinfile family="MX_1XX_2XX">
```

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

One or more `package` nodes will be listed inside the `packages` node.

```
<packages>
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="1" name="SOIC" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="2" name="SPDIP" />
<package diagram="MX_28_SOIC_SPDIP_SSOP" id="3" name="SSOP" />
<package diagram="MX_28_QFN" id="4" name="QFN" />
</packages>
```

`Package Node`

The package node description is as follows:

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

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

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

Pin Node

The `pin` node description is as follows:

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

Diagram

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

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

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

controlling the blank space around the diagram.

Shape Nodes

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

`line` – A line:

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

`circle` – A circle:

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

`rect` – A rectangle centered inside the diagram screen:

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

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

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

`string` – A text string:

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

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

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

- `size` – (attribute) the font size to use
- `margin` – (attribute) the amount of space to pad between the strings

Layout Node

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

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

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

Row Layout

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

Row Node

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

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

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

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

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

`pin_width` – describes the width of a pin cell

`pin_height` – describes the height of a pin cell

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

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

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

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

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

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

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

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

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

Grid Layout

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

An example of a `grid` layout is as follows:

```
<layout type="grid" pin_margin="40" pin_name_margin="0" pin_rows="18" pin_cols="18" pin_radius="11" />
```

The attributes of a grid layout are as follows:

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

Family

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

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

Groups

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

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

PPS Input Pins

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

PPS Output Pins

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

This data is described in XML format as follows:

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

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

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

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

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

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

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

Modules

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

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

An example of a module definition is as follows:

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

XML Specification Descriptions

Detailed descriptions of the module XML specification are as follows:

module – (node):

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

function – (node):

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

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

An example of a BSP module is as follows:

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

Index

"

"enum" 24
"execute" 25
"file" 24
"library" 25
"persistent" 25
"range" 24
"template" 24

A

Adding New BSPs 33
Adding New Libraries 4

B

BSP XML Specification 32

C

Complete hconfig Grammar Definition 26

D

Developing a Library That is Compatible With MPLAB Harmony 4
Developing a New hconfig File 4
Developing MPLAB Harmony FreeMarker Templates 18
Device Configuration 19

H

hconfig Configuration Variables 26
hconfig Development Guidelines 17
hconfig Environment Variables 25
hconfig Files 23
hconfig Language Extensions (Kconfig+) 24
Help Documentation Methods 21
HTML Alias Header File 22
HTML Browser Used by MHC 21

I

Insert the New FreeMarker Templates into the MPLAB Harmony Top-level Templates 19
Inserting New Library Help into the MPLAB Harmony Documentation Index 21
Installing a New Library into MPLAB Harmony 21
Introduction 3
    MHC Developer's Guide 3

K

Kconfig Language Specification 24

M

MHC Configuration File 31
MHC Files 30
MPLAB Harmony Configurator Developer's Guide 2
MPLAB Harmony Configurator Plug-ins 34

P

Pin Manager Development 36

S

Step 1: Create the File and Insert it into the hconfig Hierarchy 4
Step 2: Create a Menu Item for the Module in the Driver Framework Tree 6
Step 3: Creating Configuration Options 6
Step 4: Use Dependencies 7
Step 5: Use the Choice and Select Statements to Enable One Module Needed by Another 8
Step 6: Sourcing hconfig Files 10
Step 7: Adding Source Files to the MPLAB X IDE Project With the "file" Statement 11
Step 8: Add Help Links to Configuration Options 12
Step 9: Create Multiple Module Instances 12

U

Updating the BSP hconfig File 33
Using the Set Statement 15