Sample Library Help

MPLAB Harmony Integrated Software Framework

Volume V: MPLAB Harmony Framework Reference

This volume provides API reference information for the framework libraries included in your installation of MPLAB Harmony.

Description

This volume is a programmer reference that details the interfaces to the libraries that comprise MPLAB Harmony and explains how to use the libraries individually to accomplish the tasks for which they were designed.

Sample Library Help

This section describes the Sample Library that is available in MPLAB Harmony.

Introduction

This library provides a simple example of a MPLAB Harmony module.

Description

The Sample Library provides a simple example of a MPLAB Harmony module; however, its actual functionality is very limited and is not suitable for a practical application. The Sample Library provides an interface function to accept a single integer data value, another to see if the module currently holds any data, and a third to obtain any data currently held. The library can only hold up to two integer values at any given time.

Using the Library

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

Description

The interface to the Sample Library is defined in the `sample_module.h` header file.

Any C language source (`.c`) file that uses the Sample Library must include `sample_module.h`.

Library Overview

This section provides an overview of the Sample Library.

Description

The library interface routines are divided into various sub-sections, which address one of the blocks or the overall operation of the Sample Library.

Library Interface Section	Description
System Interface Functions	Functions used by the MPLAB Harmony system configuration to initialize and maintain the sample library module.
Client Interface Functions	Functions used by client modules and applications to interact with the Sample module.
Data Types and Constants	This section describes the various data types utilized in the Sample module interface.

How the Library Works

This section provides information on how the Sample Library works.

Core Functionality

Provides information on the core functionality of the Sample Library.

Description

The primary purpose of the Sample Library is to serve as an example of a dynamic MPLAB Harmony module (similar to a middleware library module). The library implements the system interface functions normally supported by a MPLAB Harmony module and provides a simple client-level interface to store and retrieve an integer data value. The library can hold up to two integer values per instance of the library. And, while it has no associated hardware or interrupts, the library can be used in an interrupt-driven configuration utilizing a timer (or other appropriate peripheral) providing it is appropriately configured and initialized.

Configuring the Library

Macros

	Name	Description
	SAMPLE_MODULE_INTERRUPT_MODE	Determines whether or not the sample module will support interrupt-driven execution mode.
	SAMPLE_MODULE_INSTANCES_NUMBER	Defines how many instances of the sample module are supported.
	SAMPLE_MODULE_TIMEOUT	Defines OSAL mutex timeout for Sample module (in milliseconds).

Description

The configuration of the Sample Library is based on the file `system_config.h`.

This header file contains the configuration selections for the Sample Library. Based on the selections made, the Sample Library may support the selected features.

This header can be placed anywhere, 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.

SAMPLE_MODULE_INTERRUPT_MODE Macro

Determines whether or not the sample module will support interrupt-driven execution mode.

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_INTERRUPT_MODE false
```

Description

Sample Module, Interrupt Mode

This macro determines whether or not the sample module will support interrupt interrupt driven execution. If defined as true, then the sample module will enable, disable, and clear the interrupt source identified in the SAMPLE_MODULE_INIT_DATA structure. Otherwise, it shoud be defined as false and the module will only support polled operation.

SAMPLE_MODULE_INSTANCES_NUMBER Macro

Defines how many instances of the sample module are supported.

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_INSTANCES_NUMBER 2
```

Description

Sample Module, Number of Instances

This macro defines how many instances of the sample module are supported.

Remarks

Defaults to 1 and displays a build warning if not specified..

SAMPLE_MODULE_TIMEOUT Macro

Defines OSAL mutex timeout for Sample module (in milliseconds).

File

sample_module_config_template.h

C

```
#define SAMPLE_MODULE_TIMEOUT 100
```

Description

Sample Module, Timeout

This macro defines how many milliseconds the sample module will block, waiting to obtain a mutex, when accessing its internal shared resources. This is only utilized in interrupt-driven modes, but needs to be defined to build correclty.

Remarks

Used by OSAL_MUTEX_Lock.

Building the Library

This section lists the files that are available in the Sample Library.

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>/framework/sample`.

Interface File(s)

This table lists and describes the header files that must be included (i.e., using `#include`) by any code that uses this library.

Source File Name	Description
`/sample_module.h`	This file provides the interface definitions of the Sample Library.

Required File(s)

MHC ***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
`/src/sample_module.c`	Sample Library implementation.

Optional File(s)

There are no optional files for the Sample Library.

Module Dependencies

The Sample Library is dependent upon the following libraries and header files.

- Interrupt System Service Library
- Operating System Abstraction Layer (OSAL)
- sys_common.h
- sys_module.h

Library Interface

a) System Interface Functions

	Name	Description
	SAMPLE_Initialize	Initializes the sample module.
	SAMPLE_Reinitialize	Reinitializes the sample module and refreshes any associated internal settings.
	SAMPLE_Deinitialize	Deinitializes the specified instance of the sample module.
	SAMPLE_Tasks	Maintains the sample module instance's state machine.
	SAMPLE_Status	Provides the current status of the sample module instance.

b) Client Interface Functions

	Name	Description
	SAMPLE_DataGive	Gives data to the sample module.
	SAMPLE_DataGet	Gets data from the sample module.
	SAMPLE_DataStatus	Identifies the current status of the data processed by the module.

c) Data Types and Constants

	Name	Description
	SAMPLE_INDEX_0	Labels for module index numbers.
	SAMPLE_INDEX_1	This is macro SAMPLE_INDEX_1.
	SAMPLE_MODULE_DATA_STATUS	Possible data processing states.
	SAMPLE_MODULE_INIT_DATA	Data necessary to initialize the sample module.

	_sample_module_data_status	Possible data processing states.
	_sample_module_init_data	Data necessary to initialize the sample module.

Description

This section describes the APIs of the Sample Library. Refer to each section for a description.

a) System Interface Functions

SAMPLE_Initialize Function

Initializes the sample module.

File

sample_module.h

C

```
SYS_MODULE_OBJ SAMPLE_Initialize(const SYS_MODULE_INDEX index, const SYS_MODULE_INIT * const init);
```

Returns

If successful, returns a valid handle to a system module object. Otherwise, it returns SYS_MODULE_OBJ_INVALID. The returned object handle must be passed as an argument to SAMPLE_Reinitialize, SAMPLE_Deinitialize, SAMPLE_Tasks and SAMPLE_Status routines.

Description

This routine initializes the sample module, making it ready for clients to open and use it.

Remarks

This routine must be called before any other sample module routine is called.

This routine should only be called once during system initialization unless SAMPLE_Deinitialize is called to deinitialize the module instance.

This routine will NEVER block for internal state changes. If the operation requires time to complete, it will be reported by the SAMPLE_Status operation. The system can use SAMPLE_Status to find out when the associated instance of the sample module is in a ready state.

Preconditions

None.

Example

```
SAMPLE_MODULE_INIT_DATA init;
SYS_MODULE_OBJ          obj;

// Populate the sample initialization structure
init.sys.powerState     = SYS_MODULE_POWER_RUN_FULL;
init.dataSome           = 42;

// Initialize the first instance of the sample module.
obj = SAMPLE_Initialize(SAMPLE_INDEX_0, (SYS_MODULE_INIT *)&init);
if (SYS_MODULE_OBJ_INVALID == obj)
{
    // Handle failure
}
```

Parameters

Parameters	Description
index	Index for the module instance to be initialized
init	Pointer to a data structure containing any data necessary to initialize the module.

Function

SYS_MODULE_OBJ SAMPLE_Initialize(const SYS_MODULE_INDEX index,

const SYS_MODULE_INIT * const init)

SAMPLE_Reinitialize Function

Reinitializes the sample module and refreshes any associated internal settings.

File

sample_module.h

C

```
void SAMPLE_Reinitialize(SYS_MODULE_OBJ object, const SYS_MODULE_INIT * const init);
```

Returns

None

Description

This routine reinitializes the sample module and refreshes any associated settings using the initialization data given. However, it will not interrupt ongoing client operations.

Remarks

This function can be called multiple times to reinitialize the module.

This operation can be used to refresh any supported internal settings as specified by the initialization data or to change the power state of the module.

This routine will NEVER block for internal state changes. If the operation requires time to compete, it will be reported by the SAMPLE_Status operation. The system can use SAMPLE_Status to find out when the specified sample module instance is in a ready state.

Preconditions

Function SAMPLE_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ must have been returned.

Example

```
SYS_MODULE_OBJ          obj;    // Returned from SAMPLE_Initialize
SAMPLE_MODULE_INIT_DATA init;
SYS_STATUS              status;

// Populate the sample initialization structure
init.sys.powerState     = SYS_MODULE_POWER_SLEEP;
init.dataSome           = 0;

SAMPLE_Reinitialize(obj, (SYS_MODULE_INIT*)&init);

status = SAMPLE_Status(obj);
if (SYS_STATUS_BUSY == status)
{
    // Check again later to ensure the driver is ready
}
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Sample module object handle, returned from the SAMPLE_Initialize routine
init	Pointer to the initialization data structure

Function

void SAMPLE_Reinitialize(SYS_MODULE_OBJ object,

const SYS_MODULE_INIT * const init)

SAMPLE_Deinitialize Function

Deinitializes the specified instance of the sample module.

File

sample_module.h

C

```
void SAMPLE_Deinitialize(SYS_MODULE_OBJ object);
```

Returns

None.

Description

Deinitializes the specified instance of the sample module, disabling its operation. Invalidates all the internal data and client operations.

Remarks

Once the Initialize operation has been called, the Deinitialize operation must be called before the Initialize operation can be called again.

This routine will NEVER block waiting for internal state changes. If the operation requires time to complete, it will be reported by the SAMPLE_Status operation. The system can to use SAMPLE_Status to find out when the module is in the SYS_STATUS_UNINITIALIZED state.

Preconditions

Function SAMPLE_Initialize must have been called before calling this routine and a valid SYS_MODULE_OBJ handle must have been returned.

Example

```
SYS_MODULE_OBJ      object;     //  Returned from SAMPLE_Initialize
SYS_STATUS          status;

SAMPLE_Deinitialize(object);

status = SAMPLE_Status(object);
if (SYS_MODULE_DEINITIALIZED == status)
{
    // Check again later if you need to know when the sample module
    // instance has been deinitialized.
}
```

Parameters

Parameters	Description
object	Sample module instance's object handle, returned from the SAMPLE_Initialize routine.

Function

void SAMPLE_Deinitialize (SYS_MODULE_OBJ object)

SAMPLE_Tasks Function

Maintains the sample module instance's state machine.

File

sample_module.h

C

```
void SAMPLE_Tasks(SYS_MODULE_OBJ object);
```

Returns

None.

Description

This routine is used to maintain the sample module instance's internal state machine.

Remarks

This routine should not be called directly a client of the sample module. It is called by the system's Tasks routine (SYS_Tasks) or by an appropriate ISR.

This routine may execute in an ISR context and will never block or access any resources that may cause it to block.

Preconditions

The SAMPLE_Initialize routine must have been called for the specified SAMPLE module instance.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SAMPLE_Initialize

while (true)
{
    SAMPLE_Tasks (object);

    // Do other tasks
}
```

Parameters

Parameters	Description
object	Object handle for the specified sample module instance (returned from SAMPLE_Initialize).

Function

void SAMPLE_Tasks(SYS_MODULE_OBJ object)

SAMPLE_Status Function

Provides the current status of the sample module instance.

File

sample_module.h

C

```
SYS_STATUS SAMPLE_Status(SYS_MODULE_OBJ object);
```

Returns

- SYS_STATUS_ERROR - Indicates that the module instance is in an error state. Note: Any value less than SYS_STATUS_ERROR is also an error state.
- SYS_STATUS_UNINITIALIZED - Indicates that the driver has been deinitialized.

Description

This routine provides the current status of the sample module instance.

Remarks

This value is greater than SYS_STATUS_ERROR

- SYS_STATUS_BUSY - Indicates that the module instance is busy with a previous system level operation and cannot start another
- SYS_STATUS_READY - Indicates that the module instance is running and ready to begin another system level operation.

Any value greater than SYS_STATUS_READY is also a normal running state in which the module is ready to accept new operations

The this operation can be used to determine when any of the system operations have completed.

If the status operation returns SYS_STATUS_BUSY, the a previous operation has not yet completed. Once the status operation returns SYS_STATUS_READY, any previous operations have completed.

The value of SYS_STATUS_ERROR is negative (-1). Any value less than that is also an error state.

This routine will NEVER block waiting for an internal state change.

If the Status operation returns an error value, the error may be cleared by calling the reinitialize operation. If that fails, the deinitialize operation will need to be called, followed by the initialize operation to return to normal operations.

Preconditions

Function SAMPLE_Initialize must have been called before calling this function.

Example

```
SYS_MODULE_OBJ      object;     // Returned from SAMPLE_Initialize
SYS_STATUS          status;
```

```
status = SAMPLE_Status(object);
else if (SYS_STATUS_ERROR >= status)
{
    // Handle error
}
```

Parameters

Parameters	Description
object	Sample module instance's object handle, returned from the SAMPLE_Initialize routine.

Function

SYS_STATUS SAMPLE_Status (SYS_MODULE_OBJ object)

b) Client Interface Functions

SAMPLE_DataGive Function

Gives data to the sample module.

File

sample_module.h

C

```
bool SAMPLE_DataGive(const SYS_MODULE_INDEX index, int data);
```

Returns

- true - If successful.
- false - If an error occurred.

Description

This function gives data to the sample module.

Remarks

Any data value given to the sample module will be available via the SAMPLE_DataGet function later, after the sample module state machine is ready to give it back.

Preconditions

The sample module should be initialized before calling this API function.

Example

```
if (SAMPLE_DataGive(SAMPLE_INDEX_0, 42))
{
    // Data given
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.
data	Data value to give to the sample module.

Function

bool SAMPLE_DataGive (const SYS_MODULE_INDEX index, int data)

SAMPLE_DataGet Function

Gets data from the sample module.

File

sample_module.h

C

```
bool SAMPLE_DataGet(const SYS_MODULE_INDEX index, int * data);
```

Returns

- true - If successful.
- false - If an error occurred. If an error occurs, the variable pointed to by pData is not updated.

Description

This function gets data from the sample module. If no data has been given to the sample module yet, it will return a failure

Remarks

Any data value given to the sample module by the SAMPLE_DataGive function will be available via this function later, after the sample module state machine is ready to give it back.

Preconditions

The sample module should be initialized before calling this API function.

Example

```
int data;

if (SAMPLE_DataGet(SAMPLE_INDEX_0, &data))
{
    // Data valid
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.
pData	Pointer to the variable to receive the data value.

Function

bool SAMPLE_DataGet (const SYS_MODULE_INDEX index, int *pData)

SAMPLE_DataStatus Function

Identifies the current status of the data processed by the module.

File

sample_module.h

C

```
SAMPLE_MODULE_DATA_STATUS SAMPLE_DataStatus(const SYS_MODULE_INDEX index);
```

Returns

- SAMPLE_MODULE_DATA_NONE - If the sample module has no data to give.
- SAMPLE_MODULE_DATA_BUSY - If the sample module is busy processing and will have data available soon.
- SAMPLE_MODULE_DATA_AVAILABLE - If the sample module has data available.

Description

This function identifies the current status of the data processed by the module.

Remarks

In a polled "super-loop" system, you cannot poll this function waiting for the sample module to change status from SAMPLE_MODULE_DATA_BUSY to SAMPLE_MODULE_DATA_AVAILABLE. The loop must be allowed to continue so that the sample module can finish processing the data before it will become available.

Preconditions

The sample module must be initialized before calling this API function.

Example

```
if (SAMPLE_MODULE_DATA_BUSY == SAMPLE_DataStatus(SAMPLE_INDEX_0))
{
```

```
    // Check again later.
}
```

Parameters

Parameters	Description
index	Index of the desired module instance.

Function

SAMPLE_MODULE_DATA_STATUS SAMPLE_DataStatus (const SYS_MODULE_INDEX index)

c) Data Types and Constants

SAMPLE_INDEX_0 Macro

Labels for module index numbers.

File

sample_module.h

C

```
#define SAMPLE_INDEX_0 0
```

Description

Module Instance Indexes

The following symbolic labels can be used in place of numeric literals to identify module indexes.

SAMPLE_INDEX_1 Macro

File

sample_module.h

C

```
#define SAMPLE_INDEX_1 1
```

Description

This is macro SAMPLE_INDEX_1.

SAMPLE_MODULE_DATA_STATUS Enumeration

Possible data processing states.

File

sample_module.h

C

```
typedef enum _sample_module_data_status {
  SAMPLE_MODULE_DATA_NONE = 0,
  SAMPLE_MODULE_DATA_BUSY,
  SAMPLE_MODULE_DATA_AVAILABLE
} SAMPLE_MODULE_DATA_STATUS;
```

Members

Members	Description
SAMPLE_MODULE_DATA_NONE = 0	Indicates that the sample module has no data available.
SAMPLE_MODULE_DATA_BUSY	Indicates that the sample module is busy processing and will have data available soon.
SAMPLE_MODULE_DATA_AVAILABLE	Indicates that the sample module has available data.

Description

Sample Module Data Status

This enumeration defines the possible data processing states of the sample module.

Remarks

This status is provided by the SAMPLE_DataStatus function..

SAMPLE_MODULE_INIT_DATA Structure

Data necessary to initialize the sample module.

File

sample_module.h

C

```
typedef struct _sample_module_init_data {
  SYS_MODULE_INIT moduleInit;
  int dataSome;
  TMR_MODULE_ID tmr;
  TMR_PRESCALE prescale;
  uint16_t period;
  INT_SOURCE interrupt;
  INT_VECTOR vector;
  INT_PRIORITY_LEVEL priority;
  INT_SUBPRIORITY_LEVEL subpriority;
} SAMPLE_MODULE_INIT_DATA;
```

Members

Members	Description
SYS_MODULE_INIT moduleInit;	Standard system module init data.
int dataSome;	Some initialization data.
TMR_MODULE_ID tmr;	Timer used in interrupt-driven configurations.
TMR_PRESCALE prescale;	Timer prescaler setting.
uint16_t period;	Timer period setting.
INT_SOURCE interrupt;	Timer interrupt source.
INT_VECTOR vector;	Timer interrupt vector.
INT_PRIORITY_LEVEL priority;	Timer interrupt priority.
INT_SUBPRIORITY_LEVEL subpriority;	Timer interrupt subpriority.

Description

Sample Module Initialization Data

This data structure contains data necessary to initialize an instance of the sample module.

Remarks

A pointer to a persistent copy of this structure must be passed into the SAMPLE_Initialize function.

Files

Files

Name	Description
sample_module.h	Sample of a MPLAB Harmony library module.
sample_module_config_template.h	Sample module configuration templates.

Description

This section lists the source and header files used by the SAMPLE Library.

sample_module.h

Sample of a MPLAB Harmony library module.

Enumerations

	Name	Description
	_sample_module_data_status	Possible data processing states.

	SAMPLE_MODULE_DATA_STATUS	Possible data processing states.

Functions

	Name	Description
	SAMPLE_DataGet	Gets data from the sample module.
	SAMPLE_DataGive	Gives data to the sample module.
	SAMPLE_DataStatus	Identifies the current status of the data processed by the module.
	SAMPLE_Deinitialize	Deinitializes the specified instance of the sample module.
	SAMPLE_Initialize	Initializes the sample module.
	SAMPLE_Reinitialize	Reinitializes the sample module and refreshes any associated internal settings.
	SAMPLE_Status	Provides the current status of the sample module instance.
	SAMPLE_Tasks	Maintains the sample module instance's state machine.

Macros

	Name	Description
	SAMPLE_INDEX_0	Labels for module index numbers.
	SAMPLE_INDEX_1	This is macro SAMPLE_INDEX_1.

Structures

	Name	Description
	_sample_module_init_data	Data necessary to initialize the sample module.
	SAMPLE_MODULE_INIT_DATA	Data necessary to initialize the sample module.

Description

MPLAB Harmony Sample Library Module Interface Header

This is a sample of a MPLAB Harmony library interface module. This file defines the system and application interface.

File Name

sample_module.h

Company

Microchip Technology Inc.

sample_module_config_template.h

Sample module configuration templates.

Macros

	Name	Description
	SAMPLE_MODULE_INSTANCES_NUMBER	Defines how many instances of the sample module are supported.
	SAMPLE_MODULE_INTERRUPT_MODE	Determines whether or not the sample module will support interrupt-driven execution mode.
	SAMPLE_MODULE_TIMEOUT	Defines OSAL mutex timeout for Sample module (in milliseconds).

Description

Sample Module Configuration Templates

This file contains examples of constants to configure the sample module.

File Name

sample_module_config_template.h

Company

Microchip Technology Inc.

Index