# Embedded USB Host MIDI Class Driver User Guide

Version 1.10

For use with USBH CD - MIDI versions 2.04 and above

**Date:** 19-Jun-2017 12:50

# Table of Contents

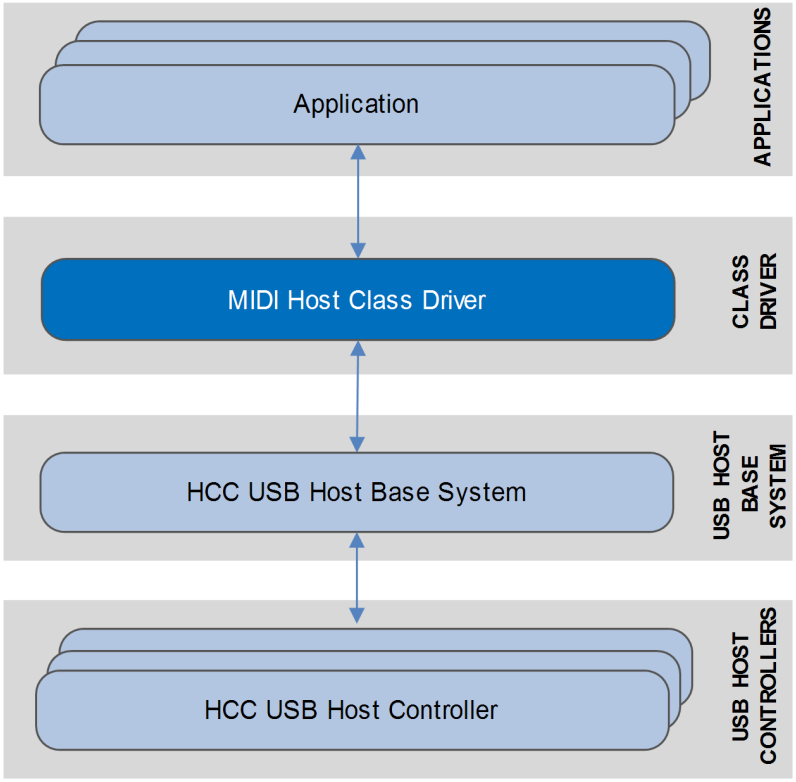
# 1 System Overview

## 1.1 Introduction

This guide is for those who want to implement an embedded MIDI (Musical Instrument Digital Interface) host class driver to control musical devices connected to a host through a USB interface. The **usbh_cd_midi** package provides a MIDI host class driver for a USB stack.

This driver allows you to plug a MIDI-compatible USB device into the system and stream music to it. It processes MIDI events received from the device using the event handler.

The system structure is shown in the diagram below:




The lower layer interface is designed to use HCC Embedded's USB Host Interface Layer. This layer is standard over different host controller implementations; this means that the code is unchanged, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, consult the *HCC USB Host Base System User Guide* that is shipped with the base system.

The package provides a set of Application Programming Interface (API) functions for controlling access to a device. These are described here, with separate sections for module and device management.

## 1.2 Feature Check

The main features of the class driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Compatible with all HCC USB host controllers.
- Supports multiple devices connected simultaneously.
- Supports all devices which comply with the MIDI specification.
- Handles a defined set of MIDI-specified events by using the event handler.
- Uses a system of callbacks for user-specified events.

## 1.3 Packages and Documents

### Packages

The table below lists the packages that you need in order to use this module:

| Package | Description |
|---|---|
| **hcc_base_doc** | This contains the two guides that will help you get started. |
| **usbh_base** | The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package. |
| **usbh_cd_midi** | The USB device MIDI host class driver package described by this document. |

### Documents

For an overview of HCC's embedded USB stacks, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

**HCC Firmware Quick Start Guide**

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

**HCC Source Tree Guide**

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

**HCC USB Host Base System User Guide**

This document defines the USB host base system upon which the complete USB stack is built.

**HCC Embedded USB Host MIDI Class Driver User Guide**

This is this document.

## 1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: Embedded USB Host MIDI Class Driver User Guide.
- For the history of changes made to the package code itself, see History: usbh_cd_midi.

The current version of this manual is 1.10. The full list of versions is as follows:

| Manual version | Date | Software version | Reason for change |
|---|---|---|---|
| 1.10 | 2017-06-19 | 2.04 | New *Change History* format. |
| 1.00 | 2017-04-21 | 2.04 | First release. |

# 2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

## 2.1 API Header File

The file **src/api/api_usbh_midi.h** should be included by any application using the system. This is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

## 2.2 Configuration File

The file **src/config/config_usbh_midi.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

## 2.3 Source Code

The file **src/usb-host/class-drivers/midi/usbh_midi.c** is the main code for the module. **This file should only be modified by HCC**.

## 2.4 Version File

The file **src/version/ver_usbh_midi.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

# 3 Configuration Options

Set the system configuration options in the file **src/config/config_usbh_midi.h**.

**USBH_MIDI_MAX_UNITS**

The number of supported MIDI devices. The default is 1.

**USBH_MIDI_EVENT_BUFFER_SIZE**

The maximum number of received events in the queue. The default is 32.

If the queue is full, new events overwrite old ones.

# 4 Application Programming Interface

This section documents the Application Programming Interface (API). It includes all the functions that are available to an application program.

## 4.1 Module Management Functions

The functions are the following:

| Function | Description |
|---|---|
| **usbh_midi_init()** | Initializes the module and allocates the required resources. |
| **usbh_midi_start()** | Starts the module. |
| **usbh_midi_stop()** | Stops the module. |
| **usbh_midi_delete()** | Deletes the module and releases the resources it used. |

## usbh_midi_init

Use this function to initialize the class driver and allocate the required resources.

> **Note:** You must call this before any other function.

### Format

```
int usbh_midi_init ( void )
```

### Arguments

| Parameter |
|---|
| None. |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_start

Use this function to start the class driver.

> **Note:** You must call **usbh_midi_init()** before this function.

### Format

```
int usbh_midi_start ( void )
```

### Arguments

| Parameter |
|---|
| None. |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_stop

Use this function to stop the class driver.

### Format

```
int usbh_midi_stop ( void )
```

### Arguments

| Parameter |
|---|
| None. |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_delete

Use this function to delete the class driver and release the associated resources.

### Format

```
int usbh_midi_delete ( void )
```

### Arguments

| Parameter |
|---|
| None. |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## 4.2 Device Management Functions

The functions are the following:

| Function | Description |
|---|---|
| **usbh_midi_add_event()** | Adds an event to the output queue. |
| **usbh_midi_get_event()** | Gets the next event from the receive queue. |
| **usbh_midi_get_info()** | Gets information from the MIDI device. |
| **usbh_midi_get_port_hdl()** | Gets the device's port handle. |
| **usbh_midi_present()** | Checks whether a MIDI device is connected. |
| **usbh_midi_send()** | Sends all the events in the queue. |
| **usbh_midi_register_ntf()** | Registers a notification function for a specified event type. |

## usbh_midi_add_event

Use this function to add an event to the output queue.

This function does not send the event; use **usbh_midi_send()** to do that.

### Format

```
int usbh_midi_add_event (
    t_usbh_unit_id        uid,
    t_usbh_midi_event *   event )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |
| event | A pointer to the event structure. | t_usbh_midi_event * |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_get_event

Use this function to get the next event from the receive queue.

### Format

```
int usbh_midi_get_event (
    t_usbh_unit_id        uid,
    t_usbh_midi_event *   event )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |
| event | Where to write the event structure. | t_usbh_midi_event * |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_get_info

Use this function to get information from the MIDI device.

### Format

```
int usbh_midi_get_info (
    t_usbh_unit_id   uid,
    uint8_t *        n_in,
    uint8_t *        n_out )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |
| n_in | The number of IN cables. | uint8_t * |
| n_out | The number of OUT cables | uint8_t * |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_get_port_hdl

Use this function to get the device's port handle.

### Format

```
t_usbh_port_hdl usbh_midi_get_port_hdl ( t_usbh_unit_id uid )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |

### Return Values

| Return value | Description |
|---|---|
| The port handle. | Successful execution. |
| USBH_PORT_HDL_INVALID | Invalid port handle. |
| Else | See Error Codes. |

## usbh_midi_present

Use this function to check whether a MIDI device is connected.

### Format

```
int usbh_midi_present ( t_usbh_unit_id uid )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |

### Return Values

| Return value | Description |
|---|---|
| 0 | No MIDI device is present. |
| 1 | A MIDI device is present. |
| Else | See Error Codes. |

## usbh_midi_send

Use this function to send all the events in the queue.

Use **usbh_midi_add_event()** to add events to the queue.

### Format

```
int usbh_midi_send ( t_usbh_unit_id uid )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## usbh_midi_register_ntf

Use this function to register a notification function for a specified event type.

When a device is connected or disconnected, the notification function is called.

> **Note:** It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

### Format

```
int usbh_midi_register_ntf (
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf,
    t_usbh_ntf_fn    ntf_fn )
```

### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |
| ntf | The notification ID. | t_usbh_ntf |
| ntf_fn | The notification function to use when an event occurs. | t_usbh_ntf_fn |

### Return Values

| Return value | Description |
|---|---|
| USBH_MIDI_SUCCESS | Successful execution. |
| Else | See Error Codes. |

## 4.3 Error Codes

### MIDI-specific Error Codes

If a function executes successfully, it returns with a USBH_MIDI_SUCCESS code, a value of 0. The following table shows the meaning of the error codes:

| Return Value | Value | Description |
|---|---|---|
| USBH_MIDI_SUCCESS | 0 | Successful execution. |
| USBH_MIDI_NO_EVENT | 1 | No event available. |
| USBH_MIDI_FULL | 2 | Event buffer full; it cannot accept more events. |
| USBH_MIDI_ERR_NOT_PRESENT | 3 | No MIDI device is connected. |
| USBH_MIDI_ERR_RESOURCE | 4 | Resource error. |
| USBH_MIDI_ERROR | 5 | General error. |

## USBH Error Codes

If a function executes successfully it returns with a USBH_SUCCESS code, a value of 0. The following table shows the meaning of the error codes:

| Return Code | Value | Description |
|---|---|---|
| USBH_SUCCESS | 0 | Successful execution. |
| USBH_SHORT_PACKET | 1 | IN transfer completed with short packet. |
| USBH_PENDING | 2 | Transfer still pending. |
| USBH_ERR_BUSY | 3 | Another transfer in progress. |
| USBH_ERR_DIR | 4 | Transfer direction error. |
| USBH_ERR_TIMEOUT | 5 | Transfer timed out. |
| USBH_ERR_TRANSFER | 6 | Transfer failed to complete. |
| USBH_ERR_TRANSFER_FULL | 7 | Cannot process more transfers. |
| USBH_ERR_SUSPENDED | 8 | Host controller is suspended. |
| USBH_ERR_HC_HALTED | 9 | Host controller is halted. |
| USBH_ERR_REMOVED | 10 | Transfer finished due to device removal. |
| USBH_ERR_PERIODIC_LIST | 11 | Periodic list error. |
| USBH_ERR_RESET_REQUEST | 12 | Reset request during enumeration. |
| USBH_ERR_RESOURCE | 13 | OS resource error. |
| USBH_ERR_INVALID | 14 | Invalid identifier/type (HC, EP HDL, and so on). |
| USBH_ERR_NOT_AVAILABLE | 15 | Item not available. |
| USBH_ERR_INVALID_SIZE | 16 | Invalid size. |
| USBH_ERR_NOT_ALLOWED | 17 | Operation not allowed. |
| USBH_ERROR | 18 | General error. |

## 4.4 Types and Definitions

### t_usbh_ntf_fn

The **t_usbh_ntf_fn** definition specifies the format of the notification function. It is defined in the USB host base system in the file **api_usb_host.h**.

#### Format

```
int  ( * t_usbh_ntf_fn )(
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf )
```

#### Arguments

| Parameter | Description | Type |
|---|---|---|
| uid | The unit ID. | t_usbh_unit_id |
| ntf | The notification code. | t_usbh_ntf |

### Notification Codes

The standard notification codes shown below are defined in the USB host base system in the file **api_usb_host.h**. This module has no specific notification codes of its own.

| Notification | Value | Description |
|---|---|---|
| USBH_NTF_CONNECT | 1 | Connection notification code. |
| USBH_NTF_DISCONNECT | 2 | Disconnection notification code. |

## t_usbh_midi_event

The *t_usbh_midi_event* structure takes this form:

| Element | Type | Description |
|---|---|---|
| cn | uint8_t | Cable number. |
| cin | uint8_t | Code Index Number (CIN). |
| event[3] | uint8_t | MIDI event. |

## CIN Codes

This table shows the Code Index Number (CIN) codes:

| CIN | Value |
|---|---|
| MIDI_CIN_MISC | 0 |
| MIDI_CIN_RESERVED | 1 |
| MIDI_CIN_COMMON_2 | 2 |
| MIDI_CIN_COMMON_3 | 3 |
| MIDI_CIN_SYSEX_ST_CNT | 4 |
| MIDI_CIN_SYSEX_END_1 | 5 |
| MIDI_CIN_SYSEX_END_2 | 6 |
| MIDI_CIN_SYSEX_END_3 | 7 |
| MIDI_CIN_NOTE_ON | 8 |
| MIDI_CIN_NOTE_OFF | 9 |
| MIDI_CIN_POLY_KEYPRESS | 10 |
| MIDI_CIN_CONTROL_CHANGE | 11 |
| MIDI_CIN_PROGRAM_CHANGE | 12 |
| MIDI_CIN_CHANNEL_PRESSURE | 13 |
| MIDI_CIN_PITCHBEND_CHANGE | 14 |
| MIDI_CIN_SINGLE_BYTE | 15 |

# 5 Integration

This section specifies the elements of this package that need porting, depending on the target environment.

## 5.1 OS Abstraction Layer (OAL)

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The MIDI class driver uses the following OAL components:

| OAL Resource | Number Required |
|---|---|
| Tasks | 0 |
| Mutexes | 1 |
| Events | 0 |

## 5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer.

The MIDI class driver makes use of the following standard PSP functions:

| Function | Package | Component | Description |
|---|---|---|---|
| **psp_memcpy()** | psp_base | psp_string | Copies a block of memory. The result is a binary copy of the data. |
| **psp_memset()** | psp_base | psp_string | Sets the specified area of memory to the defined value. |

# 6 Sample Code

This section shows example code for the class driver.

## 6.1 Initialization

This example shows the code used to initialize a USB host with the class driver.

```
/*
** Initialize USB host with MIDI class driver.
*/

int usb_host_init ( void )
{
    int  rc;
    rc = hcc_mem_init();

    /* Initialize the USB host module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_init();
    }

    /* Initialize the specific USB host controller */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_stm32uh_hc, 0 );
    }

    /* Initialize the MIDI Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_midi_init();
    }

    /* Start the MIDI Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_midi_start();
    }

    /* Start the USB host stack */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_start();  /* Start the USB host */
    }

    return rc;

} /* usb_host_init */
```