USB Device HID Class Driver User Guide

Version 3.30

For use with USBD HID Class Driver versions 8.01 and above

Date: 16-Jun-2017 13:59

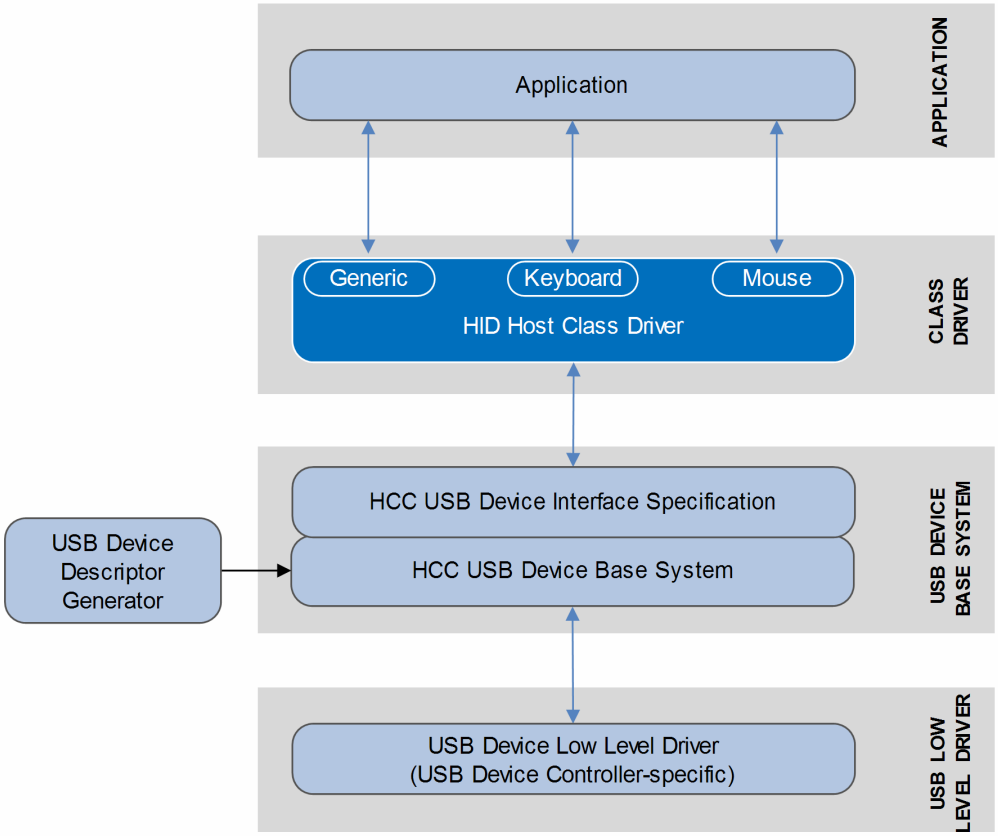
Table of Contents

1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB HID class driver to control Human Interface Device (HID) devices. HID devices include keyboards, mice and "generic devices" (pointers, buttons, sliders, and so on). HID can also be used with devices which provide data in a similar way, such as point-of-sale equipment.

The system structure is shown in the diagram below:



Note:

- This module is part of HCC's Embedded USB Device (EUSBD) system, as described in the *HCC Embedded USB Device Base System User Guide*. This module communicates with the EUSBD base system through the EUSBD Device Interface, as described in the above manual.
- Other types of HCC class driver can be added to the system, for example CDC-ACM, Mass Storage, and Audio. For the current list of supported class drivers, contact sales@hcc-embedded.com.

The package provides a set of Application Programming Interface (API) functions. The API description in this manual has separate sections describing the generic device, keyboard, and mouse functions. The management interface is the set of functions used to initialize, start, stop, and delete the module.

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.
- Supports all devices that conform to the HID specification: generic devices, keyboards and mice.
- Compatible with sample device files produced by using the HCC *USB Device Descriptor Generator*.
- Uses a system of user-defined notification functions for state change events.

1.3 Packages and Documents

Packages

This table 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.
usbd_base	The USB device base package. This is the framework used by USB class drivers to communicate over USB using a specific USB device controller package.
usbd_cd_hid	The HID class driver for keyboard, mouse, and generic devices.

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 Embedded USB Device Base System User Guide

This document describes the Embedded USB Device base system.

HCC USB Device HID Class Driver User Guide

This is this document.

HCC USB Device Descriptor Generator User Guide

This document describes the tool that creates USB descriptor files for inclusion in a project that uses the EUSBD stack.

1.4 Change History

This section describes past changes to this manual.

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

The current version of this manual is 3.30. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
3.30	2017-06-16	8.01	New *Change History* format.
3.20	2016-04-21	8.01	Added function group descriptions to API.
3.10	2015-11-27	8.01	Changes to *Configuration Options*.
3.00	2015-04-24	7.02	First online 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 files.

2.1 API Header Files

These files in the directory **src/api** should be included by any application using the system. These are the only files that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

File	Description
api_usbd_hid.h	Base HID module API.
api_usbd_hid_generic.h	Generic device HID API.
api_usbd_hid_kbd.h	HID keyboard API.
api_usbd_hid_mouse.h	HID mouse API.

2.2 Configuration Files

These files in the directory **src/config** contain all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

File	Description
config_usbd_hid.h	Base HID module configuration file.
config_usbd_hid_generic.h	Generic device configuration file.
config_usbd_hid_kbd.h	Keyboard configuration file.
config_usbd_hid_mouse.h	Mouse configuration file.

2.3 Source Code Files

These files are in the directory **src/usb-device/class-drivers/hid**. **These files should only be modified by HCC**.

File	Description
usbd_hid.c	Base HID module source code.
usbd_hid.h	Base HID module header file.
usbd_hid_generic.c	Generic device source code.
usbd_hid_kbd.c	Keyboard source code.
usbd_hid_mouse.c	Mouse source code.

2.4 Version File

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

3 Configuration Options

All system configuration options are set in the four configuration files described in this section. The available configuration options and their default values are described.

3.1 System Configuration Options

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

USE_REPORT_ID

Set this to 1 if the HID report uses report IDs, 0 if it does not. The default is 0.

HID_CHECK_OUT_REPORT_SIZE

Keep the default of 1 to check the OUT report size and discard reports that do not match the OUT report size reported to the host. Set this to 0 disable checking and receive all reports.

USE_RX_IT_EP_0

Set this to 1 if you intend to use an Interrupt OUT endpoint to receive reports from the host. In this case the configuration descriptor must contain a second Interrupt endpoint for the HID interface. The default is 0.

> **Note**: The descriptors define the additional endpoint too.

MAX_NO_OF_REPORTS

The maximum number of reports supported. The default is 2.

MAX_REPORT_LENGTH

The maximum length of a report (excluding the report ID prefix.) The default is 64.

HID_TXTASK_PRIORITY

The priority of the HID transfer task. The default is OAL_LOW_PRIORITY and this generally does not need to be modified. For information on priorities, see the OAL documentation.

HID_TXTASK_STACK_SIZE

The size of the transfer task stack in bytes. The default is 256. For some RTOSes you may need to increase this value.

HID_RXTASK_PRIORITY

The priority of the HID receive task. The default is OAL_NORMAL_PRIORITY and this generally does not need to be modified. For information on priorities, see the OAL documentation.

HID_RXTASK_STACK_SIZE

The size of the receive task stack in bytes. The default is 256. For some RTOSes you may need to increase this value.

3.2 Generic Device Options

Set the generic device configuration options in the file **src/config/config_usbd_hid_generic.h**.

MAX_HID_GENERIC_IFC

The maximum number of generic device interfaces. The default is 1.

USBD_GHID_IN_MAX_REPORT_NUM

The number of IN report IDs. The default is 1.

USBD_GHID_OUT_MAX_REPORT_NUM

The number of OUT report IDs. The default is 0.

config_usbd_hid_generic.c

There is also a generic device configuration file named **config_usbd_hid_generic.c**. Modify this as required.

This file defines the following:

- IN report sizes.
- OUT report sizes.
- A report ID definition array for every interface (unit).

The entries in the report ID definition array are as follows:

- The interface index.
- The report ID base for IN reports.
- The number of IN reports (the maximum is 16). The report IDs are set in sequential order.
- The report ID base for OUT reports (the maximum is 16).
- The number of OUT reports. The report IDs are set in sequential order.

3.3 Keyboard Options

Set the keyboard configuration options in the file **src/config/config_usbd_hid_kbd.h**.

MAX_KEYS_PRESSED

The number of keys which can be pressed simultaneously. The value must match the size of the array in the HID report descriptor. The default value is 6.

MAX_HID_KBD_IFC

The maximum number of keyboard interfaces. The default value is 1.

3.4 Mouse Options

Set the single mouse configuration option in the file **src/config/config_usbd_hid_mouse.h**.

MAX_HID_MOUSE_IFC

The maximum number of mouse interfaces. The default is 1.

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

The functions are the following:

Function	Description
usbd_hid_init()	Initializes the module and allocates the required resources.
usbd_hid_start()	Starts the module.
usbd_hid_stop()	Stops the module.
usbd_hid_delete()	Deletes the module and releases the resources it used.

usbd_hid_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 usbd_hid_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_hid_start

Use this function to start the class driver.

> **Note:** You must call **usbd_hid_init()** before this to initialize the module.

Format

```
int usbd_hid_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_hid_stop

Use this function to stop the class driver.

Format

```
int usbd_hid_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_hid_delete

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

Format

```
int usbd_hid_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

4.2 Generic Device Management

The functions are the following:

Function	Description
usbd_ghid_init()	Initializes the generic device module and allocates the required resources.
usbd_ghid_start()	Starts the generic device module.
usbd_ghid_stop()	Stops the generic device module.
usbd_ghid_delete()	Deletes the generic device module and releases the resources it used.
usbd_ghid_read_data()	Reads the data from the OUT report.
usbd_ghid_write_data()	Writes data to the IN report.
usbd_ghid_register_ntf()	Registers a notification function to be called when an OUT report is received or an IN report is sent.

usbd_ghid_init

Use this function to initialize the generic device module and allocate the required resources.

Format

```
int usbd_ghid_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_ghid_start

Use this function to start the generic device module.

> **Note:** You must call **usbd_ghid_init()** before this to initialize the generic device module.

Format

```
int usbd_ghid_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_ghid_stop

Use this function to stop the generic device module.

Format

```
int usbd_ghid_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_ghid_delete

Use this function to remove the generic device module and release the associated resources.

Format

```
int usbd_ghid_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_ghid_read_data

Use this function to read the data from the OUT report.

You can register a notification function to find when data was obtained; see usbd_ghid_register_ntf() for details.

Format

```
int usbd_ghid_read_data (
    uint8_t      unit,
    uint8_t      id,
    uint8_t *    p_buf,
    uint16_t *   length )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
id	The report ID.	uint8_t
p_buf	A pointer to the buffer to read data from.	uint8_t *
length	A pointer to the length of the received OUT report.	uint16_t *

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
USBD_HID_ERROR_NOT_READY	Device is not operational.
USBD_HID_ERROR_BUSY	Transfer in progress.
Else	See Error Codes.

usbd_ghid_write_data

Use this function to write data to the IN report.

This function reports a busy state if data written by using it earlier has not yet been sent.

Format

```
int usbd_ghid_write_data (
    uint8_t     unit,
    uint8_t     id,
    uint8_t *   p_buf)
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
id	The report ID.	uint8_t
p_buf	A pointer to the buffer to write to.	uint8_t *

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
USBD_HID_ERROR_NOT_READY	Device is not operational.
USBD_HID_ERROR_BUSY	Transfer in progress.
Else	See Error Codes.

usbd_ghid_register_ntf

Use this function to register a notification function to be called when an OUT report is received or an IN report is sent.

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

Format

```
int usbd_ghid_register_ntf (
    uint8_t             unit,
    uint8_t             id,
    t_usbd_hid_ntf      ntf,
    t_usbd_hid_ntf_fn   ntf_fn,
    uint32_t            ntf_fn_param )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
id	The desired report ID.	uint8_t
ntf	The notification ID.	t_usbd_hid_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbd_hid_ntf_fn
ntf_fn_param	The notification parameter.	uint32_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
USBD_HID_ERROR	Operation failed.
Else	See Error Codes.

4.3 Keyboard Management

The functions are the following:

Function	Description
usbd_kbd_init()	Initializes the keyboard module and allocates the required resources.
usbd_kbd_start()	Starts the keyboard module.
usbd_kbd_stop()	Stops the keyboard module.
usbd_kbd_delete()	Deletes the keyboard module and releases the resources it used.
usbd_kbd_clear_report()	Clears all key presses from the current input report.
usbd_kbd_set_report()	Sends a raw report with the modifier keys and the key states.
usbd_kbd_get_led_state()	Gets the state of the LEDs.
usbd_kbd_set_key_state()	Sets the state of a key in the input report.
usbd_kbd_set_modifier()	Sets the key modifier code bits.
usbd_kbd_register_ntf()	Registers a notification function to be called when the LED state changes, or when a keyboard IN report is sent.

usbd_kbd_init

Use this function to initialize the keyboard module and allocate the required resources.

Format

```
int usbd_kbd_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_start

Use this function to start the keyboard module.

> **Note:** You must call **usbd_kbd_init()** before this to initialize the keyboard module.

Format

```
int usbd_kbd_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_stop

Use this function to stop the keyboard module.

Format

```
int usbd_kbd_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_delete

Use this function to remove the keyboard module and release the associated resources.

Format

```
int usbd_kbd_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_clear_report

Use this function to clear all key presses from the current input report.

This sets the keys to the state where no key is depressed.

Format

```
void usbd_kbd_clear_report ( uint8_t unit )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_set_report

Use this function to send a raw report with the modifier keys and the key states.

Format

```
int usbd_kbd_set_report (
    uint8_t     unit,
    uint8_t     modifier
    uint8_t *   p_keys,
    uint8_t     key_count )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
modifier	The key modifier.	uint8_t
p_keys	A pointer to the report.	uint8_t *
key_count	The number of keys pressed.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_get_led_state

Use this function to get the state of the LEDs.

You can register a notification request to discover when the state changes; see usbd_kbd_register_ntf().

Format

```
int usbd_kbd_get_led_state (
    uint8_t     unit,
    uint8_t *   p_led_state )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
p_led_state	Where to write the state of the LEDs.	uint8_t *

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_set_key_state

Use this function to set the state of a key in the input report.

Format

```
int  usbd_kbd_set_key_state (
    uint8_t   unit,
    uint8_t   key,
    uint8_t   state )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
key	The scan code of the key that has been changed.	uint8_t
state	The key state: • 0 = the key is released. • 1 = the key is pressed.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_set_modifier

Use this function to set the key modifier code bits.

This function translates to the Shift, Alt, and Ctrl keys being depressed.

Format

```
void usbd_kbd_set_modifier (
    uint8_t   unit,
    uint8_t   modifier )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
modifier	A byte containing the modifier bits to be set.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_kbd_register_ntf

Use this function to register a notification function to be called when the LED state changes, or when a keyboard IN report is sent.

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

Format

```
int usbd_kbd_register_ntf (
    uint8_t             unit,
    t_usbd_hid_ntf      ntf,
    t_usbd_hid_ntf_fn   ntf_fn,
    uint32_t            ntf_fn_param )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
ntf	The notification ID.	t_usbd_hid_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbd_hid_ntf_fn
ntf_fn_param	The notification parameter.	uint32_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

4.4 Mouse Management

The functions are the following:

Function	Description
usbd_mhid_init()	Initializes the mouse module and allocates the required resources.
usbd_mhid_start()	Starts the mouse module.
usbd_mhid_stop()	Stops the mouse module.
usbd_mhid_delete()	Deletes the mouse module and releases the resources it used.
usbd_mhid_clear_report()	Resets the X/Y relative coordinates and the button states in the report to 0.
usbd_mhid_set_report()	Sends a raw report with the mouse X/Y coordinates and button states.
usbd_mhid_set_btn_state()	Sets the state of a mouse button.
usbd_mhid_set_xy()	Sets the mouse X/Y coordinate change.
usbd_mhid_register_ntf()	Registers a notification function to be called when an IN report is sent.

usbd_mhid_init

Use this function to initialize the mouse module and allocate the required resources.

Format

```
int usbd_mhid_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_start

Use this function to start the mouse module.

> **Note:** You must call **usbd_mhid_init()** before this to initialize the mouse module.

Format

```
int usbd_mhid_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_stop

Use this function to stop the mouse module.

Format

```
int usbd_mhid_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_delete

Use this function to remove the mouse module and release the associated resources.

Format

```
int usbd_mhid_delete ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_clear_report

Use this function to reset the X/Y relative coordinates and the button states in the report to 0.

Format

```
void usbd_mhid_clear_report ( uint8_t unit )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_set_report

Use this function to send a raw report with the mouse X/Y coordinates and button states.

Format

```
int usbd_mhid_set_report (
    uint8_t   unit,
    int8_t    dx,
    int8_t    dy,
    uint8_t   buttons )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
dx	The X coordinate.	int8_t
dy	The Y coordinate.	int8_t
buttons	The button states.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_set_btn_state

Use this function to set the state of a mouse button.

Format

```
void usbd_mhid_set_btn_state (
    uint8_t   unit,
    uint8_t   btn,
    uint8_t   state )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
btn	The index of the button (starting from 0).	uint8_t
state	The button state, one of the following: • 0 = the button is not pressed. • 1 = the button is pressed.	uint8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_set_xy

Use this function to set the mouse X/Y coordinate change.

This is incrementive, meaning that if it is called twice before the USB host polls the data over the Interrupt channel, the second change is added to the previous change. When the coordinates are read, the changes are reset.

Format

```
void usbd_mhid_set_xy (
    uint8_t   unit,
    int8_t    dx,
    int8_t    dy )
```

Arguments

Parameter	Description	Type
unit	The unit ID.	uint8_t
dx	Delta X.	int8_t
dy	Delta Y.	int8_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_mhid_register_ntf

Use this function to register a notification function to be called when an IN report is sent.

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

Format

```
int usbd_mhid_register_ntf (
    uint8_t             unit,
    t_usbd_hid_ntf      ntf,
    t_usbd_hid_ntf_fn   ntf_fn,
    uint32_t            ntf_fn_param )
```

Arguments

Parameter	Description	Type
unit	The desired report ID.	uint8_t
ntf	The notification ID.	t_usbd_hid_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbd_hid_ntf_fn
ntf_fn_param	The notification parameter.	uint32_t

Return Values

Return value	Description
USBD_HID_SUCCESS	Successful execution.
Else	See Error Codes.

4.5 Error Codes

If a function executes successfully, it returns with USBD_HID_SUCCESS, a value of zero. The following table shows the meaning of the error codes.

Return Value	Value	Description
USBD_HID_SUCCESS	0x00	Successful execution.
USBD_HID_ERR_NOT_READY	0x01	Device is not operational.
USBD_HID_ERR_BUSY	0x02	Transfer in progress.
USBD_HID_ERR_INIT	0x03	Initialization error.
USBD_HID_ERR_RESOURCE	0x04	Resource allocation error (mutex, event or task).
USBD_HID_ERR_REG_CDRV	0x05	Class driver registration error.
USBD_HID_ERROR	0xFF	General error.

4.6 Types and Definitions

t_usbd_hid_ntf_fn

The **t_usbh_hid_ntf_fn** definition specifies the format of the notification function.

Format

```
typedef int   ( * t_usbd_hid_ntf_fn )( t_usbd_hid_ntf ntf )
```

Arguments

Argument	Description	Type
ntf	The event notified; see Notification Codes.	t_usbd_hid_ntf

Notification Codes

The callback notification definitions are as follows:

Name	Value	Description
USBD_GHID_IN_CB	1	IN report sent by generic device.
USBD_GHID_OUT_CB	0	OUT report received by generic device.
USBD_KBD_IN_CB	1	IN report sent by keyboard.
USBD_KBD_LED_CB	0	Keyboard LED state changed.
USBD_MOUSE_IN_CB	1	IN report sent by mouse.

LED Mask

The LED mask values are as follows:

Return Value	Description
KBDLED_NUMLOCK	1UL << 0
KBDLED_CAPSLOCK	1UL << 1
KBDLED_SCROLLOCK	1UL << 2
KBDLED_COMPOSE	1UL << 3
KBDLED_KANA	1UL << 4

Note: These are unique bits in a set of bit values.

Modifier Keys

This table shows the key modifier code bits for the **usbd_kbd_set_modifier()** function.

Return Value	Description
MODIFIER_LEFT_CTRL	1UL << 0
MODIFIER_LEFT_SHIFT	1UL << 1
MODIFIER_LEFT_ALT	1UL << 2
MODIFIER_LEFT_GUI	1UL << 3
MODIFIER_RIGHT_CTRL	1UL << 4
MODIFIER_RIGHT_SHIFT	1UL << 5
MODIFIER_RIGHT_ALT	1UL << 6
MODIFIER_RIGHT_GUI	1UL << 7

Note: These are unique bits in a set of bit values.

Key Scan Codes

The key codes are as follows:

Description	Value
KEY_NONE	0x00
KEY_ERRORROLLOVER	0x01
KEY_POSTFAIL	0x02
KEY_ERRORUNDEFINED	0x03
KEY_A	0x04
KEY_B	0x05
KEY_C	0x06
KEY_D	0x07
KEY_E	0x08
KEY_F	0x09
KEY_G	0x0A
KEY_H	0x0B
KEY_I	0x0C
KEY_J	0x0D
KEY_K	0x0E
KEY_L	0x0F
KEY_M	0x10
KEY_N	0x11
KEY_O	0x12
KEY_P	0x13
KEY_Q	0x14
KEY_R	0x15
KEY_S	0x16
KEY_T	0x17

Description	Value
KEY_U	0x18
KEY_V	0x19
KEY_W	0x1A
KEY_X	0x1B
KEY_Y	0x1C
KEY_Z	0x1D
KEY_1_EXCLAMATION_MARK	0x1E
KEY_2_AT	0x1F
KEY_3_NUMBER_SIGN	0x20
KEY_4_DOLLAR	0x21
KEY_5_PERCENT	0x22
KEY_6_CARET	0x23
KEY_7_AMPERSAND	0x24
KEY_8_ASTERISK	0x25
KEY_9_OPARENTHESIS	0x26
KEY_0_CPARENTHESIS	0x27
KEY_ENTER	0x28
KEY_ESCAPE	0x29
KEY_BACKSPACE	0x2A
KEY_TAB	0x2B
KEY_SPACEBAR	0x2C
KEY_MINUS_UNDERSCORE	0x2D
KEY_EQUAL_PLUS	0x2E
KEY_OBRACKET_AND_OBRACE	0x2F
KEY_CBRACKET_AND_CBRACE	0x30
KEY_BACKSLASH_VERTICAL_BAR	0x31
KEY_NONUS_NUMBER_SIGN_TILDE	0x32
KEY_SEMICOLON_COLON	0x33

Description	Value
KEY_SINGLE_AND_DOUBLE_QUOTE	0x34
KEY_GRAVE_ACCENT_AND_TILDE	0x35
KEY_COMMA_AND_LESS	0x36
KEY_DOT_GREATER	0x37
KEY_SLASH_QUESTION	0x38
KEY_CAPS_LOCK	0x39
KEY_F1	0x3A
KEY_F2	0x3B
KEY_F3	0x3C
KEY_F4	0x3D
KEY_F5	0x3E
KEY_F6	0x3F
KEY_F7	0x40
KEY_F8	0x41
KEY_F9	0x42
KEY_F10	0x43
KEY_F11	0x44
KEY_F12	0x45
KEY_PRINTSCREEN	0x46
KEY_SCROLL_LOCK	0x47
KEY_PAUSE	0x48
KEY_INSERT	0x49
KEY_HOME	0x4A
KEY_PAGEUP	0x4B
KEY_DELETE	0x4C
KEY_END1	0x4D
KEY_PAGEDOWN	0x4E
KEY_RIGHTARROW	0x4F

Description	Value
KEY_LEFTARROW	0x50
KEY_DOWNARROW	0x51
KEY_UPARROW	0x52
KEY_KEYPAD_NUM_LOCK_AND_CLEAR	0x53
KEY_KEYPAD_SLASH	0x54
KEY_KEYPAD_ASTERIKS	0x55
KEY_KEYPAD_MINUS	0x56
KEY_KEYPAD_PLUS	0x57
KEY_KEYPAD_ENTER	0x58
KEY_KEYPAD_1_END	0x59
KEY_KEYPAD_2_DOWN_ARROW	0x5A
KEY_KEYPAD_3_PAGEDN	0x5B
KEY_KEYPAD_4_LEFT_ARROW	0x5C
KEY_KEYPAD_5	0x5D
KEY_KEYPAD_6_RIGHT_ARROW	0x5E
KEY_KEYPAD_7_HOME	0x5F
KEY_KEYPAD_8_UP_ARROW	0x60
KEY_KEYPAD_9_PAGEUP	0x61
KEY_KEYPAD_0_INSERT	0x62
KEY_KEYPAD_DECIMAL_SEPARATOR_DELETE	0x63
KEY_NONUS_BACK_SLASH_VERTICAL_BAR	0x64
KEY_APPLICATION	0x65

5 Integration

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

5.1 OS Abstraction Layer

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.

This module uses the following OAL components:

OAL Resource	Number Required
Tasks	1
Mutexes	2 (The second is only required if you enable the keyboard module.)
Events	1

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. For full details of its functions, see the *HCC Base Platform Support Package User Guide*.

This module makes use of the following standard PSP functions:

Function	Package	Component	Description
psp_memcmp()	psp_base	psp_string	Compares two blocks of memory.
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.