Embedded USB Host HID Class Driver User Guide

Version 2.20

For use with USBH HID Class Driver versions 5.01 and above

Date: 20-Apr-2016 13:35

Table of Contents

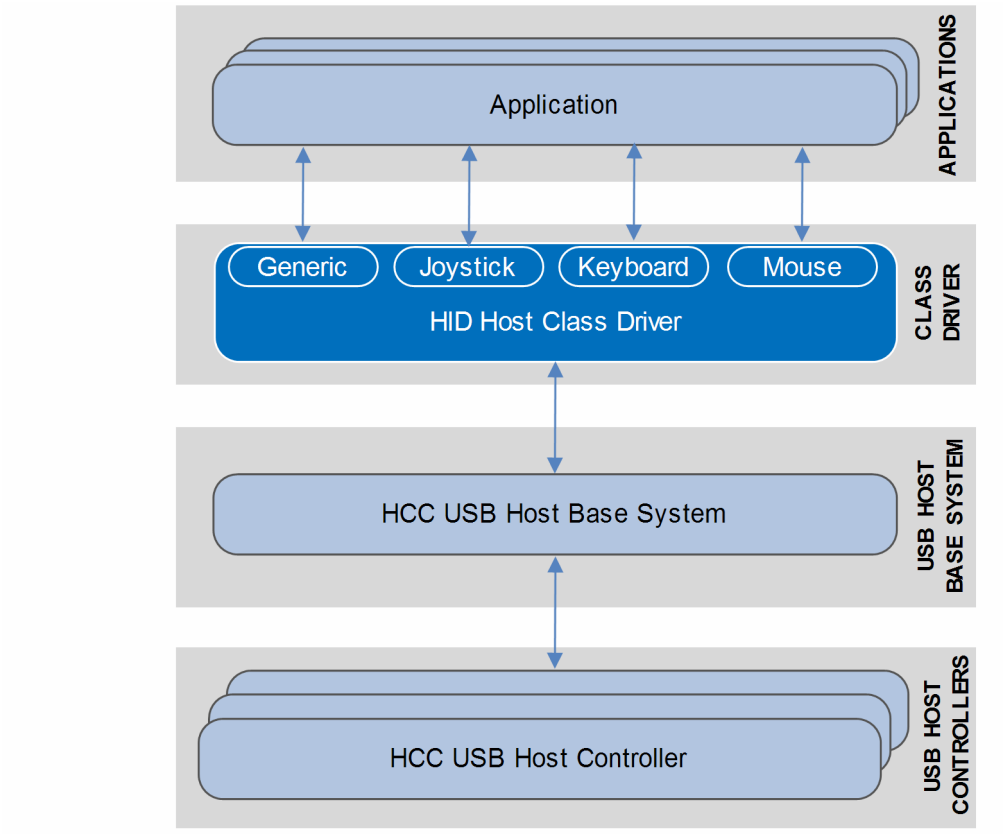
1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB host class driver to control Human Interface Device (HID) USB devices. HID devices include keyboards, mice, joysticks 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 **usbh_cd_hid** package provides a HID host class driver for a USB stack. Typically this is used for connecting devices over a USB link to a host.

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, see the *HCC USB Host Base System User Guide* that is shipped with the base system.

The package provides a set of API functions for controlling access to a device. These are described here, with separate sections for generic devices, joysticks, keyboards, and mice.

1.2 Feature Check

The main features of the Embedded USB Host HID Class Driver are the following:

- It conforms to the HCC Advanced Embedded Framework.
- It can be used with or without an RTOS.
- It is compatible with all HCC USB host controllers.
- It supports multiple devices connected simultaneously.
- It supports generic devices, joysticks and gamepads, keyboards, and mice.
- It has a sophisticated system for managing both input and output reports.
- It 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_hid	The USB device HID 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 HID Class Driver User Guide

This is this document.

1.4 Change History

This section includes recent changes to this product. For a list of all the changes, refer to the file **src/history/usb-host/usbh_cd_hid.txt** in the distribution package.

Version	Changes
5.01	Report items within one report can be distributed across different drivers, for example mouse and keyboard. Reports can be selected for processing by HID generic using new function **usbh_hid_generic_register_accept_report_ntf()**. The joystick driver now supports gamepads as well.
4.02	Generic disconnect was previously called incorrectly. Now it is called for all released units on the same disconnected device.
4.01	Added joystick support. Report items can now be retrieved for keyboard, mouse and joystick. Simplified and cleaned up report item decoding.
3.01	Added *key_count* variable to *t_kbd_report*, representing the number of valid elements in the keys buffer. Introduced HID_MAX_REPORT_ITEM_COUNT to accept HID devices with report items that are not required for the specific configuration. An example is a keyboard-only configuration with a device with generic report items. Fixed the following problems: • Corrected handling of report descriptors in case of array data. • Removed restriction on minimum number of generic HID reports.

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_usbh_hid.h	Module header file.
api_usbh_hid_generic.h	Generic device header file.
api_usbh_hid_joystick.h	Joystick and gamepad header file.
api_usbh_hid_kbd.h	Keyboard header file.
api_usbh_hid_mouse.h	Mouse header file.

2.2 Configuration Files

The following files in **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_usbh_hid.h	Module configuration file.
config_usbh_hid_generic.h	Generic device configuration file.
config_usbh_hid_joystick.h	Joystick and gamepad configuration file.
config_usbh_hid_kbd.h	Keyboard configuration file.
config_usbh_hid_mouse.h	Mouse configuration file.

2.3 HID System Files

The following files are in **src/usb-host/class-drivers/hid**. **These files should only be modified by HCC**.

File	Description
hid_parser.c	HID parser code.
hid_parser.h	HID parser header file.
hid_usage.h	HID usage values.
host_hid.c	Module code and internal public functions.
host_hid.h	Module code and internal public functions.
host_hid_generic.c	Generic device code.
host_hid_generic.h	Generic device header file.
host_hid_kbd.c	Joystick and gamepad code
host_hid_kbd.h	Joystick and gamepad header file.
host_hid_kbd.c	Keyboard code
host_hid_kbd.h	Keyboard header file.
host_hid_mouse.c	Mouse code.
host_hid_mouse.h	Mouse header file.

2.4 Version File

The file **src/version/ver_usbh_hid.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

The system configuration options are set in the configuration files described below. This section lists the available configuration options and their default values.

3.1 System Configuration Options

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

HID_MOUSE_ENABLE

Keep the default of 1 to enable mouse support. Set it to 0 if a mouse is not required.

HID_KBD_ENABLE

Keep the default of 1 to enable keyboard support. Set it to 0 if a keyboard is not required.

HID_GENERIC_ENABLE

Keep the default of 1 to enable generic HID device support. Set it to 0 to disable this.

HID_JOYSTICK_ENABLE

Keep the default of 1 to enable joystick/gamepad support. Set it to 0 to disable this.

HID_REPORT_DESCRIPTOR_MAX_SIZE

The maximum size of a report descriptor. The default is 512.

HID_PARSER_GLOBAL_PROPERTIES_STACK_DEPTH

During parsing, a report descriptor needs temporal storage for the global properties. The depth of the storage must be at least the maximum count of consecutive push instructions within a report descriptor, plus 1. This value must be 1 (the default) or higher.

HID_PARSER_MAXIMUM_REPORT_COUNT

The maximum number of reports within a report descriptor. The default is 4.

HID_PARSER_MAXIMUM_REPORT_ITEM_COUNT

The maximum number of report items within any report that the parser can process. The default is 256.

HID_ITEM_POOL_SIZE

Keep the default of 0 if you want the HID to calculate the pool size.

HID_REPORT_DATA_SIZE

The report buffer size. The default is 128.

3.2 Generic Device Options

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

MAX_GENERIC_UNITS

The maximum number of supported generic devices. The default is 1.

> **Note:** If you configure two devices, these take the unit IDs UID 0 and UID 1. (These are referred to by the parameter *uid* in the API functions.)

MAX_GENERIC_CB

The maximum number of callbacks available. Ideally, do not set this to less than MAX_GENERIC_UNITS. The default is 1.

RAW_REPORTS

Set this to 1 if you want to read/write raw reports. The default is 0.

MAX_GEN_REPORTS

The maximum number of generic device reports. The default is 2.

MAX_GENERIC_UNIT_REPORT_ITEMS

The number of report items a generic unit may have. The default is 8.

USBH_HID_GENERIC_BUFFER_SUPPORT

Keep the default of 1 to use generic buffers.

3.3 Joystick and Gamepad Options

Set the joystick configuration options in the file **src/config/config_usbh_hid_joystick.h**.

MAX_JOYSTICK_UNITS

The maximum number of joysticks and gamepads. The default is 1.

> **Note:** If you configure two joystick devices, these take the unit IDs UID 0 and UID 1. (These are referred to by the parameter *uid* in the API functions.)

MAX_JOYSTICK_BUTTONS

The maximum number of joystick buttons. The permitted range is 2 to 16 and the default is 16.

3.4 Keyboard Options

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

MAX_KBD_UNITS

The maximum number of keyboards. The default is 1.

> **Note:** If you configure two keyboard devices, these take the unit IDs UID 0 and UID 1. (These are referred to by the parameter *uid* in the API functions.)

MAX_KBD_KEYS

The maximum number of keys. The default is 6.

MAX_KBD_LEDS

The maximum number of LEDs. The default is 5.

MAX_KBD_MOD_KEYS

The maximum number of modifier keys. The default is 8.

3.5 Mouse Options

Set the mouse configuration options in the file **src/config/config_usbh_hid_mouse.h**.

MAX_MOUSE_UNITS

The maximum number of mice. The default is 1.

> **Note:** If you configure two mice, these take the unit IDs UID 0 and UID 1. (These are referred to by the parameter *uid* in the API functions.)

MAX_BUTTONS

The maximum number of mouse buttons. The default is 16.

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
usbh_hid_init()	Initializes the module and allocates the required resources.
usbh_hid_start()	Starts the module.
usbh_hid_stop()	Stops the module.
usbh_hid_delete()	Deletes the module and releases the resources it used.

usbh_hid_init

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

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

Format

```
int usbh_hid_init ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_hid_start

Use this function to start the HID module.

> **Note:** Call **usbh_hid_init()** before this to initialize the module.

Format

```
int usbh_hid_start ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_hid_stop

Use this function to stop the HID module.

Format

```
int usbh_hid_stop ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_hid_delete

Use this function to delete the HID module and release the associated resources.

Format

```
int usbh_hid_delete ( void )
```

Arguments

Parameter
None.

Return Values

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

4.2 General Management

This section describes the following functions that can be used to access any of the possible device types.

Function	Description
usbh_hid_get_report()	Gets a report.
usbh_hid_set_report()	Creates a report.
usbh_hid_get_feature_report()	Gets a feature report.
usbh_hid_set_feature_report()	Sets a feature report.
usbh_hid_get_report_descriptor()	Gets the report descriptor.
usbh_hid_get_exponent()	Gets an exponent.
usbh_hid_unit_get_system()	Gets the unit system used by a unit.
usbh_hid_read_item()	Reads the value of an item in a report.
usbh_hid_write_item()	Writes the value of an item to a report.
usbh_hid_set_idle()	Sets the idle time for a report.
usbh_hid_set_protocol()	Sets the boot or non-boot protocol.

usbh_hid_get_report

Use this function to get a report.

Format

```
int usbh_hid_get_report (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint16_t         length )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
report_id	The report ID.	uint8_t
buffer	A pointer to the data to read.	uint8_t *
length	The size of the buffer.	uint16_t

Return Values

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

usbh_hid_set_report

Use this function to create a report.

Format

```
int usbh_hid_set_report (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint16_t         length )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
report_id	The report ID.	uint8_t
buffer	A pointer to the data to send.	uint8_t *
length	The number of bytes to send.	uint16_t

Return Values

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

usbh_hid_get_feature_report

Use this function to get a feature report.

Format

```
int usbh_hid_get_feature_report (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint16_t         length )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
report_id	The report ID.	uint8_t
buffer	A pointer to the data to read.	uint8_t *
length	The size of the buffer.	uint16_t

Return Values

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

usbh_hid_set_feature_report

Use this function to set a feature report.

Format

```
int usbh_hid_set_feature_report (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint16_t         length )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
report_id	The report ID.	uint8_t
buffer	A pointer to the data to send.	uint8_t *
length	The size of the buffer.	uint16_t

Return Values

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

usbh_hid_get_report_descriptor

Use this function to get the report descriptor.

Format

```
t_usbh_port_hdl usbh_hid_get_report_descriptor (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t *        buffer,
    uint16_t         length )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
buffer	Where to write the report descriptor.	uint8_t *
length	The maximum length of the buffer.	uint16_t

Return Values

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

usbh_hid_unit_get_exponent

Use this function to get an exponent.

Format

```
int usbh_hid_unit_get_exponent (
    uint32_t   unit,
    uint8_t    extent,
    int8_t *   exponent )
```

Arguments

Parameter	Description	Type
unit	The unit.	uint32_t
extent	The extent.	uint8_t
exponent	The exponent.	int8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_hid_unit_get_system

Use this function to get the unit system used by a unit.

Format

```
int usbh_hid_unit_get_system(
    uint32_t   unit,
    uint8_t *  system )
```

Arguments

Parameter	Description	Type
unit	The unit.	uint32_t
system	A pointer to the unit system used.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_hid_read_item

Use this function to read the value of an item in a report.

Format

```
int usbh_hid_read_item (
    t_report_item *   ri,
    uint8_t *         buffer,
    uint8_t           index,
    int32_t *         value )
```

Arguments

Parameter	Description	Type
ri	A pointer to the report item.	t_report_item *
buffer	A pointer to the report data.	uint8_t *
index	The array index (in the case of an array report, otherwise this should be 0).	uint8_t
value	A pointer to the report item value.	int32_t *

Return Values

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

usbh_hid_write_item

Use this function to write the value of an item to a report.

Format

```
int usbh_hid_write_item (
    t_report_item *   ri,
    uint8_t *         buffer,
    uint8_t           index,
    int32_t           value )
```

Arguments

Parameter	Description	Type
ri	A pointer to the report item.	t_report_item *
buffer	A pointer to the report data.	uint8_t *
index	The array index (in the case of an array report, otherwise this should be 0).	uint8_t
value	The report item value.	int32_t

Return Values

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

usbh_hid_set_idle

Use this function to set the idle time for a report.

Format

```
int usbh_hid_set_idle (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          report_id,
    uint8_t          dur )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
report_id	The report ID.	uint8_t
dur	The idle time duration.	uint8_t

Return Values

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

usbh_hid_set_protocol

Use this function to set the boot or non-boot protocol.

Format

```
int usbh_hid_set_protocol (
    t_usbh_hid_hdl   hid_hdl,
    uint8_t          boot )
```

Arguments

Parameter	Description	Type
hid_hdl	The HID handle.	t_usbh_hid_hdl
boot	One of the following: • Zero = non-boot. • 1 = boot.	uint8_t

Return Values

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

4.3 Generic HID Management

Use the following functions to control generic devices including pointers, buttons, and sliders.

Function	Description
usbh_hid_generic_get_uid()	Gets the generic device unit ID from the port handle.
usbh_hid_generic_get_hid_hdl()	Gets the generic device's HID handle.
usbh_hid_generic_get_port_hdl()	Gets the generic device's port handle.
usbh_hid_generic_get_report()	Gets a pointer to a generic report.
usbh_hid_generic_get_value()	Gets the first report item of *usage_page/usage* and read its value from the buffer.
usbh_hid_generic_find_report()	Finds a report for a specified unit. The unit must be connected.
usbh_hid_generic_get_report_count()	Gets the number of reports available for a unit. The unit must be connected.
usbh_hid_generic_read_report()	Reads raw report data into a buffer. The unit must be connected.
usbh_hid_generic_write_report()	Sends an output report on the Control endpoint.
usbh_hid_generic_write_it_ep_report()	Sends an output report on the Interrupt endpoint. The unit must be connected.
usbh_hid_generic_present()	Checks whether a generic device is present.
usbh_hid_generic_register_cb()	Registers a callback function to be called when a new IN report arrives on a specific port.
usbh_hid_generic_register_ntf()	Registers a notification function for a specified event type for a generic device.

usbh_hid_generic_get_uid

Use this function to get the generic device unit ID from the port handle.

Format

```
int usbh_hid_generic_get_uid (
    t_usbh_port_hdl    port_hdl,
    t_usbh_unit_id *   p_uid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
p_uid	Where to write the unit ID.	t_usbh_unit_id *

Return Values

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

usbh_hid_generic_get_hid_hdl

Use this function to get the generic device's HID handle.

Format

```
t_usbh_hid_hdl usbh_hid_generic_get_hid_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
The HID handle.	Successful execution.
USBH_HID_HDL_INVALID	The HID handle is invalid.

usbh_hid_generic_get_port_hdl

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

Format

```
t_usbh_port_hdl usbh_hid_generic_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	The port handle is invalid.

usbh_hid_generic_get_report

Use this function to get a pointer to a generic report.

Format

```
int usbh_hid_generic_get_report (
    t_usbh_unit_id   uid,
    uint8_t          index,
    t_report * *     report )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
index	The index of the unit.	uint8_t
report	A pointer to the first report.	t_report * *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The report does not exist.

usbh_hid_generic_get_value

Use this function to retrieve the first report item of *usage_page*/*usage* and read its value from the buffer.

The unit must be connected and the *b_available* flag of the report found must be TRUE.

> **Note:** This function is only available if RAW_REPORTS is set to 0.

Format

```
int usbh_hid_generic_get_value (
    t_usbh_unit_id   uid,
    uint16_t         usage_page,
    uint8_t          usage,
    int32_t *        value )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
usage_page	The usage page.	uint16_t
usage	The usage.	uint8_t
value	Where to write the usage page value.	int32_t *

Return Values

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

usbh_hid_generic_find_report

Use this function to find a report for a specified unit. The unit must be connected.

Format

```
int usbh_hid_generic_find_report (
    t_usbh_unit_id   uid,
    t_report_type    report_type,
    uint8_t          report_id,
    t_report * *     report )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
report_type	The report type.	t_report_type
report_id	The report ID.	uint8_t
report	A pointer to the report.	t_report * *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The unit is not connected.
Else	See Error Codes.

usbh_hid_generic_get_report_count

Use this function to get the number of reports available for a unit. The unit must be connected.

Format

```
int usbh_hid_generic_get_report_count (
    t_usbh_unit_id   uid,
    uint8_t *        report_count )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
report_count	The number of reports.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The report count is not available.

usbh_hid_generic_read_report

Use this function to read raw report data into a buffer. The unit must be connected.

Read the data in one of two ways:

- By initiating a Get Report transfer on the control channel to get an up-to-date report.
- From an internal buffer (the last received data).

Format

```
int usbh_hid_generic_read_report (
    t_usbh_unit_id   uid,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint8_t          length )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
report_id	The report ID.	uint8_t
buffer	Where to put the input report.	uint8_t *
length	The length of the report to read.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The unit is not connected.
Else	See Error Codes.

usbh_hid_generic_write_report

Use this function to send an output report on the Control endpoint.

The unit must be connected.

Format

```
int usbh_hid_generic_write_report (
    t_usbh_unit_id   uid,
    uint8_t          report_id,
    uint8_t *        buffer,
    uint8_t          length )
```

Arguments

Parameter	Description	Type
uid	The unit ID	t_usbh_unit_id
report_id	The report ID.	uint8_t
buffer	A pointer to the output report.	uint8_t *
length	The length of the report to send.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The unit is not connected.
Else	See Error Codes.

usbh_hid_generic_it_ep_write_report

Use this function to send an output report on the Interrupt endpoint. The unit must be connected.

Format

```
int usbh_hid_generic_it_ep_write_report (
    t_usbh_unit_id   uid,
    uint8_t *        buffer,
    uint8_t          length )
```

Arguments

Parameter	Description	Type
uid	The unit ID	t_usbh_unit_id
buffer	A pointer to the output report.	uint8_t *
length	The length of the report to send.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The unit is not connected.
Else	See Error Codes.

usbh_hid_generic_present

Use this function to check whether a generic device is present.

Format

```
int usbh_hid_generic_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 generic device is present.
1	A generic device is present.
Else	See Error Codes.

usbh_hid_generic_register_cb

Use this function to register a callback function to be called when a new IN report arrives on a specific port.

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

Format

```
int usbh_hid_generic_register_cb (
    t_hid_generic_cb   cb,
    uint32_t           param)
```

Arguments

Parameter	Description	Type
cb	The callback function.	t_hid_generic_cb
param	The parameter to pass to the callback function.	uint32_t

Return Values

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

usbh_hid_generic_register_ntf

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

When a generic 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_hid_generic_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_SUCCESS	Successful execution.
Else	See Error Codes.

4.4 Joystick and Gamepad Management

Use the following functions to control joysticks and gamepads.

Function	Description
usbh_hid_joystick_get_uid()	Gets the unit ID value of the first joystick/gamepad assigned to the given port.
usbh_hid_joystick_get_hid_hdl()	Gets the joystick or gamepad's HID handle.
usbh_hid_joystick_get_port_hdl()	Gets the joystick or gamepad's port handle.
usbh_hid_joystick_get_report()	Gets joystick/gamepad input report data.
usbh_hid_joystick_get_report_item()	Gets a joystick/gamepad report item.
usbh_hid_joystick_get_supported_buttons()	Gets a connected joystick or gamepad's button configuration.
usbh_hid_joystick_get_supported_axes()	Gets a connected joystick or gamepad's axis configuration
usbh_hid_joystick_present()	Checks whether a joystick/gamepad is present.
usbh_hid_joystick_register_ntf()	Registers a notification function for a specified event type for a joystick/gamepad.

usbh_hid_joystick_get_uid

Use this function to get the unit ID value of the first joystick/gamepad assigned to the given port. The unit must be connected.

This can be useful if the HID report belongs to another interface that has assigned a different class driver. In this case that other class driver can request the unit ID based on the port handle in order to be able to access the reports/items.

> **Note:** Many units may be assigned to one port, but only one unit of a subclass may be assigned to one interface.

Format

```
int usbh_hid_joystick_get_uid (
    t_usbh_port_hdl    port_hdl,
    t_usbh_unit_id *   p_uid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
p_uid	Where to write the unit ID.	t_usbh_unit_id *

Return Values

Return value	Description
The unit ID.	Successful execution.
Else	See Error Codes.

usbh_hid_joystick_get_hid_hdl

Use this function to get the joystick/gamepad HID handle.

Format

```
t_usbh_hid_hdl usbh_hid_joystick_get_hid_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
HID handle.	Successful execution.
USBH_HID_HDL_INVALID	The HID handle is invalid.

usbh_hid_joystick_get_port_hdl

Use this function to get the joystick/gamepad port handle.

Format

```
t_usbh_port_hdl usbh_hid_joystick_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	The port handle is invalid.

usbh_hid_joystick_get_report

Use this function to get joystick/gamepad input report data.

The report shows the state of the control buttons and which buttons are currently depressed.

Format

```
int usbh_hid_joystick_get_report (
    t_usbh_unit_id        uid,
    t_joystick_report *   p_joystick_report )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_joystick_report	A pointer to a t_joystick_report structure holding joystick/gamepad input report data.	t_joystick_report *

Return Values

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

usbh_hid_joystick_get_report_item

Use this function to get a joystick/gamepad report item.

Format

```
int usbh_hid_joystick_get_report_item (
    t_usbh_unit_id      uid,
    uint8_t             item_index,
    t_report_item * *   report_item )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
item_index	A USBH_HID_JOYSTICK_REPORT_ITEM value.	uint8_t
report_item	On return, the report_item structure of the report item at *item_index*.	t_report_item * *

Return Values

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

usbh_hid_joystick_get_supported_buttons

Use this function to get a connected joystick or gamepad's button configuration. This shows the supported buttons as a bitmap, for example:

- Bit 0 (1U << 0) is non-zero if button 0 is supported.
- Bit 1 (1U << 1) is non-zero if button 1 is supported.

And so on.

Format

```
int usbh_hid_joystick_get_supported_buttons (
    t_usbh_unit_id   uid,
    uint16_t *       p_supported_buttons,
    uint8_t *        p_number_of_buttons )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_supported_axes	On return, a pointer to the bitmap .	uint16_t *
p_number_of_axes	On return, a pointer to the number of buttons on the device.	uint8_t *

Return Values

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

usbh_hid_joystick_get_supported_axes

Use this function to get a connected joystick's axis configuration. This shows the supported axes as a bitmap, for example:

- Bit (1U << USBH_HID_JOYSTICK_REPORT_ITEM_X) is non-zero if axis X is supported.
- Bit (1U << USBH_HID_JOYSTICK_REPORT_ITEM_Y) is non-zero if axis Y is supported.
- Bit (1U << USBH_HID_JOYSTICK_REPORT_ITEM_Z) is non-zero if axis Z is supported.

And so on.....

Format

```
int usbh_hid_joystick_get_supported_axes (
    t_usbh_unit_id   uid,
    uint16_t *       p_supported_axes,
    uint8_t *        p_number_of_axes )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_supported_axes	On return, a pointer to the bitmap .	uint16_t *
p_number_of_axes	On return, a pointer to the number of axes on the device.	uint8_t *

Return Values

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

usbh_hid_joystick_present

Use this function to check whether a joystick/gamepad is present.

Format

```
int usbh_hid_joystick_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 joystick/gamepad is present.
1	A joystick/gamepad is present.
Else	See Error Codes.

usbh_hid_joystick_register_ntf

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

When a joystick/gamepad is connected or disconnected, or a joystick RX event occurs, 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_hid_joystick_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 type.	t_usbh_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbh_ntf_fn

Return Values

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

4.5 Keyboard Management

Use the following functions to control keyboards.

Function	Description
usbh_hid_kbd_get_uid()	Gets the unit ID value of the first keyboard assigned to the given port. The unit must be connected.
usbh_hid_kbd_get_hid_hdl()	Gets the keyboard's HID handle.
usbh_hid_kbd_get_port_hdl()	Gets the keyboard's port handle.
usbh_hid_kbd_get_report()	Gets keyboard input report data. The unit must be connected.
usbh_hid_kbd_get_in_report_item()	Gets a keyboard IN report item.
usbh_hid_kbd_get_out_report_item()	Gets a keyboard OUT report item.
usbh_hid_kbd_set_leds()	Sets the state of the keyboard LEDs.
usbh_hid_kbd_present()	Checks whether a keyboard is present.
usbh_hid_kbd_register_ntf()	Registers a notification function for a specified event type for a keyboard.

usbh_hid_kbd_get_uid

Use this function to get the unit ID value of the first keyboard assigned to the given port. The unit must be connected.

This can be useful if the HID report belongs to another interface that has assigned a different class driver. In this case that other class driver can request the unit ID based on the port handle in order to be able to access the reports/items.

> **Note:** Many units may be assigned to one port, but only one unit of a subclass may be assigned to one interface.

Format

```
int usbh_hid_kbd_get_uid (
    t_usbh_port_hdl    port_hdl,
    t_usbh_unit_id *   p_uid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
p_uid	On return, a pointer to the unit ID.	t_usbh_unit_id *

Return Values

Return value	Description
The unit ID.	Successful execution.
Else	See Error Codes.

usbh_hid_kbd_get_hid_hdl

Use this function to get the keyboard HID handle.

Format

```
t_usbh_hid_hdl usbh_hid_kbd_get_hid_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
HID handle.	Successful execution.
USBH_HID_HDL_INVALID	The HID handle is invalid.

usbh_hid_kbd_get_port_hdl

Use this function to get the keyboard port handle.

Format

```
t_usbh_port_hdl usbh_hid_kbd_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	The port handle is invalid.

usbh_hid_kbd_get_report

Use this function to get keyboard input report data. The unit must be connected.

The report shows the state of the control keys and which keys are currently depressed.

For an example of this function in use, see the Code Example.

Format

```
int usbh_hid_kbd_get_report (
    t_usbh_unit_id   uid,
    t_kbd_report *   p_kbd_report )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_kbd_report	A pointer to a t_kbd_report structure holding keyboard input report data.	t_kbd_report *

Return Values

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

usbh_hid_kbd_get_in_report_item

Use this function to get a keyboard IN report item.

Format

```
int usbh_hid_kbd_get_in_report_item (
    t_usbh_unit_id      uid,
    uint8_t             item_index,
    t_report_item * *   report_item )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
item_index	A USBH_HID_KEYBOARD_IN_REPORT_ITEM value.	uint8_t
report_item	On return, the report_item structure of the report item at *item_index*.	t_report_item * *

Return Values

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

usbh_hid_kbd_get_out_report_item

Use this function to get a keyboard OUT report item.

Format

```
int usbh_hid_kbd_get_out_report_item (
    t_usbh_unit_id      uid,
    uint8_t             item_index,
    t_report_item * *   report_item )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
item_index	A USBH_HID_KEYBOARD_OUT_REPORT_ITEM value.	uint8_t
report_item	On return, the report_item structure of the report item at *item_index*.	t_report_item * *

Return Values

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

usbh_hid_kbd_set_leds

Use this function to set the state of the keyboard LEDs.

A value of 0 switches the LED off, a value of 1 switches it on.

Format

```
int usbh_hid_kbd_set_leds (
    t_usbh_unit_id   uid,
    uint8_t          leds )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
leds	The LED state mask, one of the following (see Keyboard LED Codes): • USBH_HID_KBD_LED_NUM • USBH_HID_KBD_LED_CAPS • USBH_HID_KBD_LED_SCROLL • USBH_HID_KBD_LED_COMPO • USBH_HID_KBD_LED_KANA	uint8_t

Return Values

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

usbh_hid_kbd_present

Use this function to check whether a keyboard is present.

Format

```
int usbh_hid_kbd_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 keyboard is present.
1	A keyboard is present.
Else	See Error Codes.

usbh_hid_kbd_register_ntf

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

When a keyboard is connected or disconnected, or a keyboard RX event occurs, the notification function is called.

For an example of this function in use, see the Code Example.

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

Format

```
int usbh_hid_kbd_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 type.	t_usbh_ntf
ntf_fn	The notification function to use when an event occurs.	t_usbh_ntf_fn

Return Values

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

4.6 Mouse Management

Use the following functions to control a mouse.

Function	Description
usbh_hid_mouse_get_uid()	Gets the unit ID value of the first mouse assigned to the given port. The unit must be connected.
usbh_hid_mouse_get_hid_hdl()	Gets the mouse's HID handle.
usbh_hid_mouse_get_port_hdl()	Gets the mouse's port handle.
usbh_hid_mouse_get_report()	Gets mouse input report data.
usbh_hid_mouse_get_in_report_item()	Gets a mouse report item.
usbh_hid_mouse_present()	Checks whether a mouse is present.
usbh_hid_mouse_register_ntf()	Registers a notification function for a specified event type for a mouse.

usbh_hid_mouse_get_uid

Use this function to get the unit ID value of the first mouse assigned to the given port. The mouse must be connected.

This can be useful if the HID report belongs to another interface that has assigned a different class driver. In this case that other class driver can request the unit ID based on the port handle in order to be able to access the reports/items.

> **Note:** Many units may be assigned to one port, but only one unit of a subclass may be assigned to one interface.

Format

```
int usbh_hid_mouse_get_uid (
    t_usbh_port_hdl    port_hdl,
    t_usbh_unit_id *   p_uid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
p_uid	On return, a pointer to the unit ID.	t_usbh_unit_id *

Return Values

Return value	Description
The unit ID.	Successful execution.
Else	See Error Codes.

usbh_hid_mouse_get_hid_hdl

Use this function to get the mouse HID handle.

Format

```
t_usbh_hid_hdl usbh_hid_mouse_get_hid_hdl ( t_usbh_unit_id uid )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

Return Values

Return value	Description
The HID handle.	Successful execution.
USBH_HID_HDL_INVALID	The HID handle is invalid.

usbh_hid_mouse_get_port_hdl

Use this function to get the mouse port handle.

Format

```
t_usbh_port_hdl usbh_hid_mouse_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	The port handle is invalid.

usbh_hid_mouse_get_report

Use this function to get mouse input report data.

Format

```
int usbh_hid_mouse_get_report  (
    t_usbh_unit_id     uid,
    t_mouse_report *   p_mouse_report )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_mouse_report	A pointer to the structure holding mouse input report data.	t_mouse_report *

Return Values

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

usbh_hid_mouse_get_report_item

Use this function to get a mouse report item.

Format

```
int usbh_hid_mouse_get_report_item (
    t_usbh_unit_id      uid,
    uint8_t             item_index,
    t_report_item * *   report_item )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
item_index	A USBH_HID_MOUSE_REPORT_ITEM value.	uint8_t
report_item	On return, the report_item structure of the report item at *item_index*.	t_report_item * *

Return Values

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

usbh_hid_mouse_present

Use this function to check whether a mouse is present.

Format

```
int usbh_hid_mouse_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 mouse is present.
1	A mouse is present.
Else	See Error Codes.

usbh_hid_mouse_register_ntf

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

When a mouse is connected or disconnected, or a mouse RX event occurs, 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_hid_mouse_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_SUCCESS	Successful execution.
Else	See Error Codes.

4.7 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.

4.8 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**.

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

The additional notification codes provided by this module are as follows:

Notification	Value	Description
USBH_NTF_HID_JOYSTICK_RX	USBH_NTF_CD_BASE + 0	Joystick RX notification.
USBH_NTF_HID_KBD_RX	USBH_NTF_CD_BASE + 0	Keyboard RX notification.
USBH_NTF_HID_MOUSE_RX	USBH_NTF_CD_BASE + 0	Mouse RX notification.

t_hid_generic_cb

The **t_hid_generic_cb** typedef defines the callback function that may be called when a new IN report arrives on a specific port.

Format

```
void (* t_hid_generic_cb) (
    t_usbh_port_hdl   port_hdl,
    uint32_t          param)
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
param	The parameter passed.	uint32_t

Report Item Indexes

Joystick and Gamepad

The joystick/gamepad report item indexes are as follows:

Notification	Value	Description
USBH_HID_JOYSTICK_REPORT_ITEM_X	0	X axis.
USBH_HID_JOYSTICK_REPORT_ITEM_Y	1	Y axis.
USBH_HID_JOYSTICK_REPORT_ITEM_Z	2	Rotation axis.
USBH_HID_JOYSTICK_REPORT_ITEM_RX	3	RX axis.
USBH_HID_JOYSTICK_REPORT_ITEM_RY	4	RY axis.
USBH_HID_JOYSTICK_REPORT_ITEM_RZ	5	RZ axis.
USBH_HID_JOYSTICK_REPORT_ITEM_SLIDER	6	Slider control.
USBH_HID_JOYSTICK_REPORT_ITEM_DIAL	7	Dial control.
USBH_HID_JOYSTICK_REPORT_ITEM_WHEEL	8	Wheel control.
USBH_HID_JOYSTICK_REPORT_ITEM_HAT_SWITCH	9	Hat switch control.
USBH_HID_JOYSTICK_REPORT_ITEM_BUTTON(index)	((index) + 10)	Button states.

Keyboard

The keyboard report item indexes are as follows:

Notification	Value	Description
USBH_HID_KBD_IN_REPORT_ITEM_KEY	0	Keys pressed.
USBH_HID_KBD_IN_REPORT_ITEM_MODKEY(x)	((x) + 1)	Modifier.
USBH_HID_KBD_OUT_REPORT_ITEM_LEDS	0	LED states.

Mouse

The mouse report item indexes are as follows:

Notification	Value	Description
USBH_HID_MOUSE_REPORT_ITEM_X	0	X axis.
USBH_HID_MOUSE_REPORT_ITEM_Y	1	Y axis.
USBH_HID_MOUSE_REPORT_ITEM_WHEEL	2	Wheel state.
USBH_HID_MOUSE_REPORT_ITEM_BUTTON(x)	((x) + 3)	Button states.

t_joystick_report

The *t_joystick_report* structure defines a joystick report, with entries as shown below:

Notification	Description	Type
axes [USBH_HID_JOYSTICK_MAX_AXES]	The index range: USBH_HID_JOYSTICK_REPORT_ITEM_X .. USBH_HID_JOYSTICK_REPORT_ITEM_WHEEL	int32_t
hat_switch_valid	Hat switch control valid.	int8_t
hat_switch	Hat switch control.	int32_t
buttons	The button positions.	uint16_t

Keyboard LED Codes

The LED codes are as follows:

LED	Value	Description
USBH_HID_KBD_LED_NUM	1U << 0	Num lock LED.
USBH_HID_KBD_LED_CAPS	1U << 1	CAPS lock LED.
USBH_HID_KBD_LED_SCROLL	1U << 2	Scroll lock LED.
USBH_HID_KBD_LED_COMPO	1U << 3	Compose lock LED.
USBH_HID_KBD_LED_KANA	1U << 4	Kana lock LED.

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

t_kbd_report

The *t_kbd_report* structure is used to report on the state of the Control keys and which keys are depressed (only six pressed keys can be present at any time in the key buffer).

Element	Type	Description
control_keys	uint16_t	A control key code (see the table below).
keys[MAX_KBD_KEYS]	uint8_t	The key array.
key_count	uint8_t	The number of keys.

Control Key Codes

The Control key codes are as follows:

Control Key	Value
USBH_HID_KBD_KEY_LSHIFT	1U << 0
USBH_HID_KBD_KEY_RSHIFT	1U << 1
USBH_HID_KBD_KEY_RALT	1U << 2
USBH_HID_KBD_KEY_LALT	1U << 3
USBH_HID_KBD_KEY_RCTRL	1U << 4
USBH_HID_KBD_KEY_LCTRL	1U << 5
USBH_HID_KBD_KEY_RGUI	1U << 6
USBH_HID_KBD_KEY_LGUI	1U << 7

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

Key Scan Codes

The key codes are as follows:

Key	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
KEY_U	0x18

Key	Value
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
KEY_SINGLE_AND_DOUBLE_QUOTE	0x34

Key	Value
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
KEY_LEFTARROW	0x50

Key	Value
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

Unit Systems

The unit systems are as follows:

Unit System	Value	Description
USBH_HID_UNIT_SYSTEM_NONE	0	No system in use.
USBH_HID_UNIT_SYSTEM_SI_LINEAR	1	SI linear.
USBH_HID_UNIT_SYSTEM_SI_ROTATION	2	SI with rotation.
USBH_HID_UNIT_SYSTEM_ENGLISH_LINEAR	3	English linear.
USBH_HID_UNIT_SYSTEM_ENGLISH_ROTATION	4	English with rotation.

5 Code Example

This code example shows how to set up and handle keyboard event notification.

```
/* Code sample to show how to set up a callback that is called */
/* whenever data is received from the first attached keyboard */

rc = usbh_hid_kbd_register_ntf ( 0u , USBH_NTF_HID_KBD_RX, usbh_hid_kbd_ntf );

/* Code sample of a possible callback function implementation */
/* to handle a keyboard input received notification. */
/* This function prints out the state of the control keys and each key pressed */

int usbh_hid_kbd_ntf ( t_usbh_unit_id uid, t_usbh_ntf ntf )
{
    int  rc;
    int  i,n;
    t_kbd_report kbd_report;
    char  print_buf[80], print_buf2[10];

    (void)uid;
    (void)ntf;

    /* Read the report */

    rc = usbh_hid_kbd_get_report( 0, &kbd_report );

    if ( rc == USBH_SUCCESS )
    {
        n = 0;
        /* Add control key state to the print buffer */
        sprintf( print_buf, "Control keys = 0x%x  ; Pressed keys : ", kbd_report.control_keys );

        /* Add each pressed key to the print buffer */
        for ( i = 0 ; i < MAX_KBD_KEYS ; i++ )
        {
            if ( kbd_report.keys[i] != 0u )
            {
                 sprintf( print_buf2, "%d ", kbd_report.keys[i] );
                 strcat( print_buf, print_buf2 );
                 n++;
            }
        }

        sprintf( print_buf2, "(%d) \r\n", n );
        strcat( print_buf, print_buf2 );
        _print( print_buf );  /* Print the result */
    }
}
```

6 Integration

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

6.1 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.

The HID module uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1 per module type (see note below)
Events	0

Note: The generic device, joystick, keyboard, and mouse modules each use one mutex. For each module enabled in your application, a mutex must be allocated.

6.2 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 module 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.

The module makes use of the following standard PSP macro:

Macro	Package	Component	Description
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.