Embedded USB Host Raw Class Driver User Guide

Version 1.50

For use with USBH Raw Class Driver versions 4.02 and above

Exported on 10/18/2018

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

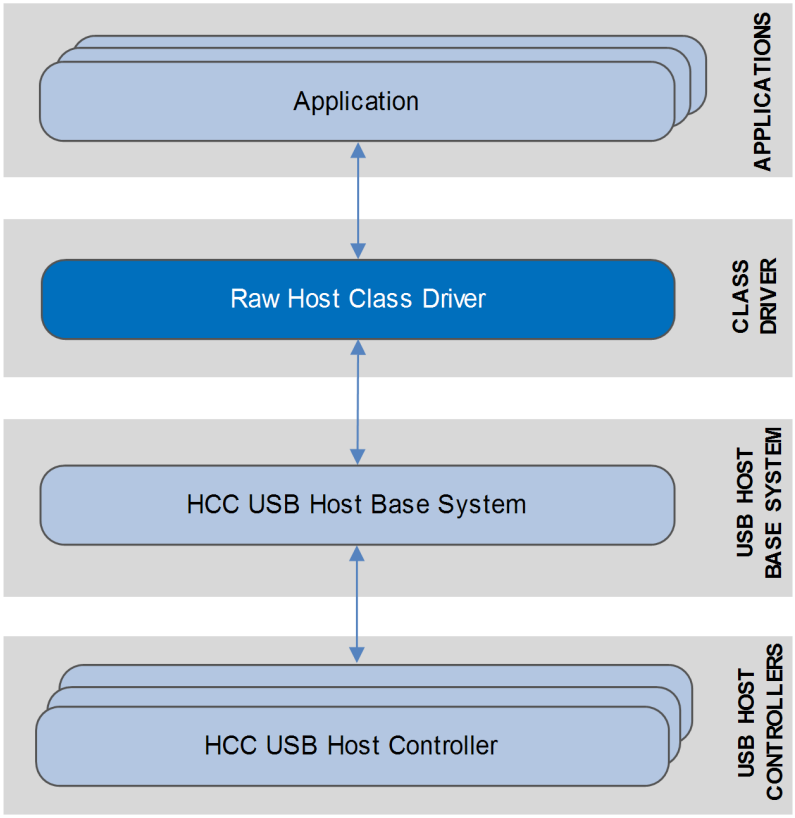
Note: To download this manual as a PDF, see USB Host PDFs.

1.1 Introduction

This guide is for those who want to implement an embedded USB host class driver to control Raw transfers. The **usbh_cd_raw** package provides a Raw host class driver for a USB stack. Typically this is used for connecting devices over a USB link to a host.

Raw transfers are used for infrequent transfers of larger amounts of data. The transfer takes place only when bandwidth is available, so may be delayed when other applications are using the bandwidth. Raw transfers may be used to support devices like printers and scanners. Data delivery is guaranteed by means of CRC error detection and limited retries. Raw transfers are only supported by full and high speed devices. Multiple channels can be used on the same interface.

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, refer to 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 module management and channel management functions.

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 a raw interface to a connected USB device.
- 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_raw	The USB device Raw 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 Raw Class Driver User Guide

This is this document.

1.4 Change History

This section describes past changes to this manual.

- To download this manual or a PDF describing an earlier software version, see USB Host PDFs.
- For the history of changes made to the package code itself, see History: usbh_cd_raw.

The current version of this manual is 1.50. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.50	2018-10-18	4.02	Corrected functions **usbh_raw_read_int()** and **usbh_raw_write_int()**.
1.40	2018-06-21	4.02	Removed three configuration options, added USBH_RAW_MAX_INT_CHANNELS_PER_INTERFACE Added functions: **usbh_raw_read_int(), usbh_raw_read_int_state(),** **usbh_raw_write_int()** and **usbh_raw_write_int_state()** Added *t_usbh_raw_setup_data* and *Request Type Bitmaps sections.*
1.30	2017-06-19	4.01	New *Change History* format.
1.20	2016-04-20	4.01	Added function group descriptions to API.
1.10	2015-11-27	2.07	Added new parameters to read-write functions.
1.00	2015-04-22	2.05	First release.

2 Source File List

The following sections describe 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_raw.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_raw.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 File

The file **src/usb-host/class-drivers/raw/usbh_raw.c** is the main code for the Raw class driver. **This file should only be modified by HCC.**

2.4 Version File

The file **src/version/ver_usbh_raw.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_raw.h**. This section lists the available options and their default values.

USBH_RAW_MAX_UNITS

The maximum number of device instances the class driver can handle concurrently. This is the maximum number of devices allowed to run in parallel. The default is 2.

USBH_RAW_MAX_BULK_CHANNELS_PER_INTERFACE

The maximum number of Bulk channels per interface. The default is 2.

USBH_RAW_MAX_INT_CHANNELS_PER_INTERFACE

The maximum number of INT channels per interface. The default is 2.

USBH_RAW_CLASS

The USB class value used to identify compatible interfaces this class driver can handle. The default is 0xFE.

USBH_RAW_SUBCLASS

The USB subclass value used to identify compatible interfaces this class driver can handle. The default is 0x01.

USBH_RAW_PROTOCOL

The USB protocol value used to identify compatible interfaces this class driver can handle. The default is 0x07.

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_raw_init()	Initializes the module and allocates the required resources.
usbh_raw_start()	Starts the module.
usbh_raw_stop()	Stops the module.
usbh_raw_delete()	Deletes the module and releases the resources it used.

usbh_raw_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_raw_init ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_raw_start

Use this function to start the class driver.

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

Format

```
int usbh_raw_start ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_raw_stop

Use this function to stop the class driver.

Format

```
int usbh_raw_stop ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_raw_delete

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

Format

```
int usbh_raw_delete ( void )
```

Arguments

Parameter
None.

Return Values

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

4.2 Channel Management

The functions are the following:

Function	Description
usbh_raw_read()	Starts a read transfer (from device to host).
usbh_raw_read_ctrl()	Starts a read transfer (from device to host) on the Control channel.
usbh_raw_read_ep()	Starts a read transfer (from device to host) by endpoint number instead of the channel.
usbh_raw_read_state()	Checks the status of the ongoing read operation.
usbh_raw_read_state_ep()	Checks the status of the ongoing **usbh_raw_read_ep()** operation.
usbh_raw_read_int()	Starts a read transfer on the Interrupt channel.
usbh_raw_read_int_state()	Checks the state of a previous read on the Interrupt channel.
usbh_raw_write()	Starts a write transfer (from host to device).
usbh_raw_write_ctrl()	Starts a write transfer (from host to device) on the Control channel.
usbh_raw_write_ep()	Starts a write transfer (from host to device) by endpoint number instead of the channel.
usbh_raw_set_write_state()	Checks the status of the ongoing write operation.
usbh_raw_set_write_state_ep()	Checks the status of the ongoing **usbh_raw_write_ep()** operation.
usbh_raw_write_int()	Starts a write transfer on the Interrupt channel.
usbh_raw_write_int_state()	Checks the state of a previous write on the Interrupt channel.
usbh_raw_get_ep_number()	Gets a channel's endpoint number.
usbh_raw_present()	Checks whether a compatible device is connected.
usbh_raw_get_port_hdl()	Gets the port handle.
usbh_raw_register_ntf()	Registers a notification function for a specified event type.

usbh_raw_read

Use this function to start a read transfer (from device to host).

> **Note:** This function starts the transfer but does not wait for it to end. To check for completion, call **usbh_raw_read_state()**.

Format

```
int usbh_raw_read (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t *        pc_dst,
    uint32_t         blen,
    uint8_t          skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
pc_dst	A pointer to the destination buffer.	uint8_t *
blen	The size of the buffer.	uint32_t
skip_zlp	If the requested receive length is a multiple of the USB packet size then: • If this is not set, the receive will only return after a terminating zero length packet is received. • If this is set, the receive will complete as soon as the requested data length has been received.	uint8_t

Return Values

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

usbh_raw_read_ctrl

Use this function to start a read transfer (from device to host) on the Control channel.

Format

```
int usbh_raw_read_ctrl (
    t_usbh_unit_id            uid,
    t_usbh_raw_setup_data *   p_setup_data )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_setup_data	A pointer to the buffer that holds the SETUP data and buffer.	t_usbh_raw_setup_data *

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
Else	See Error Codes.

usbh_raw_read_ep

Use this function to start a read transfer (from device to host) by endpoint number instead of the channel.

> **Note:** This function starts the transfer but does not wait for it to end. To check for completion, call **usbh_raw_read_state_ep()**.

Format

```
int usbh_raw_read_ep (
    t_usbh_unit_id   uid,
    uint8_t          ep,
    uint8_t *        pc_dst,
    uint32_t         blen,
    uint8_t          skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ep	The endpoint number. The range is 0x81 .. 0x84.	uint8_t
pc_dst	A pointer to the destination buffer.	uint8_t *
blen	The length of the buffer.	uint32_t
skip_zlp	If the requested receive length is a multiple of the USB packet size then: • If this is not set, the receive will only return after a terminating zero length packet is received. • If this is set, the receive will complete as soon as the requested data length has been received.	uint8_t

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
Else	See Error Codes.

usbh_raw_read_state

Use this function to check the status of the ongoing read operation.

Format

```
int usbh_raw_read_state (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t          b_block,
    uint32_t *       p_rlen )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
b_block	Set this to 1 if this call needs to block (wait until the read finishes). Ignore this in non-OS mode.	uint8_t
p_rlen	A pointer to the number of bytes read.	uint32_t *

Return Values

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

usbh_raw_read_state_ep

Use this function to check the status of the ongoing **usbh_raw_read_ep()** operation.

Format

```
int usbh_raw_read_state_ep (
    t_usbh_unit_id   uid,
    uint8_t          ep,
    uint8_t          b_block,
    uint32_t *       p_rlen  )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ep	The endpoint number. The range is 0x81 .. 0x84.	uint8_t
b_block	Set this to 1 if this call needs to block (wait until the read finishes). Ignore this in non-OS mode.	uint8_t
p_rlen	A pointer to the number of bytes read.	uint32_t *

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_BUSY	Operation is still ongoing.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
Else	See Error Codes.

usbh_raw_read_int

Use this function to start a read transfer on the Interrupt channel.

> **Note:** To check for completion, call **usbh_raw_read_int_state( )** .

Format

```
int usbh_raw_read_int (
    t_usbh_unit_id   uid
    uint8_t          channel,
    uint8_t *        p_buf,
    uint32_t         blen,
    uint8_t          b_skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number (the interface). The range is 0 to 3.	uint8_t
p_buf	A pointer to the destination buffer.	uint8_t *
blen	The length of the buffer.	uint32_t
b_skip_zlp	Set this to 1 to skip zero length packets.	uint8_t

Return Values

Return value	Description
USBD_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
USBH_RAW_ERROR	Operation failed.

usbh_raw_read_int_state

Use this function to check the state of a previous read on the Interrupt OUT endpoint.

> **Note:** To check for completion, call **usbh_raw_read_int_state( )** .

Format

```
int usbh_raw_read_int_state (
    t_usbh_unit_id   uid
    uint8_t          channel,
    uint8_t          b_block,
    uint32_t *       p_rlen )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number (the interface). The range is 0 to 3.	uint8_t
b_block	Set this non-zero to wait until the IN transfer finishes. Ignore this in non-OS mode.	uint8_t
p_rlen	Where to write the number of bytes read.	uint32_t *

Return Values

Return value	Description
USBD_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
USBH_RAW_BUSY	Operation is still ongoing.
USBH_RAW_ERROR	Operation failed.

usbh_raw_write

Use this function to start a write transfer (from host to device).

> **Note:** This function starts the transfer but does not wait for it to end. To check for completion, call **usbh_raw_write_state()**.

Format

```
int usbh_raw_write (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t *        pc_buf,
    uint32_t         blen,
    uint8_t          skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
pc_buf	A pointer to the buffer. This buffer preserves data until the end of the transfer.	uint8_t *
blen	The length of the buffer.	uint32_t
skip_zlp	If the requested transmit length is a multiple of the USB packet size then: • If this is not set the transfer will be terminated with an additional zero length packet after the data. • If it is set then only the requested data is sent.	uint8_t

Return Values

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

usbh_raw_write_ctrl

Use this function to start a write transfer (from host to device) on the Control channel.

> **Note:** This function starts the transfer but does not wait for it to end. To check for completion, call **usbh_raw_write_state()**.

Format

```
int usbh_raw_write_ctrl (
    t_usbh_unit_id            uid,
    t_usbh_raw_setup_data *   p_setup_data )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_setup_data	A pointer to the buffer that holds the SETUP data and buffer.	t_usbh_raw_setup_data *

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
USBH_RAW_ERROR	Operation failed.

usbh_raw_write_ep

Use this function to start a write transfer (from host to device) by endpoint number instead of the channel.

> **Note:** This function starts the transfer but does not wait for it to end. To check for completion, call **usbh_raw_write_state_ep()**.

Format

```
int usbh_raw_write_ep (
    t_usbh_unit_id   uid,
    uint8_t          ep,
    uint8_t *        pc_buf,
    uint32_t         blen,
    uint8_t          skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ep	The endpoint number. The range is 0x01 .. 0x04.	uint8_t
pc_buf	A pointer to the buffer holding the data. This buffer preserves data until the transfer ends.	uint8_t *
blen	The number of bytes to write.	uint32_t
skip_zlp	If the requested transmit length is a multiple of the USB packet size then: • If this is not set the transfer will be terminated with an additional zero length packet after the data. • If it is set then only the requested data is sent.	uint8_t

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
Else	See Error Codes.

usbh_raw_write_state

Use this function to check the status of the ongoing write operation.

Format

```
int usbh_raw_write_state (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t          b_block )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
b_block	Set this to 1 if this call needs to block (wait until the write finishes). Ignore this in non-OS mode.	uint8_t

Return Values

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

usbh_raw_write_state_ep

Use this function to check the status of the ongoing **usbh_raw_write_ep()** operation.

Format

```
int usbh_raw_write_state_ep (
    t_usbh_unit_id   uid,
    uint8_t          ep,
    uint8_t          b_block )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_idt
ep	The endpoint number. The range is 0x01 .. 0x04.	uint8_t
b_block	Set this to 1 if this call needs to block (wait until the write finishes). Ignore this in non-OS mode.	uint8_t

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_BUSY	Operation is still ongoing.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
Else	See Error Codes.

usbh_raw_write_int

Use this function to start a write transfer on the Interrupt channel.

> **Note:** To check for completion, call **usbh_raw_write_int_state( )** .

Format

```
int usbd_raw_write_int (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t *        pc_buf,
    uint32_t         blen,
    uint8_t          b_skip_zlp )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
pc_buf	A pointer to the buffer with the data. This buffer preserves the data until the end of the transfer.	uint8_t *
blen	The number of bytes to write.	uint32_t
b_skip_zlp	Set this to 1 to skip zero length packets.	uint8_t

Return Values

Return value	Description
USBD_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
USBH_RAW_ERROR	Operation failed.

usbh_raw_write_int_state

Use this function to check the state of a previous write on the Interrupt channel.

Format

```
int usbh_raw_write_int_state (
    t_usbh_unit_id   uid
    uint8_t                 channel,
    uint8_t                 b_block )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number (the interface). The range is 0 to 3.	uint8_t
b_block	Set this non-zero to wait until the transfer finishes. Ignore this in non-OS mode.	uint8_t

Return Values

Return value	Description
USBH_RAW_SUCCESS	Successful execution.
USBH_RAW_ERR_NOT_PRESENT	No active device is connected.
USBH_RAW_BUSY	Operation is still ongoing.
USBH_RAW_ERROR	Operation failed.

usbh_raw_get_ep_number

Use this function to get a channel's endpoint number.

Format

```
int usbh_raw_get_ep_number (
    t_usbh_unit_id   uid,
    uint8_t          channel,
    uint8_t          direction,
    uint8_t *        ep )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
channel	The channel number. The range is 0 .. 3.	uint8_t
direction	USBH_DIR_IN or USBH_DIR_OUT.	uint8_t
ep	The endpoint number.	uint8_t *

Return Values

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

usbh_raw_present

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

Format

```
int usbh_raw_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 compatible device is connected.
Non-zero.	At least one compatible device is connected.

usbh_raw_get_port_hdl

Use this function to get a port handle.

Format

```
t_usbh_port_hdl usbh_raw_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
USBH_RAW_SUCCESS	Successful execution.
USBH_PORT_HDL_INVALID	Invalid port handle.

usbh_raw_register_ntf

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

When a device is connected or disconnected, or one of the specific events for this type of device 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

```
usbh_raw_register_ntf (
    t_usbh_unit_id   uid,
    t_usbh_ntf       ntf,
    t_usbh_ntf_fn    p_ntf_fn )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification type.	t_usbh_ntf
p_ntf_fn	The notification function.	t_usbh_ntf_fn

Return Values

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

4.3 Error Codes

The first table below shows the meaning of the error codes specific to this raw class driver, the second table shows the generic USB host error codes.

Raw-specific Error Codes

Return Code	Value	Description
USBH_RAW_SUCCESS	0	Operation ended with no error.
USBH_RAW_BUSY	1	Operation is still ongoing.
USBH_RAW_NOT_PRESENT	2	No active device is connected.
USBH_RAW_NOT_STARTED	3	Class driver is not started.
USBH_RAW_ERROR	4	Operation failed.

Generic USB Host 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

This section describes the *t_usbh_ntf_fn* and the notification codes that are defined in the USB host base system in the file **api_usb_host.h**.

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_RAW_RX (ch)	(USBH_NTF_CD_BASE + ch)	RX notification.
USBH_NTF_RAW_TX (ch)	(USBH_NTF_CD_BASE + USBH_RAW_MAX_CHANNELS + ch)	TX notification.
USBH_NTF_RAW_INT_RX(ch)	(USBH_NTF_CD_BASE + 2 * USBH_RAW_MAX_CHANNELS + ch)	RX INT notification.
USBH_NTF_RAW_INT_TX(ch)	(USBH_NTF_CD_BASE + 3 * USBH_RAW_MAX_CHANNELS + ch)	TX INT notification.

t_usbh_raw_setup_data

The *t_usbh_raw_setup_data* structure holds setup transaction data. Its elements are as follows:

Element	Type	Description
The SETUP packet's fields:		
bmRequestType	uint8_t	The request type bitmap specifying the direction of the request, the request type, and the designated recipient.
bRequest	uint8_t	The request type.
wValue	uint16_t	The 16 bit value for the request.
wIndex	uint16_t	The 16 bit index for the request.
wLength	uint16_t	The number of bytes to transfer if there is a data phase.
Others:		
p_buffer	uint8_t *	Pointer to buffer (used when *wLength* is not zero).
received_length	uint32_t	The number of bytes actually received (used when **usbh_raw_read_ctrl()** is called).

Request Type Bitmaps

The bitmaps in the Request type field of the setup packet (*bmRequestType*) specify the direction of the request,the request type, and the designated recipient. These are as follows:

Name	Value	Description
USBD_RAW_USBRQT_DIR_IN	(1u << 7)	IN request.
USBD_RAW_USBRQT_DIR_OUT	(0u << 7)	OUT request.
USBD_RAW_USBRQT_DIR_MASK	(1u << 7)	Mask for request.
USBD_RAW_USBRQT_TYP_STD	(0u << 5)	Standard request.
USBD_RAW_USBRQT_TYP_CLASS	(1u << 5)	Class-specific request.
USBD_RAW_USBRQT_TYP_VENDOR	(2u << 5)	Vendor-specific request.
USBD_RAW_USBRQT_TYP_MASK	(3u << 5)	Mask for request type bits.
USBD_RAW_USBRQT_RCP_DEVICE	(0u << 0)	Recipient is the device.
USBD_RAW_USBRQT_RCP_IFC	(1u << 0)	Recipient is an interface.
USBD_RAW_USBRQT_RCP_EP	(2u << 0)	Recipient is an endpoint.
USBD_RAW_USBRQT_RCP_OTHER	(3u << 0)	Recipient is something else.
USBD_RAW_USBRQT_RCP_MASK	(3u << 0)	Mask for recipient bits.

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.

The module uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	4

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 module makes use of the following standard PSP function:

Function	Package	Element	Description
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_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as little-endian to a memory location.

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

    /* Initialize the Raw class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_raw_init();
    }

    /* Start the Raw class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_raw_start();
    }

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

    return rc;

} /* usb_host_init */
```