USB Device Raw Class Driver User Guide

Version 1.40

For use with USBD Raw Class Driver versions 3.01 and above

Exported on 06/20/2018

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.

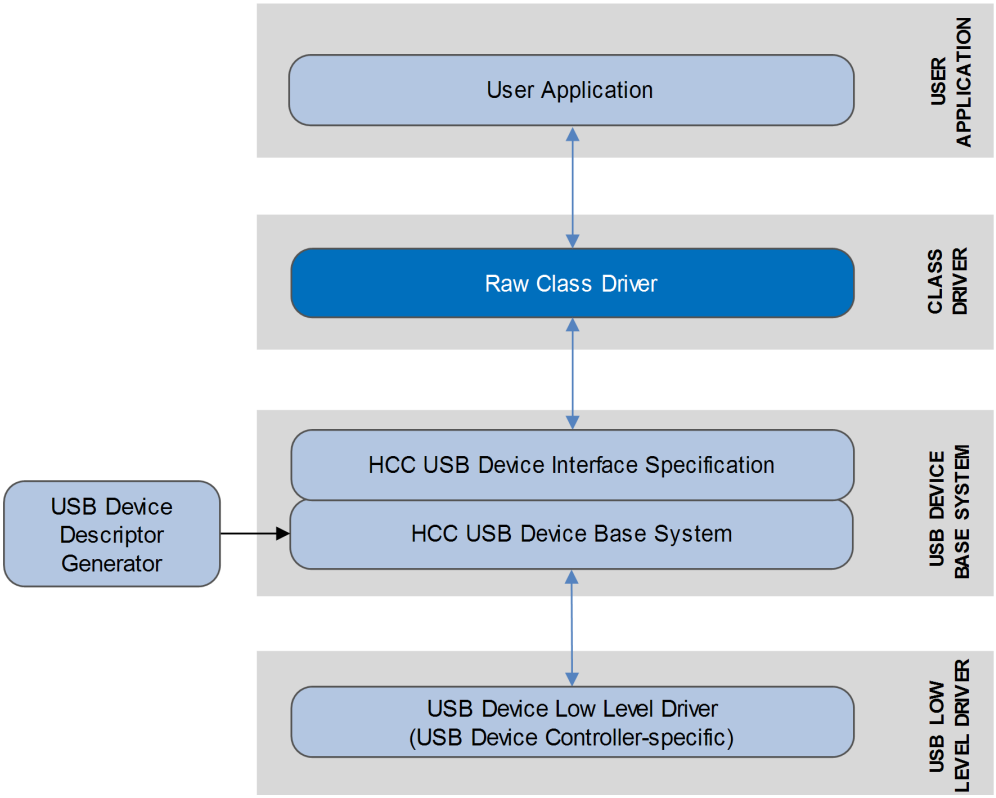
1.1 Introduction

This guide is for those who want to implement an Embedded USB Raw class driver, also known as a vendor-specific class driver. This class driver supports Bulk and Interrupt, IN and OUT transfers.

Bulk 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. Bulk may be used to support devices like printers and scanners. Data delivery is guaranteed by means of CRC error detection and limited retries. Bulk transfers are only supported by full and high speed devices.

Interrupt transfers are used to transfer data that needs to be delivered promptly. These can transfer fixed amounts of data in each USB frame.

The system structure is shown in the diagram below:



The package provides a set of Application Programming Interface (API) functions. The API description in this manual has separate sections describing the management interface and channel management functions.

Note: 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.

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.

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 development of vendor-specific class drivers.
- Supports configurable use of Bulk In and OUT endpoints.
- Supports configurable use of Interrupt IN and OUT endpoints.
- Zero Length Packet (ZLP) handling in TX/RX can be enabled/disabled.
- Compatible with sample device files produced by using the *HCC USB Device Descriptor Generator*.

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_raw	This Raw or vendor-specific class driver.

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 Raw 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 manuals, see USB Device PDFs.
- For the history of changes made to the package code itself, see History: usbd_cd_raw.

The current version of this manual is 1.40. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.40	2018-06-20	3.01	Package name changed from bulk to raw. Most other references to "bulk" replaced by "raw". Added *Types and Definitions* section. PSP Files added to *Source Files* section and *PSP Porting* section added.
1.30	2017-06-16	1.07	New *Change History* format.
1.20	2016-04-21	1.07	Added function group descriptions to API.
1.10	2015-03-27	1.07	Added *Change History*.
1.00	2014-08-27	1.04	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 file.

2.1 API Header File

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

2.3 System File

The file **src/usb-device/class-drivers/raw/usbd_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_usbd_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.

2.5 Platform Support Package (PSP) Files

These files in the directory **src/psp/board/demo** provide functions and elements the core code needs to use, depending on the hardware. Modify these files as required for your hardware.

Note:

- These are PSP implementations for the specific microcontroller and board; you may need to modify these to work with a different microcontroller and/or development board. See PSP Porting for details.
- In the package these files are offset to avoid overwriting an existing implementation. Copy them to the root **hcc** directory for use.

File	Description
raw_demo.c	Demo source code.
raw_demo.h	Demo header file.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbd_raw.h**. This section lists the available options and their default values.

> **Note:** The values of the first three options must match those in the device configuration descriptor. The USB device base package includes sample descriptor files for this class driver to use. You can use the HCC USB Device Descriptor Generator provided to modify these.

USBD_RAW_CLASS

The class that the Raw class driver uses. The default is 0xFE.

USBD_RAW_SUBCLASS

The subclass that the Raw class driver uses. The default is 0x01.

USBD_RAW_PROTOCOL

The protocol that the Raw class driver uses. The default is 0x07.

USBD_RAW_USE_INT_IN

Keep the default of 1 if the Interrupt IN channel is used. Otherwise, set this to 0.

USBD_RAW_USE_INT_OUT

Keep the default of 1 if the Interrupt OUT channel is used. Otherwise, set this to 0.

USBD_RAW_ONE_INTERFACE

Keep the default of 1 if all channels are on the same interface. Otherwise, set this to 0.

USBD_RAW_NUM_CH

The number of channels. The default is 1. This value must match that in the device descriptor.

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_raw_init()	Initializes the module and allocates the required resources.
usbd_raw_start()	Starts the module.
usbd_raw_stop()	Stops the module.
usbd_raw_delete()	Deletes the module and releases the resources it used.

usbd_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 usbd_raw_init ( void )
```

Arguments

Parameter
None.

Return Values

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

usbd_raw_start

Use this function to start the class driver.

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

Format

```
int usbd_raw_start ( void )
```

Arguments

Parameter
None.

Return Values

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

usbd_raw_stop

Use this function to stop the class driver.

Format

```
int usbd_raw_stop ( void )
```

Arguments

Parameter
None.

Return Values

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

usbd_raw_delete

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

Format

```
int usbd_raw_delete ( void )
```

Arguments

Parameter
None.

Return Values

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

4.2 Channel Management

The functions are the following:

Function	Description
usbd_raw_present()	Checks whether the class driver is active.
usbd_raw_read()	Starts reading data on the raw endpoint.
usbd_raw_read_state()	Checks the state of a previous read.
usbd_raw_read_int()	Reads data from the host on the Interrupt OUT endpoint.
usbd_raw_read_int_state()	Checks the state of a previous read on the Interrupt OUT endpoint.
usbd_raw_write()	Starts writing on the raw endpoint.
usbd_raw_write_state()	Checks the state of a previous write.
usbd_raw_write_int_state()	Sends data to the host on the Interrupt channel.
usbd_raw_write_int_state()	Checks the state of a previous write on the Interrupt channel.
usbd_raw_register_ep0_callback()	Registers an optional callback function that can be called when a transfer finishes.

usbd_raw_present

Use this function to check whether the class driver is active.

Format

```
int usbd_raw_present ( uint8_t ch )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t

Return Values

Return value	Description
USBD_RAW_SUCCESS	The class driver is active.
USBD_RAW_NOT_PRESENT	The class driver is not active.

usbd_raw_read

Use this function to start reading data on the raw endpoint.

Note:

- This function starts the transfer but does not wait for it to end. To check for completion, call **usbd_raw_read_state()**.
- If the buffer length is not an exact multiple of the endpoint size, the host must **never** send more data than the buffer size. For example, if the endpoint size is 64 and a read starts with a buffer size of 10, the host can only send 10 bytes at a time. Otherwise the system may corrupt memory by writing outside the buffer.

Format

```
int usbd_raw_read (
    uint8_t     ch,
    uint8_t *   p_buf,
    uint32_t    blen,
    uint8_t     b_skip_zlp )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
p_buf	A pointer to the buffer.	uint8_t *
blen	The length of the buffer.	uint32
b_skip_zlp	Set this to 1 to skip zero length packets.	uint8_t

Return Values

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

usbd_raw_read_state

Use this function to check the state of a previous read.

> **Note:** If the buffer length is greater than the maximum endpoint packet size then, if the host sends an exact multiple of endpoint packet size, **usbd_raw_read_state()** reports USBD_RAW_BUSY, as long as the whole buffer is not full, or a short packet is not sent by the host.

Format

```
int usbd_raw_read_state (
    uint8_t      ch,
    uint8_t      b_block,
    uint32_t *   p_rlen )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
b_block	1 if the call needs to block. 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.
Else	See Error Codes.

usbd_raw_read_int

Use this function to read data from the host on the Interrupt OUT endpoint.

> **Note:** This function is non-blocking. To check the completion state, call **usbd_raw_read_int_state()**.

Format

```
int usbd_raw_read_int (
    uint8_t     ch,
    uint8_t *   p_buf,
    uint32_t    blen,
    uint8_t     b_skip_zlp )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
p_buf	A pointer to the 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.
Else	See Error Codes.

usbd_raw_read_int_state

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

> **Note:** The contents of the buffer are not copied, so do not change these as long as **usbd_raw_read_int_state()** returns USBD_RAW_BUSY.

Format

```
int usbd_raw_read_int_state (
    uint8_t      ch,
    uint8_t      b_block,
    uint32_t *   p_rlen )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
b_block	1 if the call needs to block. 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.
Else	See Error Codes.

usbd_raw_write

Use this function to start writing on the raw endpoint.

Note:

- This function starts the transfer but does not wait for it to end. To check for completion, call **usbd_raw_write_state()**.
- The contents of the buffer are not copied, so do not change these as long as **usbd_raw_write_state()** returns USBD_RAW_BUSY.

Format

```
int usbd_raw_write (
    uint8_t     ch,
    uint8_t *   p_buf,
    uint32_t    blen,
    uint8_t     b_skip_zlp )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
p_buf	A pointer to the 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.
Else	See Error Codes.

usbd_raw_write_state

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

Format

```
int usbd_raw_write_state (
    uint8_t        ch,
    uint8_t        b_block )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
b_block	1 if the call needs to block. Ignore this in non-OS mode.	uint8_t

Return Values

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

usbd_raw_write_int

Use this function to send data to the host on the Interrupt channel.

> **Note:** The contents of the buffer are not copied, so do not change these as long as **usbd_raw_write_int_state()** returns USBD_RAW_BUSY.

Format

```
int usbd_raw_write_int (
    uint8_t     ch,
    uint8_t *   p_buf,
    uint32_t    blen,
    uint8_t     b_skip_zlp )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
p_buf	A pointer to the 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.
Else	See Error Codes.

usbd_raw_write_int_state

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

Format

```
int usbd_raw_write_int_state (
    uint8_t        ch,
    uint8_t        b_block )
```

Arguments

Parameter	Description	Type
ch	The channel number (the interface).	uint8_t
b_block	1 if the call needs to block. Ignore this in non-OS mode.	uint8_t

Return Values

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

usbd_raw_register_ep0_callback

Use this function to register an optional callback function to be called when a transfer finishes.

Note:

- The callback only applies when USBD_RAW_SETUP_ANSWER_DATA_IN or USBD_RAW_SETUP_ANSWER_DATA_OUT is used (there is data transfer).
- It is the user's responsibility to provide the callback function. Providing this function is optional.

Format

```
void usbd_raw_register_ep0_callback ( t_usbd_raw_ep0_cb p_callback )
```

Arguments

Parameter	Description	Type
p_callback	A pointer to the name of the callback.	t_usbd_raw_ep0_cb

Return Values

None.

4.3 Error Codes

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

Return Value	Value	Description
USBD_RAW_SUCCESS	0	Successful execution
USBD_RAW_BUSY	1	Operation is busy (transfer has not finished yet).
USBD_RAW_NOT_PRESENT	2	Driver not present (not able to perform any read/write request).
USBD_RAW_NOT_STARTED	3	Read/write state was requested without starting the read or write.
USBD_RAW_ERROR	4	General error.

4.4 Types and Definitions

This section describes the main elements that are defined in the API Header file.

t_usbd_raw_ep0_event

The *t_usbd_raw_ep0_event* typedef values are as follows:

Name	Description
USBD_RAW_EP0_EVENT_SETUP_RECEIVED = 1	Passed to the callback that was registered using **usbd_raw_register_ep0_callback()**.
USBD_RAW_EP0_EVENT_TRANSFER_FINISHED_SUCCESS	Passed to the **transfer_finished_callback()** function when transfer finished successfully.
USBD_RAW_EP0_EVENT_TRANSFER_FINISHED_ERROR	Passed to the **transfer_finished_callback()** function when an error occurs during transfer.

t_usbd_raw_setup_answer

The *t_usbd_raw_setup_answer* typedef values are as follows:

Name	Description
USBD_RAW_SETUP_ANSWER_STALL = 1	Request not recognized/respond stall.
USBD_RAW_SETUP_ANSWER_ACK	Acknowledge (no data transfer).
USBD_RAW_SETUP_ANSWER_DATA_IN	Write from the device's perspective (data transfer from device to host).
USBD_RAW_SETUP_ANSWER_DATA_OUT	Read from the device's perspective (data transfer from host to device).

t_usbd_raw_setup_data_tag

The *t_usbd_raw_setup_data_tag* structure holds setup transaction data. This is used in **t_usbd_raw_ep0_cb()** callback functions.

Its elements are as follows:

Element	Type	Description
ep0_event	t_usbd_raw_ep0_event	Inputs to the callback function.
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.
Outputs from the callback function:		
answer	t_usbd_raw_setup_answer	
p_buffer	uint8_t *	Pointer to buffer, used when USBD_RAW_SETUP_ANSWER_IN/OUT answered.
buffer_length	uint32_t	Size of buffer in bytes.
user_data	uint32_t	Optional, can be used by transfer_finished callback function.

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 a 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	3

5.2 PSP Porting

These functions are provided by the PSP to run the demo. They are designed for a specific microcontroller and development board. You may need to port them to work with your hardware solution; they are designed to make porting easy.

The package includes samples in the **raw_demo.c** file.

Function	Description
raw_demo_init()	Initializes the demo.
raw_demo_start()	Starts the demo.

These functions are described in the following sections.

raw_demo_init

This function is provided by the PSP to initialize the demo.

Format

```
int raw_demo_init ( void )
```

Arguments

None.

Return Values

Return value	Description
RAWDEMO_SUCCESS	Successful execution.
RAWDEMO_FAILURE	Operation failed.

raw_demo_start

This function is provided by the PSP to start the demo.

Format

```
int raw_demo_start ( void )
```

Arguments

None.

Return Values

Return value	Description
RAWDEMO_SUCCESS	Successful execution.
RAWDEMO_FAILURE	Operation failed.