

---

# USB Stack Host Reference Manual

Document Number: USBSHRM

Rev. 0, 12/2014

# Contents

<b>Chapter 1</b>	<b>Before You Begin.....</b>	<b>3</b>
1.1	About this book.....	3
1.2	Acronyms and abbreviations .....	3
1.3	Function listing format.....	3
<b>Chapter 2</b>	<b>Overview.....</b>	<b>5</b>
2.1	USB overview .....	5
2.2	Using the USB host API .....	6
2.3	API overview .....	8
<b>Chapter 3</b>	<b>USB Host Controller driver APIs .....</b>	<b>15</b>
3.1	USB host API.....	15
<b>Chapter 4</b>	<b>USB Host Class APIs .....</b>	<b>29</b>
4.1	HID class driver APIs .....	29
4.2	MSD class driver APIs .....	33
4.3	CDC class driver APIs .....	40
4.4	PHDC class driver APIs.....	49
4.5	Audio class driver.....	53
<b>Chapter 5</b>	<b>USB Host Configuration.....</b>	<b>62</b>
5.1	Common configure .....	62
5.2	KHCI configure .....	64
5.3	EHCI configure .....	65
<b>Chapter 6</b>	<b>Data Structures.....</b>	<b>67</b>
6.1	USB host controller driver structures.....	67
6.2	HID class structures .....	74
6.3	MSD class structures .....	76
6.4	CDC class structures .....	78
6.5	PHDC class structures .....	81
6.6	AUDIO class structures.....	83
<b>Chapter 7</b>	<b>OS Adapter.....</b>	<b>87</b>
7.1	OS adapter overview .....	87
7.2	API overview .....	87

# Chapter 1 Before You Begin

## 1.1 About this book

This *USB Stack Host Reference Manual* describes the USB Host driver and the programming interface in the USB Stack.

The audience should be familiar with the following reference material:

- *Universal Serial Bus Specification Revision 1.1*
- *Universal Serial Bus Specification Revision 2.0*

Use this book in addition to:

- *Source Code*

## 1.2 Acronyms and abbreviations

**Table 1 Acronyms and abbreviations**

Term	Description
API	Application Programming Interface
CDC	Communication Device Class
HID	Human Interface Device
MSD	Mass Storage Device
PHDC	Personal Healthcare Device Class

## 1.3 Function listing format

This is the general format of an entry for a function, compiler intrinsic, or a macro.

**function\_name()**

A short description of what function **function\_name()** does.

### Synopsis

Provides a prototype for function **function\_name()**.

```
<return_type> function_name(  
<type_1> parameter_1,  
<type_2> parameter_2,  
...  
<type_n> parameter_n)
```

## Parameters

parameter\_1 [in] – Pointer to x

parameter\_2 [out] – Handle for y

parameter\_n [in/out] – Pointer to z

Parameter passing is categorized as follows:

- *In* – indicates that the function uses one or more values in the parameter you give it without storing any changes.
- *Out* – indicates that the function saves one or more values in the parameter you give it. You can examine the saved values to find out useful information about your application.
- *In/out* – indicates the function changes one or more values in the parameter you give it and saves the result. You can examine the saved values to find out useful information about your application.

**Description** – Describes the function **function\_name()**. This section also describes any special characteristics or restrictions that might apply:

- Function blocks or might block under certain conditions
- Function must be started as a task
- Function creates a task
- Function has pre-conditions that might not be obvious
- Function has restrictions or special behavior

**Return value** – Specifies any value or values returned by function **function\_name()**.

**See also** – Lists other functions or data types related to function **function\_name()**.

**Example** – Provides an example (or a reference to an example) that illustrates the use of function **function\_name()**.

## Chapter 2 Overview

### 2.1 USB overview

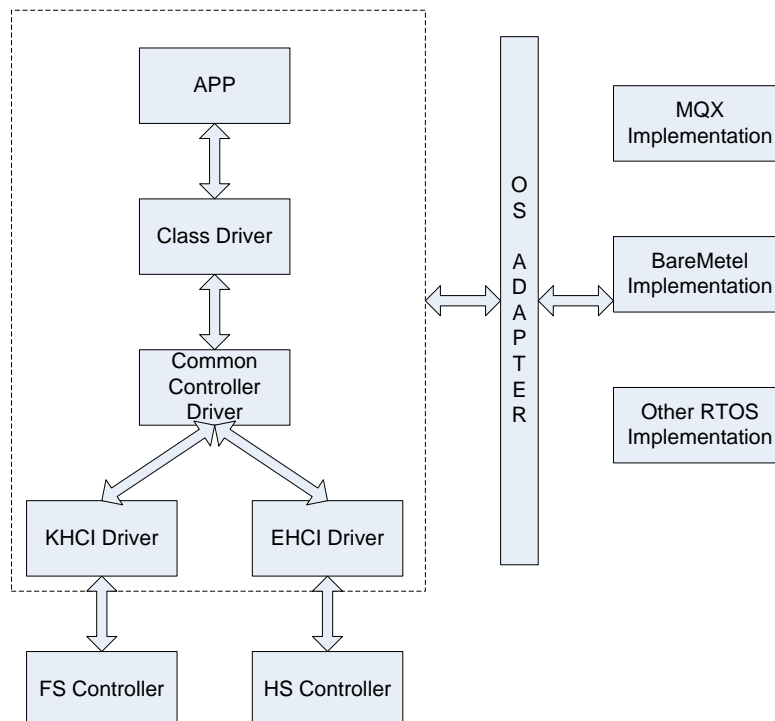
Universal Serial Bus (USB) is a polled bus. USB Host configures devices attached to it, either directly or through a USB hub, and initiates all bus transactions. USB Device responds only to the requests sent to it by a USB Host.

Because the USB Host manages the attachment and detachment of peripherals along with their power requirements dynamically, all the hardware implementation details can be hidden from applications. The USB Host determines which device driver to load for the connected device, and assigns a unique address to the device for run-time data transfers. The USB Host also manages data transfers and bus bandwidth allocation.

The USB Host software consists of the following:

- USB Host application
- USB Host Class Driver (contains USB Host Class APIs)
- USB Host Common Controller Driver APIs (independent of hardware)
- USB Host controller interface (HCI) - low-level functions used to interact with the USB Host controller hardware
- OS adapter to provide unified OS API to USB Stack

The whole architecture and components of USB stack as follows:



**Figure 1 USB Host stack architecture**

## 2.2 Using the USB host API

To use the USB Host API, follow these general steps. Each API function is described in the subsequent chapters.

1. Initialize the USB Host controller interface (`usb_host_init()`).
2. Optionally register services for types of events (`usb_host_register_service()`).
3. Open the pipe for a connected device or devices (`_usb_host_open_pipe()`).
4. Send control packets to configure the device or devices (`usb_host_send_setup()`).
5. Send (`usb_host_send_data()`) and receive (`usb_host_recv_data()`) data on pipes.
6. If required, cancel a transfer on a pipe (`usb_host_cancel()`).
7. If applicable, unregister services for pipes or types of events (`usb_host_unregister_service()`) and close pipes for disconnected devices (`_usb_host_close_pipe()`).
8. Shut down the USB Host controller interface (`usb_host_deinit()`).

Alternatively:

1. Define a table of driver capabilities that the application uses (as follows):

Sample driver information table

---

```

static usb_host_driver_info_t DriverInfoTable[] =
{
    {
        {0x00,0x00}, /* Vendor ID 1
USB-IF */
        {0x00,0x00}, /* Product
per manufacturer */
        USB_CLASS_HID, /* Class code
        USB_SUBCLASS_HID_BOOT, /* Sub-Class code
        USB_PROTOCOL_HID_MOUSE, /* Protocol
        0, /* Reserv
    */
        usb_host_hid_mouse_event /* Application call ba
function */
    },
    /* USB hub */
    {
        {0x00,0x00}, /* Vendor ID 1
USB-IF */
        {0x00,0x00}, /* Product
per manufacturer */
        USB_CLASS_HUB, /* Class code
        USB_SUBCLASS_HUB_NONE, /* Sub-Class code
        USB_PROTOCOL_HUB_ALL, /* Protocol
        0, /* Reserv
    */
        usb_host_hub_device_event /* Application call ba
function */
    },
    {
        {0x00,0x00}, /* All-zero ent
terminates */
        {0x00,0x00}, /* driver in
list. */
        0,
        0,
        0,
        0,
        NULL
    }
};

```

---

2. Initialize the USB Host controller interface (usb\_host\_init()).
3. The application should then register the driver information table to the host stack by calling the usb\_host\_register\_driver\_info() host API function.

4. Optionally register services for types of events (usb\_host\_register\_service()).
5. Wait for the callback function, which is specified in the driver info table, to be called.
6. Check for the events in the callback function: one of USB\_ATTACH\_EVENT, USB\_DETACH\_EVENT, USB\_CONFIG\_EVENT or USB\_INTF\_OPENED\_EVENT.
  - USB\_ATTACH\_EVENT: A newly attached device was just enumerated and a default configuration was selected.
  - USB\_DETACH\_EVENT: The device was detached.
  - USB\_CONFIG\_EVENT: A new configuration was selected on the device.
  - USB\_INTF\_OPENED\_EVENT: A new interface was selected on the device.
7. If it is a USB\_CONFIG\_EVENT event, select an interface by calling the host API function usb\_host\_open\_dev\_interface() .
8. After the USB\_INTF\_OPENED\_EVENT event is notified in the callback function, issue class-specific commands by using the class API.
9. If the USB\_DETACH\_EVENT event is notified in the callback function, close an interface by calling the host API function usb\_host\_close\_dev\_interface().
10. Open the pipe for a connected device or devices (usb\_host\_open\_pipe() ).
11. Transfer data by using the host API functions usb\_host\_send\_data() and/or usb\_host\_rcv\_data().
12. If required, cancel a transfer on a pipe (usb\_host\_cancel()).
13. If applicable, unregister services for types of events (usb\_host\_unregister\_service()) and close pipes for disconnected devices ( usb\_host\_close\_pipe()).
14. Shut down the USB Host controller interface (usb\_host\_deinit()).

## 2.3 API overview

This section describes the USB Stack Host API functions. The interfaces between USB Common Controller driver and xHCI driver are not listed here. Note that all USB host APIs can't be invoked in interrupt service routine.

**Table 2 USB Host controller driver APIs**

No.	API function	Description
1	usb_host_init()	Initializes the USB host controller
2	usb_host_deinit()	Un-initializes the host controller
3	usb_host_register_driver_info()	Registers driver information



4	usb_host_register_unsupported_device_notify()	Registers the callback function for unsupported device
5	usb_host_open_dev_interface()	Opens the selected interface
6	usb_host_close_dev_interface()	Closes the selected interface
7	usb_host_open_pipe()	Opens a pipe
8	usb_host_close_pipe()	Closes a pipe
9	usb_host_get_tr()	Gets a valid TR
10	usb_host_release_tr()	Releases a TR
11	usb_host_send_data()	Sends data to a specified endpoint
12	usb_host_send_setup()	Sends the setup data to a specified endpoint
13	usb_host_rcv_data()	Receives data from a specified endpoint
14	usb_host_cancel()	Cancels the TR in a endpoint
15	usb_host_bus_control()	Controls the BUS status
16	usb_host_get_host_handle()	Gets the host handle of the device
17	usb_host_dev_mng_get_address()	Gets a device address
18	usb_host_dev_mng_get_hubno()	Gets the hub index to which the target device attached.
19	usb_host_dev_mng_get_portno()	Gets the hub port index to which the target device attached.
20	usb_host_dev_mng_get_speed()	Gets the target device speed
21	usb_host_dev_mng_get_level()	Gets the hub level of the device
22	usb_host_dev_mng_get_host()	Gets the host handle
23	usb_host_dev_mng_get_control_pipe()	Gets the control pipe of the device
24	usb_host_dev_remove()	Removes the device

The following table summarizes the HID class API functions.

**Table 3 HID class driver APIs**

<b>No.</b>	<b>API function</b>	<b>Description</b>
1	usb_class_hid_init()	Initializes the HID class
2	usb_class_hid_deinit()	Un-initializes the HID class driver
3	usb_class_hid_pre_deinit()	Pre un-initializes the HID class driver
4	usb_class_hid_get_idle()	Reads the idle rate of a particular hid device report
5	usb_class_hid_get_protocol()	Reads the active protocol
6	usb_class_hid_get_report ()	Gets a report from the HID device
7	usb_class_hid_set_idle()	Reads silently a particular report on interrupt in the pipe
8	usb_class_hid_set_protocol()	Switches between the boot protocol and the report protocol
9	usb_class_hid_set_report()	Sends a report to the HID device

The following table summarizes the MSD class API functions.

**Table 4 MSD class driver APIs**

<b>No.</b>	<b>API function</b>	<b>Description</b>
1	usb_class_mass_init()	Initializes the MSD class
2	usb_class_msss_deinit()	Un-initializes the MSD class driver
3	usb_class_mass_pre_deinit()	Pre un-initializes the MSD class driver
4	usb_class_mass_storage_device_command()	Sends an MSD command to the target device
5	usb_class_mass_storage_device_command_cancel()	Cancels the last MSD command
6	usb_class_mass_getmaxlun_bulk only()	Gets the number of the logical units on the device
7	usb_class_mass_getvidpid()	Gets the VID and PID of the device

8	usb_class_mass_reset_recovery_on_usb()	Sends the RESET RECOVERY command to the command
9	usb_class_mass_q_init()	Initializes the command queue
10	usb_class_mass_q_insert()	Inserts a command to the queue
11	usb_class_mass_deleteq()	Deletes a command from the queue
12	usb_class_mass_cancelq()	Cancels the current command
13	usb_class_mass_get_pending_request()	Gets the current command
14	usb_mass_ufi_generic()	Initializes a UFI command
15	usb_mass_ufi_cancel()	Cancels a UFI command

The following table summarizes the CDC class API functions.

**Table 5 CDC class driver APIs**

No.	API function	Description
1	usb_class_cdc_acm_init()	Initializes the CDC class driver for Abstract Class Control
2	usb_class_cdc_acm_deinit()	Un-initializes the CDC class driver for Abstract Class Control
3	usb_class_cdc_acm_pre_deinit()	Pre un-initializes the CDC class driver for Abstract Class Control
4	usb_class_cdc_acm_use_lwevent()	Adds an LW event for Abstract Class Control
5	usb_class_cdc_data_init()	Initializes the class driver for Abstract Class Data
6	usb_class_cdc_data_deinit()	Un-initializes the class driver for Abstract Class Data
7	usb_class_cdc_data_pre_deinit()	Pre un-initializes the CDC class driver for Abstract Class Data
8	usb_class_cdc_data_use_lwevent()	Adds an LW event for Abstract Class Data

9	usb_class_cdc_get_ctrl_interface()	Gets the registered control interface
10	usb_class_cdc_get_data_interface()	Gets the registered data interface
11	usb_class_cdc_get_acm_line_coding()	Gets parameters of the current line (baudrate, HW control)
12	usb_class_cdc_set_acm_line_coding()	Sets parameters of the current line (baudrate, HW control)
13	usb_class_cdc_get_acm_descriptors()	Searches for descriptors in the device configuration and fills back fields if the descriptor is found
14	usb_class_cdc_set_acm_descriptors()	Sets descriptors for the ACM interfaces
15	usb_class_cdc_get_ctrl_descriptor()	Gets descriptor of the control interface
16	usb_class_cdc_bind_data_interfaces()	Binds all data interfaces belonging to the ACM control instance
17	usb_class_cdc_unbind_data_interfaces()	Un-binds all data interfaces belonging to the ACM control instance
18	usb_class_cdc_bind_acm_interfaces()	Binds data interfaces belonging to the control interface
19	usb_class_cdc_unbind_acm_interfaces()	Un-binds data interfaces belonging to the control interface
20	usb_class_cdc_init_ipipe()	Starts interrupt endpoint to poll for interrupt on specified device
21	usb_class_cdc_intf_validate()	Determines whether the class interface is validated

The following table summarizes the PHDC class API functions.

**Table 6 PHDC class driver APIs**

No.	API Function	Description
1	usb_class_phdc_init()	Initializes the PHDC class

2	usb_class_phdc_deinit()	Un-initializes the PHDC class driver
3	usb_class_phdc_pre_deinit()	Pre un-initializes the PHDC class driver
4	usb_class_phdc_recv_data()	Receives data from either a bulk-in or an interrupt pipe
5	usb_class_phdc_send_control_request()	Sends a PHDC class specific request
6	usb_class_phdc_send_data()	Send data to a PHDC device through a bulk-out pipe.
7	usb_class_phdc_set_callbacks( )	Registers application callback pointers to the current PHDC interface

The following table summarizes the AUDIO class API functions.

**Table 7 AUDIO class driver APIs**

No.	API function	Description
1	usb_class_audio_control_init()	Initializes the audio control interface
2	usb_class_audio_control_deinit()	Un-initializes the audio control interface
3	usb_class_audio_control_pre_deinit()	Pre un-initializes the audio control interface
4	usb_class_audio_stream_init()	Initializes the audio data interface
5	usb_class_audio_stream_deinit()	Un-initializes the audio data interface
6	usb_class_audio_stream_pre_deinit()	Pre un-initializes the audio data interface
7	usb_class_audio_control_get_descriptors()	Searches for descriptors of the audio control interface
8	usb_class_audio_control_set_descriptors()	Sets descriptors for the audio control interface
9	usb_class_audio_stream_get_descriptors()	Searches for descriptors of the audio stream interface
10	usb_class_audio_stream_set_descriptors()	Sets descriptors for the audio stream interface

11	<code>usb_class_audio_cntrl_common()</code>	Sends data through the control interface
12	<code>usb_class_audio_cntrl_callback()</code>	Controls interface callback
13	<code>usb_class_audio_rcv_data()</code>	Receives data through the data interface
14	<code>usb_class_audio_rcv_callback()</code>	Receives callback through the data interface
15	<code>usb_class_audio_send_data()</code>	Sends data through the data interface
16	<code>usb_class_audio_send_callback()</code>	Sends callback through the data interface

## Chapter 3 USB Host Controller driver APIs

### 3.1 USB host API

There are several kinds of USB classes supported in the USB Host stack. Maybe not all of them need to be supported in one specific USB example. For example, a HID mouse example may only be interested in the HID class device and HUB. The APP must register the class driver information to the USB stack so that the USB stack knows which class device the APP wants to manage. An event handler callback must be provided to USB stack so that the device's attach/detach/configuration events can be notified to APP by the USB stack.

In addition, to let the APP know about the detailed information about an attached unsupported device, `usb_host_register_unsupported_device_notify` is provided so that the APP can get the detailed unsupported interface information.

#### 3.1.1 `usb_host_init`

##### Synopsis

```
usb_status usb_host_init
(
    uint8_t controller_id,
    usb_host_handle * handle
)
```

##### Parameters

<code>controller_id</code>	[in] – controller ID
	KHCI 0 --- 0
	KHCI 1 --- 1
	EHCI 0 --- 2
	EHCI 1 --- 3
<code>handle</code>	[out] – host handle, refer to <a href="#">6.1.7</a>

##### Description

The function calls an HCI function to initialize the USB Host hardware and install an ISR that services all interrupt sources on the USB Host hardware.

The function also allocates and initializes all internal host-specific data structures and USB Host internal data and returns a USB Host controller handle for subsequent use with other USB Host API functions.

##### Return Value

USB\_OK (success)

Others (failure)

### 3.1.2 usb\_host\_deinit

#### Synopsis

```
usb_status usb_host_deinit
(
    usb_host_handle * handle
)
```

#### Parameters

handle [in] – host handle

#### Description

The function calls an HCI function to stop the specified USB Host controller. Call the function when the services of the USB Host controller are no longer required, or if the USB Host controller needs to be re-configured.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.3 usb\_host\_register\_driver\_info

#### Synopsis

```
usb_status usb_host_register_driver_info
(
    usb_host_handle * handle,
    void* info_table_ptr
)
```

#### Parameters

handle [in] – USB host

info\_table\_ptr [in] – Device info table, refer to [6.1.5](#)

#### Description

This function is used by the application to register a driver for a device with a particular vendor ID, product ID, class, subclass, and protocol code.

#### Return Value

USB\_OK (success)

Others (failure)

#### Data structure

```
typedef struct driver_info
{
    uint8_t idVendor[2]; /* Vendor ID per USB-IF */
}
```



```

uint8_t      idProduct[2];      /* Product ID per manufacturer */
uint8_t      bDeviceClass;      /* Class code, 0xFF if any */
uint8_t      bDeviceSubClass;   /* Sub-Class code, 0xFF if any */
uint8_t      bDeviceProtocol;   /* Protocol, 0xFF if any */
uint8_t      reserved;          /* Alignment padding */
event_callback attach_call;
} USB_HOST_DRIVER_INFO, * USB_HOST_DRIVER_INFO_PTR;

```

**Note:** For the `attach_call` callback function, the following events are sent to APP by this callback function in current implementation:

- **USB\_ATTACH\_EVENT**  
A device attached and a valid interface which match the registered information by APP are provided
- **USB\_CONFIG\_EVENT**  
The device is configured and all the valid interfaces which match the registered information by APP are provided. The APP can select an interface from these interfaces as the one to be enabled.
- **USB\_INTF\_OPENED\_EVENT**  
The selected interface is enabled, and the APP can call the class driver interface to do anything.
- **USB\_DETACH\_EVENT**  
The attached device is detached.

In the future, the following event may be added:

- **USB\_REMOTE\_WAKEUP**  
The USB BUS is waked up by a remote device.
- **USB\_SUSPENDED**  
The USB BUS is suspended.
- **USB\_WAKEUP**  
The USB BUS is waked up from suspended state (not from remote device but from host side).

### 3.1.4 usb\_host\_register\_unsupported\_device\_notify

#### Synopsis

```

usb_status usb_host_register_unsupported_device_notify
(
    usb_host_handle *   handle,
    event_callback      unsupported_device_notify
)

```

#### Parameters

handle [in] – USB host

unsupported\_device\_notify [in] – callback function to get the unsupported device notification

### Description

This function is used by the application to register a callback function to get all the information of unsupported device.

### Return Value

USB\_OK (success)

Others (failure)

### Data structure

```
typedef void (_CODE_PTR_ event_callback)(usb_device_instance_handle,
usb_interface_descriptor_handle intf_handle, uint32_t event_code);
```

If event\_code is USB\_ATTACH\_INTF\_NOT\_SUPPORT, then intf\_handle is a pointer point to [USB\\_DEVICE\\_INTERFACE\\_STRUCT](#)

If event\_code is USB\_ATTACH\_DEVICE\_NOT\_SUPPORT, then both the device handle and interface handle are NULL.

## 3.1.5 usb\_host\_open\_dev\_interface

### Synopsis

```
usb_status usb_host_open_dev_interface
(
usb_host_handle handle,
usb_device_instance_handle dev_handle,
usb_interface_descriptor_handle intf_handle,
class_handle * class_handle_ptr
)
```

### Parameters

handle [in] – USB host handle

dev\_handle [in] – attached device handle

intf\_handle [in] – interface handle to be opened

class\_handle\_ptr [out] – class handle associated to the interface

### Description

This function is used by the application to open the selected interface, and the corresponding class driver handle will be obtained through the class\_handle\_ptr parameter that can be used for the following transfer.

### Return Value

USB\_OK (success)

Others (failure)

### 3.1.6 usb\_host\_close\_dev\_interface

#### Synopsis

```
usb_status usb_host_close_dev_interface
(
    usb_host_handle                handle,
    usb_device_instance_handle     dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle                   class_handle
)
```

#### Parameters

handle	[in] – USB host handle
dev_handle	[in] – attached device handle
intf_handle	[in] – interface handle to be closed
class_handle	[in] – class handle associated to the interface

#### Description

This function is used by the application to close the selected interface. For the detailed information for this API, see Section 4.4.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.7 usb\_host\_open\_pipe

#### Synopsis

```
usb_status usb_host_open_pipe
(
    usb_host_handle                handle,
    usb_pipe_handle *              pipe_handle_ptr,
    pipe_init_struct_t*            pipe_init_ptr
)
```

#### Parameters

handle	[in] – USB host handle
pipe_handle_ptr	[out] – returned pipe handle, refer to Section <a href="#">6.1.4</a>
pipe_init_ptr	[in] – parameter to initialize the pipe, refer to Section <a href="#">6.1.3</a>

#### Description

This function is used by the application to open a pipe. The pipe detailed information is included in the pipe\_init\_ptr, and it points to PIPE\_INIT\_STRUCT.

#### Return Value

USB\_OK (success)

Others (failure)

#### Data structure

```
typedef struct pipe_init_struct
{
    void*                dev_instance;
    uint32_t             flags;
    uint16_t             max_packet_size;
    uint16_t             nak_count;
    uint8_t              interval;
    uint8_t              endpoint_number;
    uint8_t              direction;
    uint8_t              pipetype;
}
```

### 3.1.8 pipe\_init\_struct\_t; pipe\_init\_struct\_t\*usb\_host\_close\_pipe

#### Synopsis

```
usb_status usb_host_open_pipe
(
    usb_host_handle      handle,
    usb_pipe_handle      pipe_handle_ptr,
)
```

#### Parameters

handle [in] – USB host handle

pipe\_handle [in] – pipe handle

#### Description

This function is used by the application to close an opened pipe so that the pipe resource can be free.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.9 usb\_host\_get\_tr

#### Synopsis

```
usb_status usb_host_get_tr
(
    usb_host_handle      handle,
    tr_callback          callback,
    void*                callback_param,
    tr_struct_t**        tr_ptr_ptr
)
```

## Parameters

handle	[in] – USB host handle
callback	[in] – callback function that will be called when TR completed
callback_param	[in] – callback parameter to the callback function
tr_ptr_ptr	[out] – return the pointer to TR, refer to Section <a href="#">6.1.2</a>

## Description

This function is used by the application to get a valid TR that will be used in the following transfer. Meanwhile, the TR callback and callback parameter are provided to be initialized.

## Return Value

USB\_OK (success)

Others (failure)

## Data structure

```
typedef struct tr_struct
{
    struct tr_struct*      next;
    uint32_t               status;
    uint32_t               tr_index;           /* Transfer number on this pipe */
    uint8_t *              tx_buffer;         /* Address of buffer containing the
data to be transmitted (including OUT data phase of control transfers) */
    uint8_t *              rx_buffer;         /* Address of buffer to receive data
*/
    uint32_t               tx_length;         /* Length of data to transmit. For
control transfers, the length of data for the data phase */
    uint32_t               rx_length;         /* Length of data to be received.
For control transfers, this indicates the length of data for the data phase */
    bool                   send_phase;        /* Second phase of setup packet: Send/Receive
*/
    usb_setup_t            setup_packet;      /* Setup packet raw data */
    uint32_t               transfered_length;
    tr_callback             callback;         /* The callback function to be invoked
when a transfer is completed or an error is to be reported */
    void*                  callback_param;    /* The second parameter to be passed
into the callback function when it is invoked */
    void*                  hw_transaction_head; /* used only for EHCI */
    void*                  hw_transaction_tail; /* used only for EHCI */
    uint8_t                occupied;
    uint8_t                setup_status;
    uint8_t                no_of_itds_sitds;
    uint8_t                setup_first_phase;
} tr_struct_t;
```

### 3.1.10 usb\_host\_release\_tr

#### Synopsis

```
usb_status usb_host_release_tr
(
    usb_host_handle          handle,
    tr_struct_t**            tr_ptr
)
```

#### Parameters

handle                                   [in] – USB host handle  
tr\_ptr                                   [in] – pointer to TR to be released

#### Description

This function is used by the application to release a TR so that the TR resource can be free in the USB stack.

#### Return Value

USB\_OK (success)  
Others (failure)

### 3.1.11 usb\_host\_send\_data

#### Synopsis

```
usb_status usb_host_send_data
(
    usb_host_handle          handle,
    usb_pipe_handle          pipe_handle,
    tr_struct_t**            tr_ptr
)
```

#### Parameters

handle                                   [in] – USB host handle  
pipe\_handle                           [in] – pipe handle  
tr\_ptr                                   [in] – pointer to TR

#### Description

This function is used by the application to send data through target pipe that is assigned by the pipe\_handle parameter. The detailed data about the address, the length, the transfer type is assigned in the TR structure.

#### Return Value

USB\_OK (success)  
Others (failure)

### 3.1.12 usb\_host\_send\_setup

#### Synopsis

```
usb_status usb_host_send_setup
(
    usb_host_handle          handle,
    usb_pipe_handle          pipe_handle,
    tr_struct_t**            tr_ptr
)
```

#### Parameters

handle	[in] – USB host handle
pipe_handle	[in] – pipe handle
tr_ptr	[in] – pointer to TR

#### Description

This function is used by the application to send a setup through target pipe, which is always the control pipe 0.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.13 usb\_host\_recv\_data

#### Synopsis

```
usb_status usb_host_recv_data
(
    usb_host_handle          handle,
    usb_pipe_handle          pipe_handle,
    tr_struct_t**            tr_ptr
)
```

#### Parameters

handle	[in] – USB host handle
pipe_handle	[in] – pipe handle
tr_ptr	[in] – pointer to TR

#### Description

This function is used by the application to receive data through the target pipe that is assigned by the pipe\_handle parameter. The detailed data about the address, the length, the transfer type is assigned in the TR structure.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.14 usb\_host\_cancel

#### Synopsis

```
usb_status usb_host_cancel
(
    usb_host_handle          handle,
    usb_pipe_handle          pipe_handle,
    tr_struct_t**            tr_ptr
)
```

#### Parameters

handle	[in] – USB host handle
pipe_handle	[in] – pipe handle
tr_ptr	[in] – pointer to TR

#### Description

This function is used by the application to cancel all the uncompleted TRs in a target pipe.

**Note:** There is no API provided by the stack to cancel a specific TR in a target pipe. The tr\_ptr parameter is not used in this API now, but we can extend this API in the future to cancel a specific TR, so we keep tr\_ptr parameter here.

#### Return Value

USB\_OK (success)

Others (failure)

### 3.1.15 usb\_host\_bus\_control

#### Synopsis

```
usb_status usb_host_bus_control
(
    usb_host_handle          handle,
    uint8_t                  bcontrol
)
```

#### Parameters

handle	[in] – USB host handle
bcontrol	[in] – control code of the BUS

#### Description

This function is used by the application to control the BUS status, for example, to suspend the BUS or resume the BUS. Currently this function is not implemented yet.

#### Return Value

USB\_OK (success)



Others (failure)

### 3.1.16 usb\_host\_get\_host\_handle

#### Synopsis

```
usb_status usb_host_get_host_handle
(
    usb_device_instance_handle dev_handle,
    usb_host_handle            * handle
);
```

#### Parameters

dev\_handle            [in] – attached device handle

handle                [out] – USB host handle

#### Description

This function is used by the application to get the host handle of the device.

Return Value

USB\_OK (success)

Others (failure)

### 3.1.17 usb\_host\_dev\_mng\_get\_address

#### Synopsis

```
uint8_t usb_host_dev_mng_get_address
(
    usb_device_instance_handle dev_handle
)
```

#### Parameters

dev\_handle            [in] – attached device handle

#### Description

This function is used by the application to get the USB address of the target attached device.

#### Return Value

USB address of the device (success)

0 (failure)

### 3.1.18 usb\_host\_dev\_mng\_get\_hubno

#### Synopsis

```
uint8_t  usb_host_dev_mng_get_hubno
(
usb_device_instance_handle  dev_handle
)
```

#### Parameters

dev\_handle [in] – attached device handle

#### Description

This function is used by the application to get the hub index to which the target device attached.

#### Return Value

Hub index of the device (success)

0xFF (failure)

### 3.1.19 usb\_host\_dev\_mng\_get\_portno

#### Synopsis

```
uint8_t  usb_host_dev_mng_get_portno
(
usb_device_instance_handle  dev_handle
)
```

#### Parameters

dev\_handle [in] – attached device handle

#### Description

This function is used by the application to get the hub port index to which the target device attached.

#### Return Value

Port number of the device (success)

0xFF (failure)

### 3.1.20 usb\_host\_dev\_mng\_get\_speed

#### Synopsis

```
uint8_t  usb_host_dev_mng_get_speed
(
usb_device_instance_handle  dev_handle
)
```

#### Parameters

dev\_handle [in] – attached device handle

#### Description

This function is used by the application to get the speed of the target device.

## Return Value

Speed of the device: 0 for full speed, 1 for low speed and 2 for high speed (success)

0xFF (failure)

### 3.1.21 usb\_host\_dev\_mng\_get\_level

#### Synopsis

```
uint8_t  usb_host_dev_mng_get_level
(
    usb_device_instance_handle    dev_handle
)
```

#### Parameters

dev\_handle [in] – attached device handle

#### Description

This function is used by the application to get hub level of the target device attached.

## Return Value

Hub level of the device (success)

0xFF (failure)

### 3.1.22 usb\_host\_dev\_mng\_get\_host

#### Synopsis

```
usb_host_handle  usb_host_dev_mng_get_host
(
    usb_device_instance_handle    dev_handle
)
```

#### Parameters

dev\_handle [in] – attached device handle

#### Description

This function is used by the application to get host handler of the target device.

## Return Value

Host handler of the device (success)

NULL (failure)

### 3.1.23 usb\_host\_dev\_mng\_get\_control\_pipe

#### Synopsis

```
usb_pipe_handle  usb_host_dev_mng_get_control_pipe
(
```

```
usb_device_instance_handle    dev_handle
)
```

### Parameters

dev\_handle [in] – attached device handle

### Description

This function is used by the application to get the control pipe handler of the target device.

### Return Value

Control pipe handler of the device (success)

NULL (failure)

## 3.1.24 usb\_host\_dev\_remove

### Synopsis

```
usb_status usb_host_dev_remove
(
usb_host_handle    handle,
usb_device_instance_handle    dev_handle
)
```

### Parameters

handle [in] – USB host handle

dev\_handle [in] – attached device handle

### Description

This function is used by the application to remove device. This function should not be used in normal situation, user need to be clear about the function when calling it.

### Return Value

USB\_OK (success)

Others (failure)

## Chapter 4 USB Host Class APIs

### 4.1 HID class driver APIs

#### 4.1.1 usb\_class\_hid\_init

##### Synopsis

```
usb_status usb_class_hid_init
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle*                 class_handle_ptr
)
```

##### Parameters

dev\_handle                      [in] – device handle  
intf\_handle                    [in] – interface handle  
class\_handle\_ptr               [out] – class driver's handle, refer to Section [6.2.1](#)

##### Description

This function is used to initialize the corresponding class driver, and it is not called by the APP directly. The class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by usb\_host\_open\_dev\_interface, the init function will be called automatically and the class driver's handle will be returned.

#### 4.1.2 usb\_class\_hid\_deinit

##### Synopsis

```
usb_status usb_class_hid_deinit
(
    class_handle                  handle
)
```

##### Parameters

handle                        [in] – class driver's handle

##### Description

This function is used to de-initialize a class driver handle.

This function is not called by the APP directly. It will be called when the USB host application call the usb\_host\_close\_dev\_interface() function.

#### 4.1.3 usb\_class\_hid\_pre\_deinit

##### Synopsis

```
usb_status usb_class_hid_pre_deinit
(
```

```

class_handle          handle
)

```

### Parameters

handle [in] – class driver's handle

### Description

This function is used to do some pre de-initialization operation, such as cancelling all the uncompleted transfers. This function is not called by the APP directly, and it will be included in a global class interface table so that it can be called automatically when the device detached.

## 4.1.4 usb\_class\_hid\_get\_idle

### Synopsis

```

usb_status usb_class_hid_get_idle
(
hid_command_t* com_ptr,
uint8_t          rid,
uint8_t *        idle_rate
)

```

### Parameters

com\_ptr [in] – class interface structure pointer, refer to Section [6.2.2](#)

rid [in] – report ID (see HID specification)

idle\_rate [out] – idle rate of this report

### Description

This function is called by the application to read the idle rate of a particular HID device report.

### Return Value

USB\_OK (success)

Others (failure)

### Data structure

```

typedef struct _hid_command
{
    class_handle    class_ptr;
    tr_callback     callback_fn;
    void*           callback_param;
}

```

## 4.1.5 } hid\_command\_t;usb\_class\_hid\_get\_protocol

### Synopsis

```

usb_status usb_class_hid_get_protocol
(
hid_command_t* com_ptr,

```

```
uint8_t *          protocol
)
```

### Parameters

com\_ptr                    [in] – class interface structure pointer  
 protocol                  [out] – protocol (1 byte, 0 = Boot Protocol or 1 = Report Protocol)

### Description

This function is called by the application to read the the active protocol.

### Return Value

USB\_OK (success)  
 Others (failure)

## 4.1.6 usb\_class\_hid\_get\_report

### Synopsis

```
usb_status usb_class_hid_get_report
(
  hid_command_t* com_ptr,
  uint8_t          rid,
  uint8_t          rtype,
  void*           buf,
  uint16_t         blen
)
```

### Parameters

com\_ptr                    [in] – class interface structure pointer  
 rid                        [in] – report ID (see HID specification)  
 rtype                    [in] – report type (see HID specification)  
 buf                       [in] – buffer to receive report data  
 blen                      [in] – length of the buffer

### Description

This function is called by the application to get a report from the HID device.

### Return Value

USB\_OK (success)  
 Others (failure)

## 4.1.7 usb\_class\_hid\_set\_idle

### Synopsis

```
usb_status usb_class_hid_set_idle
(
```

```

hid_command_t* com_ptr,
uint8_t          rid,
uint8_t *        duration
)

```

### Parameters

com\_ptr [in] – class interface structure pointer  
rid [in] – report ID (see HID specification)  
duration [in] – duration of the idle

### Description

This function is called by the application to set the idle rate of a particular HID device report.

### Return Value

USB\_OK (success)  
Others (failure)

## 4.1.8 usb\_class\_hid\_set\_protocol

### Synopsis

```

usb_status usb_class_hid_set_protocol
(
hid_command_t* com_ptr,
uint8_t          protocol
)

```

### Parameters

com\_ptr [in] – class interface structure pointer  
protocol [in] – protocol (1 byte, 0 = Boot Protocol or 1 = Report Protocol)

### Description

This function is called by the application to switches between the boot protocol and the report protocol.

### Return Value

USB\_OK (success)  
Others (failure)

## 4.1.9 usb\_class\_hid\_set\_report

### Synopsis

```

usb_status usb_class_hid_set_report
(
hid_command_t* com_ptr,
uint8_t          rid,
uint8_t          rtype,
void*            buf,
)

```



```
uint16_t          blen
)
```

### Parameters

com_ptr	[in] – class interface structure pointer
rid	[in] – report ID (see HID specification)
rtype	[in] – report type (see HID specification)
buf	[in] – buffer to send report data
blen	[in] – length of the buffer

### Description

This function is called by the application to send a report to the HID.

### Return Value

USB\_OK (success)

Others (failure)

## 4.2 MSD class driver APIs

### 4.2.1 usb\_class\_mass\_init

#### Synopsis

```
usb_status usb_class_mass_init
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle*                 class_handle_ptr
)
```

### Parameters

dev_handle	[in] – device handle
intf_handle	[in] – interface handle
class_handle_ptr	[out] – the class driver's handle, refer to Section <a href="#">6.3.1</a>

### Description

This function is used to initialize the corresponding class driver, and it is not called by the APP directly. The class driver's init/deinit functions are included in the global interface map table. When the corresponding interface is opened by usb\_host\_open\_dev\_interface, the initialization function will be called automatically and the class driver's handle will be returned.

### 4.2.2 usb\_class\_mass\_deinit

#### Synopsis

```
usb_status usb_class_mass_deinit
(
```

```

class_handle          handle
)

```

### Parameters

handle [in] – class driver's handle

### Description

This function is used to de-initialize a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the `usb_host_close_dev_interface()` function.

## 4.2.3 usb\_class\_mass\_pre\_deinit

### Synopsis

```

usb_status usb_class_mass_pre_deinit
(
class_handle          handle
)

```

### Parameters

handle [in] – class driver's handle

### Description

This function is used to do some pre de-initialization operation, such as cancelling all the uncompleted transfer. This function is not called by the APP directly, and it will be included in a global class interface table so that it can be called automatically when the device detached.

## 4.2.4 usb\_class\_mass\_storage\_device\_command

### Synopsis

```

usb_status usb_class_mass_storage_device_command
(
mass_command_struct_t*          cmd_ptr
)

```

### Parameters

cmd\_ptr [in] – command, refer to refer to Section [6.3.2](#)

### Description

This function is called by the protocol layer to execute the command defined in protocol API.

### Return Value

USB\_OK- Command has been successfully queued in class driver queue (or has been passed to USB, if there is no other command pending).

Others (failure)

## 4.2.5 usb\_class\_mass\_storage\_device\_command\_cancel

### Synopsis

```
bool usb_class_mass_storage_device_command_cancel
(
    mass_command_struct_t*          cmd_ptr
)
```

### Parameters

cmd\_ptr                    [in] – command

### Description

This function de-queues the command in the class driver queue.

### Return Value

USB\_OK- Command has been successfully de-queued in the class driver queue.

Others (failure)

## 4.2.6 usb\_class\_mass\_getmaxlun\_bulkonly

### Synopsis

```
usb_status usb_class_mass_getmaxlun_bulkonly
(
    class_handle          handle,
    uint8_t *             pLUN,
    tr_callback           callback,
    void*                 callback_param
)
```

### Parameters

handle                    [in] – class driver's handle

pLUN                      [in] – pointer to Logical Unit Number (LUN)

callback                  [in] – callback upon completion

callback\_param           [in] – callback parameter

### Description

This is a class specific command. This command is used to get the number of logical units on the device. Caller will use the LUN number to direct the commands (as a part of CBW).

### Return Value

USB\_OK- Command has been successfully queued in class driver queue (or has been passed to USB, if there is no other command pending).

Others (failure)

## 4.2.7 usb\_class\_mass\_getvidpid

### Synopsis

```
usb_status usb_class_mass_getvidpid
(
    class_handle          handle,
    uint16_t *            vid,
    uint16_t *            pid
)
```

### Parameters

handle                    [in] – the class driver's handle  
vid                        [out] – vendor ID  
pid                        [out] – Product ID

### Description

This function is used to get device's vid and pid.

### Return Value

USB\_OK - success.

Others (failure)

## 4.2.8 usb\_class\_mass\_reset\_recovery\_on\_usb

### Synopsis

```
usb_status usb_class_mass_reset_recovery_on_usb
(
    usb_mass_class_struct_t * mass_class)

```

### Parameters

mass\_class                [in] – the class driver's handle

### Description

This function gets the pending request from class driver queue and sends the RESET command on control pipe. This function is called when a phase of the pending command fails and class driver decides to reset the device. If there is no pending request in the queue, it will just return. This routine registers a call back for control pipe commands to ensure that pending command is queued again.

### Return Value

USB\_OK - success.

Others (failure)

## 4.2.9 usb\_class\_mass\_q\_init

### Synopsis

```
void usb_class_mass_q_init
(
    usb_mass_class_struct_t *    mass_class)
```

### Parameters

mass\_class                      [in] – the class driver's handle

### Description

This function initializes a mass storage class command queue.

## 4.2.10 usb\_class\_mass\_q\_insert

### Synopsis

```
int32_t usb_class_mass_q_insert
(
    usb_mass_class_struct_t *    mass_class,
    mass_command_struct_t*      pCmd
)
```

### Parameters

mass\_class                      [in] – the class driver's handle  
pCmd                              [in] - Command

### Description

This function is called by class driver to insert a command in the queue.

### Return Value

The index which inserted in queue.

## 4.2.11 usb\_class\_mass\_deleteq

### Synopsis

```
void usb_class_mass_deleteq
(
    usb_mass_class_struct_t *    mass_class,
)
```

### Parameters

mass\_class                      [in] – the class driver's handle

### Description

This function is called by class driver to delete a command from the queue.

#### 4.2.12 usb\_class\_mass\_cancelq

##### Synopsis

```
bool usb_class_mass_cancelq
(
    usb_mass_class_struct_t *          mass_class,
    mass_command_struct_t*             pCmd
)
```

##### Parameters

mass\_class            [in] – the class driver's handle  
pCmd                  [in] - Command

##### Description

This function is called by class driver to cancel a command in the queue.

##### Return Value

TRUE – Canceled.

FALSE – failed (May be not found the command)

#### 4.2.13 usb\_class\_mass\_get\_pending\_request

##### Synopsis

```
void usb_class_mass_get_pending_request
(
    usb_mass_class_struct_t *          mass_class,
    mass_command_struct_t* *           cmd_ptr_ptr
)
```

##### Parameters

mass\_class            [in] – the class driver's handle  
cmd\_ptr\_ptr           [out] - Command

##### Description

This function is called by class driver to get a command from the queue.

#### 4.2.14 usb\_mass\_ufi\_generic

##### Synopsis

```
usb_status usb_mass_ufi_generic
(
    mass_command_struct_t*             cmd_ptr,
```

```

uint8_t          opcode,
uint8_t          lun,
uint32_t         lbaddr,
uint32_t         blen,
uint8_t          cbwflags,
uint8_t *        buf,
uint32_t         buf_len
)

```

### Parameters

cmd_ptr	[in] – Command handle
opcode	[in] – Command code
lun	[in] – Logical unit number of command block
lbaddr	[in] – Logical block address
blen	[in] – Block length
cbwflags	[in] – Command block wrapper flags
buf	[in] – Transfer buffer address
buf_len	[in] – Transfer buffer length

### Description

This function is used to initialize a command transfer.

### Return Value

USB\_OK- Command has been successfully queued in class driver queue (or has been passed to USB, if there is no other command pending).

Others (failure)

## 4.2.15 usb\_mass\_ufi\_cancel

### Synopsis

```

bool usb_mass_ufi_cancel
(
    mass_command_struct_t*          cmd_ptr
)

```

### Parameters

cmd_ptr	[in] – Command
---------	----------------

### Description

This function is used to cancel a command in the queue.

### Return Value

TRUE – Canceled.

FALSE – failed (May be not found the command)

## 4.3 CDC class driver APIs

### 4.3.1 usb\_class\_cdc\_acm\_init

#### Synopsis

```
usb_status usb_class_cdc_acm_init
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle*                 class_handle_ptr
)
```

#### Parameters

dev\_handle                    [in] – device handle  
intf\_handle                  [in] – interface handle  
class\_handle\_ptr            [out] – the class driver's handle, refer to [6.4.1](#)

#### Description

This function is used to initialize the corresponding class driver, it is not called by the APP directly, normally, the class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by usb\_host\_open\_dev\_interface, the init function will be called automatically and the class driver's handle will be returned.

### 4.3.2 usb\_class\_cdc\_acm\_deinit

#### Synopsis

```
usb_status usb_class_cdc_acm_deinit
(
    class_handle                  handle
)
```

#### Parameters

handle                    [in] – the class driver's handle

#### Description

This function is used to deinit a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the usb\_host\_close\_dev\_interface() function.



### 4.3.3 usb\_class\_cdc\_acm\_pre\_deinit

#### Synopsis

```
usb_status usb_class_cdc_acm_pre_deinit
(
    class_handle          handle
)
```

#### Parameters

handle [in] – the class driver's handle

#### Description

This function is used to do some pre deinit operation like cancel all the uncompleted transfer. This function is not called by the APP directly, it will be included in a global class interface table so that it can be called automatically when the device detached.

### 4.3.4 usb\_class\_cdc\_acm\_use\_lwevent

#### Synopsis

```
usb_status usb_class_cdc_acm_use_lwevent
(
    cdc_class_call_struct_t *    ccs_ptr,
    os_event_handle              acm_event
)
```

#### Parameters

ccs\_ptr [in] – acm call struct pointer

acm\_event [in] – acm event

#### Description

This function is injector of event that is used in the class but the destruction is allowed only in task context..

#### Return Value

USB\_OK- event is successfully injected.

Others (failure)

### 4.3.5 usb\_class\_cdc\_data\_init

#### Synopsis

```
usb_status usb_class_cdc_data_init
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle*                  class_handle_ptr
)
```

#### Parameters

dev_handle	[in] – device handle
intf_handle	[in] – interface handle
class_handle_ptr	[out] – the class driver's handle, refer to <a href="#">6.4.2</a>

### Description

This function is used to initialize the corresponding class driver, it is not called by the APP directly, normally, the class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by `usb_host_open_dev_interface`, the init function will be called automatically and the class driver's handle will be returned.

## 4.3.6 usb\_class\_cdc\_data\_deinit

### Synopsis

```
usb_status usb_class_cdc_data_deinit
(
    class_handle          handle
)
```

### Parameters

handle	[in] – the class driver's handle
--------	----------------------------------

### Description

This function is used to deinit a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the `usb_host_close_dev_interface()` function.

## 4.3.7 usb\_class\_cdc\_data\_pre\_deinit

### Synopsis

```
usb_status usb_class_cdc_data_pre_deinit
(
    class_handle          handle
)
```

### Parameters

handle	[in] – the class driver's handle
--------	----------------------------------

### Description

This function is used to do some pre deinit operation like cancel all the uncompleted transfer. This function is not called by the APP directly, it will be included in a global class interface table so that it can be called automatically when the device detached.

## 4.3.8 usb\_class\_cdc\_data\_use\_lwevent

### Synopsis

```
usb_status usb_class_cdc_data_use_lwevent
(
```

```

cdc_class_call_struct_t *      ccs_ptr,
os_event_handle                data_event
)

```

### Parameters

ccs\_ptr                    [in] – data call struct pointer  
data\_event                [in] – data event

### Description

This function is injector of events that are used in the class but the destruction are allowed only in task context..

### Return Value

USB\_OK- events are successfully injected.  
Others (failure)

## 4.3.9 usb\_class\_cdc\_get\_ctrl\_interface

### Synopsis

```

cdc_class_call_struct_t * usb_class_cdc_get_ctrl_interface
(
    void *                intf_handle
)

```

### Parameters

intf\_handle               [in] – pointer to interface handle

### Description

This function is used to find registered control interface in the chain.

### Return Value

control interface instance

## 4.3.10 usb\_class\_cdc\_get\_data\_interface

### Synopsis

```

cdc_class_call_struct_t * usb_class_cdc_get_data_interface
(
    void *                intf_handle
)

```

### Parameters

intf\_handle               [in] – pointer to interface handle

### Description

This function is used to find registered data interface in the chain.

### Return Value

data interface instance

## 4.3.11 usb\_class\_cdc\_get\_acm\_line\_coding

### Synopsis

```
usb_status usb_class_cdc_get_acm_line_coding
(
    cdc_class_call_struct_t *      ccs_ptr,
    usb_cdc_uart_coding_t *      uart_coding_ptr
)
```

### Parameters

ccs\_ptr [in] – the communication device data instance structure

uart\_coding\_ptr [in] – Where to store coding

### Description

This function is used to get parameters of current line (baudrate,HW control...)

This function cannot be run in ISR

**Note:** DATA instance communication structure is passed here as parameter, not control interface.

### Return Value

USB\_OK - success.

Others (failure)

## 4.3.12 usb\_class\_cdc\_set\_acm\_line\_coding

### Synopsis

```
usb_status usb_class_cdc_set_acm_line_coding
(
    cdc_class_call_struct_t *      ccs_ptr,
    usb_cdc_uart_coding_t *      uart_coding_ptr
)
```

### Parameters

ccs\_ptr [in] – the communication device data instance structure

uart\_coding\_ptr [in] – Coding to set

### Description

This function is used to get parameters of current line (baudrate,HW control...)

This function cannot be run in ISR

NOTE!!!

DATA instance communication structure is passed here as parameter, not control interface.

#### **Return Value**

USB\_OK - success.

Others (failure)

### **4.3.13 usb\_class\_cdc\_get\_acm\_descriptors**

#### **Synopsis**

```
usb_status usb_class_cdc_get_acm_descriptors
(
    usb_device_instance_handle    dev_handle,

    usb_interface_descriptor_handle intf_handle,
    usb_cdc_desc_acm_t * *        acm_desc,
    usb_cdc_desc_cm_t * *         cm_desc,
    usb_cdc_desc_header_t * *     header_desc,
    usb_cdc_desc_union_t * *      union_desc
)
```

#### **Parameters**

dev_handle	[in] – pointer to device instance
intf_handle	[in] – pointer to interface descriptor
acm_desc	[in] – pointer to acm descriptor
cm_desc	[in] – pointer to cm descriptor
header_desc	[in] – pointer to header descriptor
union_desc	[in] – pointer to union descriptor

#### **Description**

This function is hunting for descriptors in the device configuration and fills back fields if the descriptor was found. Must be run in locked state and validated USB device.

#### **Return Value**

USB\_OK - success.

Others (failure)

### **4.3.14 usb\_class\_cdc\_set\_acm\_descriptors**

#### **Synopsis**

```

usb_status usb_class_cdc_set_acm_descriptors
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    usb_cdc_desc_acm_t * *        acm_desc,
    usb_cdc_desc_cm_t * *         cm_desc,
    usb_cdc_desc_header_t * *     header_desc,
    usb_cdc_desc_union_t * *      union_desc
)

```

### Parameters

dev_handle	[in] – pointer to device instance
intf_handle	[in] – pointer to interface descriptor
acm_desc	[in] – pointer to acm descriptor
cm_desc	[in] – pointer to cm descriptor
header_desc	[in] – pointer to header descriptor
union_desc	[in] – pointer to union descriptor

### Description

This func This function is used to set descriptors for ACM interface Descriptors can be used afterwards by application or by driver .

### Return Value

USB\_OK - success.

Others (failure)

## 4.3.15 usb\_class\_cdc\_get\_ctrl\_descriptor

### Synopsis

```

usb_status usb_class_cdc_get_ctrl_descriptor
(
    usb_device_instance_handle    dev_handle,

    usb_interface_descriptor_handle intf_handle,

    interface_descriptor_t* *     if_desc_ptr
)

```

### Parameters

dev_handle	[in] – pointer to device instance
intf_handle	[in] – pointer to interface
if_desc_ptr	[in] – pointer to interface descriptor

## Description

This function is hunting for descriptor of control interface, which controls data interface identified by descriptor (intf\_handle). The found control interface descriptor is written to if\_desc\_ptr.

## Return Value

USB\_OK - success.

Others (failure)

### 4.3.16 usb\_class\_cdc\_bind\_data\_interfaces

#### Synopsis

```
usb_status usb_class_cdc_bind_data_interfaces
(
    usb_device_instance_handle    dev_handle,

    cdc_class_call_struct_t *      ccs_ptr
)
```

#### Parameters

dev\_handle                      [in] – pointer to device instance

ccs\_ptr                         [in] – the communication device control instance structure

## Description

All data interfaces belonging to ACM control instance (specified by ccs\_ptr) will be bound to this interface. Union functional descriptor describes which data interfaces should be bound.

## Return Value

USB\_OK - success.

Others (failure)

### 4.3.17 usb\_class\_cdc\_unbind\_data\_interfaces

#### Synopsis

```
usb_status usb_class_cdc_unbind_data_interfaces
(
    cdc_class_call_struct_t *      ccs_ptr
)
```

#### Parameters

ccs\_ptr                         [in] – the communication device control instance structure

## Description

All data interfaces bound to ACM control instance will be unbound from this interface.

#### **Return Value**

USB\_OK - success.

Others (failure)

### **4.3.18 usb\_class\_cdc\_bind\_acm\_interface**

#### **Synopsis**

```
usb_status usb_class_cdc_bind_acm_interfaces
(
    usb_device_instance_handle    dev_handle,

    cdc_class_call_struct_t *      ccs_ptr
)
```

#### **Parameters**

dev\_handle                    [in] – pointer to device instance

ccs\_ptr                      [in] – the communication device control instance structure

#### **Description**

Data interface (specified by ccs\_ptr) will be bound to appropriate control interface.

Must be run in locked state and validated USB device..

#### **Return Value**

USB\_OK - success.

Others (failure)

### **4.3.19 usb\_class\_cdc\_unbind\_acm\_interface**

#### **Synopsis**

```
usb_status usb_class_cdc_unbind_acm_interfaces
(
    cdc_class_call_struct_t *      ccs_ptr
)
```

#### **Parameters**

ccs\_ptr                      [in] – the communication device control instance structure

#### **Description**

Data interface (specified by ccs\_ptr) will be unbound from appropriate control interface.

Must be run in locked state and validated USB device.

#### **Return Value**



USB\_OK - success.

Others (failure)

### 4.3.20 usb\_class\_cdc\_init\_ipipe

#### Synopsis

```
usb_status usb_class_cdc_init_ipipe
(
    cdc_class_call_struct_t *    acm_instance
)
```

#### Parameters

acm\_instance                      [in] – ACM interface instance

#### Description

Starts interrupt endpoint to poll for interrupt on specified device.

#### Return Value

USB\_OK - success.

Others (failure)

### 4.3.21 usb\_class\_cdc\_intf\_validate

#### Synopsis

```
uint32_t usb_class_cdc_intf_validate
(
    void * param
)
```

#### Parameters

param                      [in] – the pointer to interface

#### Description

This function is called to determine whether class interface is validated.

#### Return Value

1 – valid

0 – invalid

## 4.4 PHDC class driver APIs

### 4.4.1 usb\_class\_phdc\_init

#### Synopsis

```
usb_status usb_class_phdc_init
```

```
(
usb_device_instance_handle      dev_handle,
usb_interface_descriptor_handle intf_handle,
class_handle*                   class_handle_ptr
)
```

### Parameters

dev\_handle                    [in] – device handle  
 intf\_handle                [in] – interface handle  
 class\_handle\_ptr           [out] – the class driver's handle, refer to [6.5.1](#)

### Description

This function is used to initialize the corresponding class driver, it is not called by the APP directly, normally, the class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by usb\_host\_open\_dev\_interface, the init function will be called automatically and the class driver's handle will be returned.

### Return Value

USB\_OK- success  
 Others (failure)

## 4.4.2 usb\_class\_phdc\_deinit

### Synopsis

```
usb_status usb_class_phdc_deinit
(
class_handle      handle
)
```

### Parameters

handle                    [in] – the class driver's handle

### Description

This function is used to deinit a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the usb\_host\_close\_dev\_interface() function.

### Return Value

USB\_OK- success  
 Others (failure)

## 4.4.3 usb\_class\_phdc\_pre\_deinit

### Synopsis

```
usb_status usb_class_phdc_pre_deinit
(
```

```

class_handle
)
handle

```

### Parameters

handle [in] – the class driver's handle

### Description

This function is used to do some pre deinit operation like cancel all the uncompleted transfer. This function is not called by the APP directly, it will be included in a global class interface table so that it can be called automatically when the device detached.

### Return Value

USB\_OK- success

Others (failure)

## 4.4.4 usb\_class\_phdc\_recv\_data

### Synopsis

```

usb_status usb_class_phdc_recv_data
(
usb_phdc_param_t* call_param_ptr
)

```

### Parameters

usb\_phdc\_param\_t [in] – call param struct pointer, refer to [6.5.2](#)

### Description

This function is called to receive data from either a bulk-in or an interrupt pipe.

## 4.4.5 usb\_class\_phdc\_send\_control\_request

### Synopsis

```

usb_status usb_class_phdc_send_control_request
(
usb_phdc_param_t* call_param_ptr
)

```

### Parameters

usb\_phdc\_param\_t [in] – call param struct pointer

### Description

This function is called by application to send a PHDC class specific request.

### Return Value

USB\_OK- success

Others (failure)

## 4.4.6 usb\_class\_phdc\_send\_data

### Synopsis

```
usb_status usb_class_phdc_send_data
(
    usb_phdc_param_t*                call_param_ptr
)
```

### Parameters

usb\_phdc\_param\_t [in] – call param struct pointer

### Description

This function is called to send data to a PHDC device through a bulk-out pipe.

### Return Value

USB\_OK- success

Others (failure)

## 4.4.7 usb\_class\_phdc\_set\_callbacks

### Synopsis

```
usb_status usb_class_phdc_set_callbacks
(
    class_handle          handle,
    phdc_callback         sendCallback,
    phdc_callback         recvCallback,
    phdc_callback         ctrlCallback
)
```

### Parameters

handle [in] – PHDC class handle  
sendCallback [in] – send callback pointer  
recvCallback [in] – receive callback pointer  
ctrlCallback [in] – control callback pointer

### Description

This function is called by application to register application callback pointers for the current PHDC interface.

### Return Value

USB\_OK- success

Others (failure)

## 4.5 Audio class driver

### 4.5.1 usb\_class\_audio\_control\_init

#### Synopsis

```
usb_status usb_class_audio_control_init
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle *                class_handle_ptr
)
```

#### Parameters

dev\_handle                      [in] – device handle  
intf\_handle                    [in] – interface handle  
class\_handle\_ptr               [out] – the class driver's handle, refer to [6.6.1](#)

#### Description

This function is used to initialize the corresponding class driver, it is not called by the APP directly, normally, the class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by usb\_host\_open\_dev\_interface, the init function will be called automatically and the class driver's handle will be returned.

#### Return Value

USB\_OK- success

Others (failure)

### 4.5.2 usb\_class\_audio\_control\_deinit

#### Synopsis

```
usb_status usb_class_audio_control_deinit
(
    class_handle    handle
)
```

#### Parameters

handle                      [in] – the class driver's handle

#### Description

This function is used to deinit a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the usb\_host\_close\_dev\_interface() function.

#### Return Value

USB\_OK- success

Others (failure)

### 4.5.3 usb\_class\_audio\_control\_pre\_deinit

#### Synopsis

```
usb_status usb_class_audio_control_pre_deinit
(
    class_handle          handle
)
```

#### Parameters

handle [in] – the class driver's handle

#### Description

This function is used to do some pre deinit operation like cancel all the uncompleted transfer. This function is not called by the APP directly, it will be included in a global class interface table so that it can be called automatically when the device detached.

#### Return Value

USB\_OK- success

Others (failure)

### 4.5.4 usb\_class\_audio\_stream\_init

#### Synopsis

```
usb_status usb_class_audio_stream_init
(
    usb_device_instance_handle dev_handle,
    usb_interface_descriptor_handle intf_handle,
    class_handle *              class_handle_ptr
)
```

#### Parameters

dev\_handle [in] – device handle

intf\_handle [in] – interface handle

class\_handle\_ptr [out] – the class driver's handle, refer to [6.6.2](#)

#### Description

This function is used to initialize the corresponding class driver, it is not called by the APP directly, normally, the class driver's init/deinit functions are included in the global interface map table. When corresponding interface is opened by usb\_host\_open\_dev\_interface, the init function will be called automatically and the class driver's handle will be returned.

#### Return Value

USB\_OK- success

Others (failure)

## 4.5.5 usb\_class\_audio\_stream\_deinit

### Synopsis

```
usb_status usb_class_audio_stream_deinit
(
    class_handle          handle
)
```

### Parameters

handle [in] – the class driver's handle

### Description

This function is used to deinit a class driver handle,

This function is not called by the APP directly. It will be called when the USB host application call the usb\_host\_close\_dev\_interface() function.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.6 usb\_class\_audio\_stream\_pre\_deinit

### Synopsis

```
usb_status usb_class_audio_control_pre_deinit
(
    class_handle          handle
)
```

### Parameters

handle [in] – the class driver's handle

### Description

This function is used to do some pre deinit operation like cancel all the uncompleted transfer. This function is not called by the APP directly, it will be included in a global class interface table so that it can be called automatically when the device detached.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.7 usb\_class\_audio\_control\_get\_descriptors

### Synopsis

```
usb_status usb_class_audio_control_get_descriptors
(
    usb_device_instance_handle    dev_handle,
    usb_interface_descriptor_handle    intf_handle,
```

```

        usb_audio_ctrl_desc_header_t* *      header_desc,
        usb_audio_ctrl_desc_it_t* *         it_desc,
        usb_audio_ctrl_desc_ot_t**          ot_desc,
        usb_audio_ctrl_desc_fu_t* *         fu_desc
    )

```

### Parameters

dev_handle	[in] - pointer to device instance
intf_handle	[in] - pointer to interface descriptor
header_desc	[out] - pointer to header descriptor
it_desc	[out] - pointer to input terminal descriptor
ot_desc	[out] - pointer to output terminal descriptor
fu_desc	[out] - pointer to feature unit descriptor

### Description

This function is hunting for descriptors in the device configuration and fills back fields if the audio control descriptor was found.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.8 usb\_class\_audio\_stream\_get\_descriptors

### Synopsis

```

usb_status usb_class_audio_stream_get_descriptors
(
    usb_device_instance_handle      dev_handle,
    usb_interface_descriptor_handle intf_handle,
    usb_audio_stream_desc_spepific_as_if_t* * as_itf_desc,
    usb_audio_stream_desc_format_type_t* *   frm_type_desc,
    usb_audio_stream_desc_specific_iso_endp_t* * iso_endp_spec_desc
)

```

### Parameters

dev_handle	[in] - pointer to device instance
intf_handle	[in] - pointer to interface descriptor
as_itf_desc	[out] - pointer to specific audio stream interface descriptor
frm_type_desc	[out] - pointer to format type descriptor
iso_endp_spec_desc	[out] - pointer to specific isochronous endpoint descriptor

### Description



This function is hunting for descriptors in the device configuration and fills back fields if the audio stream descriptor was found.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.9 usb\_class\_audio\_control\_get\_descriptors

### Synopsis

```
usb_status usb_class_audio_control_get_descriptors
(
    class_handle                      handle
    usb_interface_descriptor_handle  intf_handle,
    usb_audio_ctrl_desc_header_t*    header_desc,
    usb_audio_ctrl_desc_it_t*        it_desc,
    usb_audio_ctrl_desc_ot_t*        ot_desc,
    usb_audio_ctrl_desc_fu_t*        fu_desc
)
```

### Parameters

handle	[in] – handle of the class
header_desc	[in] – pointer to header descriptor
it_desc	[in] – pointer to input terminal descriptor
ot_desc	[in] – pointer to output terminal descriptor
fu_desc	[in] – pointer to feature unit descriptor

### Description

This function is used to set the audio control descriptors for control interface descriptors can be used afterwards by application or by driver.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.10 usb\_class\_audio\_stream\_set\_descriptors

### Synopsis

```
usb_status usb_class_audio_stream_set_descriptors
(
    class_handle                      handle,
    usb_audio_stream_desc_specific_as_if_t* * as_itf_desc,
```

```

        usb_audio_stream_desc_format_type_t* *      frm_type_desc,
        usb_audio_stream_desc_specific_iso_endp_t* * iso_endp_spec_desc
    )

```

### Parameters

handle [in] – handle of the class

as\_itf\_desc [in] – pointer to specific audio stream interface descriptor

frm\_type\_desc [in] – pointer to format type descriptor

iso\_endp\_spec\_desc [in] – pointer to specific isochronous endpoint descriptor

### Description

This function is used to set the audio stream descriptors for stream interface descriptors can be used afterwards by application or by driver.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.11 usb\_class\_audio\_cntrl\_common

### Synopsis

```

usb_status usb_class_audio_cntrl_common
(
    audio_command_t*      com_ptr,
    uint8_t               bmrequesttype,
    uint8_t               brequest,
    uint16_t              wvalue,
    uint16_t              windex,
    uint16_t              wlength,
    uint8_t *             data
)

```

### Parameters

com\_ptr [in] – The communication device common command structure, refer to [6.6.3](#)

bmrequesttype [in] – pointer to format type descriptor

brequest [in] – Bitmask of the request type

wvalue [in] – Value to copy into wvalue field of the REQUEST

windex [in] – Value to copy into windex field of the REQUEST

wlength [in] – Length of the data associated with REQUEST

data [in] – Pointer to data buffer used to send/recv

### Description

This function is used to send a control request.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.12 usb\_class\_audio\_cntrl\_callback

### Synopsis

```
void usb_class_audio_cntrl_callback
(
void*                tr_ptr,
void*                param,
uint8_t *           buffer,
uint32_t             len,
usb_status            status
)
```

### Parameters

tr_ptr [in]	[in] - unused
param [in]	[in] - The pointer of class interface instance
buffer	[in] - Data buffer
len	[in] - Length of buffer
status	[out] - USB_OK- success Others (failure)

### Description

This is the callback used when audio control information is sent or received.

### Return Value

No return value.

## 4.5.13 usb\_class\_audio\_recv\_data

### Synopsis

```
usb_status usb_class_audio_recv_data
(
audio_command_t*    audio_ptr,
uint8_t *           buffer,
uint32_t             buf_size
)
```

### Parameters

audio_ptr	[in] - audio control class interface pointer
-----------	--

buffer [in] – Data buffer  
buf\_size [in] – Length of buffer

### Description

The function is used to receive data on isochronous IN pipe.

### Return Value

USB\_OK- success

Others (failure)

## 4.5.14 usb\_class\_audio\_recv\_callback

### Synopsis

```
void usb_class_audio_recv_callback  
(  
    void*          tr_ptr,  
    void*          param,  
    uint8_t *      buffer,  
    uint32_t       len,  
    usb_status     status  
)
```

### Parameters

tr\_ptr [in] [in] - unused  
param [in] [in] - The pointer of class interface instance  
buffer [in] [in] - Data buffer  
len [in] [in] - Length of buffer  
status [out] – USB\_OK mean success Others (failure)

### Description

This is the callback used when audio receive data on isochronous IN pipe.

### Return Value

No return value.

## 4.5.15 usb\_class\_audio\_send\_data

### Synopsis

```
usb_status usb_class_audio_send_data  
(  
    audio_command_t* audio_ptr,  
    uint8_t *        buffer,
```

```
uint32_t          buf_size
)

```

### Parameters

audio_ptr	[in] – audio control class interface pointer
buffer	[in] – Data buffer
buf_size	[in] – Length of buffer

### Description

The function is used to send data on isochronous out pipe.

### Return Value

USB\_OK- success  
Others (failure)

## 4.5.16 usb\_class\_audio\_send\_callback

### Synopsis

```
void usb_class_audio_send_callback
(
void*          tr_ptr,
void*          param,
uint8_t *      buffer,
uint32_t       len,
usb_status     status
)

```

### Parameters

tr_ptr [in]	[in] – unused
param [in]	[in] - The pointer of class interface instance
buffer	[in] – Data buffer
len	[in] – Length of buffer
status	[out] – USB_OK mean success Others (failure)

### Description

This is the callback used when audio send data on isochronous out pipe.

### Return Value

No return value.

## Chapter 5 USB Host Configuration

### 5.1 Common configure

#### 5.1.1 USBCFG\_HOST\_KHCI

If macro is no-zero, enable KHCI controller driver (full speed), otherwise disable.

#### 5.1.2 USBCFG\_HOST\_EHCI

If macro is no-zero, enable EHCI controller driver (high speed), otherwise disable.

#### 5.1.3 USBCFG\_HOST\_MAX\_PIPES

How many pipes HOST stack supports?

Such as:

```
#define USBCFG_HOST_MAX_PIPES (16)
```

It indicates host supports max pipes are 16.

#### 5.1.4 USBCFG\_HOST\_DEFAULT\_MAX\_NAK\_COUNT

MAX retries times, when receive a NAK.

Such as:

```
#define USBCFG_HOST_DEFAULT_MAX_NAK_COUNT (3000)
```

It indicates the largest continuous NAK retry times are 3000.

#### 5.1.5 USBCFG\_HOST\_CTRL\_FAILED\_RETRY

Control pipe retries times, when host get an error.

Such as:

```
#define USBCFG_HOST_CTRL_FAILED_RETRY (3)
```

It indicates the max continuous error retry times are 3.

#### 5.1.6 USBCFG\_HOST\_CTRL\_STALL\_RETRY

Control pipe retries times, when host get an error.

Such as:

```
#define USBCFG_HOST_CTRL_STALL_RETRY (3)
```

It indicates the max continuous error retry times are 3.

### 5.1.7 USBCFG\_HOST\_MAX\_POWER

MAX power host can support. The unit is 2mA.

Such as:

```
#define USBCFG_HOST_MAX_POWER (250)
```

It indicates the max power is 500mA for every device.

### 5.1.8 USBCFG\_HOST\_MAX\_CONFIGURATION\_PER\_DEV

How many configure descriptor device has, host supports?

Such as:

```
#define USBCFG_HOST_MAX_CONFIGURATION_PER_DEV (2)
```

It indicates the max configure descriptor number is 2 for every device.

### 5.1.9 USBCFG\_HOST\_MAX\_INTERFACE\_PER\_CONFIGURATION

How many interface for each configure descriptor, host supports?

Such as:

```
#define USBCFG_HOST_MAX_INTERFACE_PER_CONFIGURATION (4)
```

It indicates the max interface number of each configure descriptor is 4.

### 5.1.10 USBCFG\_HOST\_MAX\_EP\_PER\_INTERFACE

How many ep for each interface descriptor, host supports?

Such as:

```
#define USBCFG_HOST_MAX_EP_PER_INTERFACE (4)
```

It indicates the max ep number of each interface descriptor is 4.

### 5.1.11 USBCFG\_HOST\_HID

1 indicates the HID class driver is enabled

0 indicates the HID class driver is disabled

### 5.1.12 USBCFG\_HOST\_MSD

1 indicates the MSD class driver is enabled

0 indicates the MSD class driver is disabled

### **5.1.13 USBCFG\_HOST\_CDC**

1 indicates the CDC class driver is enabled

0 indicates the CDC class driver is disabled

### **5.1.14 USBCFG\_HOST\_PHDC**

1 indicates the PHDC class driver is enabled

0 indicates the PHDC class driver is disabled

### **5.1.15 USBCFG\_HOST\_AUDIO**

1 indicates the AUDIO class driver is enabled

0 indicates the AUDIO class driver is disabled

### **5.1.16 USBCFG\_HOST\_HUB**

1 indicates the HUB class driver is enabled

0 indicates the HUB class driver is disabled

### **5.1.17 USBCFG\_HOST\_PRINTER**

1 indicates the PRINTER class driver is enabled

0 indicates the PRINTER class driver is disabled

### **5.1.18 USBCFG\_BUFF\_PROPERTY\_CACHEABLE**

1 cacheable, buffer cache maintenance is needed

0 uncacheable, buffer cache maintenance is not needed

## **5.2 KHCI configure**

### **5.2.1 USBCFG\_HOST\_KHCI\_TASK\_PRIORITY**

The priority of khci task.

### **5.2.2 USBCFG\_HOST\_KHCI\_TR\_QUE\_MSG\_CNT**

The max msg count in message queue.

### **5.2.3 USBCFG\_HOST\_KHCI\_MAX\_INT\_TR**

The max TR count for khci controller driver.



## **5.2.4 USBCFG\_KHCI\_4BYTE\_ALIGN\_FIX**

The Full Speed controller requires all the buffers used for the transfer need to be 4 bytes aligned (both the start address and the length). If the application can guarantee it, then the MACRO USBCFG\_KHCI\_4BYTE\_ALIGN\_FIX can be set to 0. Otherwise, it needs to set to 1, then the USB HostStack will use a internal 4 bytes aligned buffer to replace the buffer provided by the user so that the above requirement can be meet.

And the internal buffer size is assigned by the MACRO USBCFG\_HOST\_KHCI\_SWAP\_BUF\_MAX.

## **5.2.5 USBCFG\_HOST\_PORT\_NATIVE**

This macro is only valid when using micro usb port of TWR board.

If using micro usb port of TWR board, the macro should be set to (1).

If using SER board usb port, it should be set to (0).

## **5.3 EHCI configure**

### **5.3.1 USBCFG\_EHCI\_USE\_SW\_TOGGLING**

1 indicates host uses software toggle.

0 indicates host not uses software toggle.

### **5.3.2 USBCFG\_EHCI\_MAX\_QH\_DESCRS**

How many QH HOST stacks are supported at the maximum?

### **5.3.3 USBCFG\_EHCI\_MAX\_QTD\_DESCRS**

How many QTD HOST stacks are supported at the maximum?

### **5.3.4 USBCFG\_EHCI\_MAX\_ITD\_DESCRS**

How many ITD HOST stacks are supported at the maximum?

### **5.3.5 USBCFG\_EHCI\_MAX\_SITD\_DESCRS**

How many SITD HOST stacks are supported at the maximum?

### **5.3.6 USBCFG\_EHCI\_PIPE\_TIMEOUT**

Maximum pipe timeout number.

### **5.3.7 USBCFG\_EHCI\_FRAME\_LIST\_SIZE**

EHCI periodic list frame size.

---

### **5.3.8 USBCFG\_EHCI\_HS\_DISCONNECT\_ENABLE**

1 indicates the connect function is enabled.

0 indicates the connect function is disabled.

## Chapter 6 Data Structures

### 6.1 USB host controller driver structures

#### 6.1.1 usb\_setup\_t

All USB devices respond to requests from the host on the device's Default Control Pipe. These requests are made using control transfers. The request and the request's parameters are sent to the device in the Setup packet. The host is responsible for establishing the values passed in the fields listed in the struct.

##### Synopsis

```
typedef struct
{
    uint8_t                bmrequesttype;
    uint8_t                brequest;
    uint8_t                wvalue[2];
    uint8_t                windex[2];
    uint8_t                wlength[2];
} usb_setup_t;
```

##### Fields

- bmrequesttype - Characteristics of request.
- brequest - Specific request.
- wvalue - Word-sized field that varies according to request
- windex - Word-sized field that varies according to request; typically used to pass an index or offset.
- wlength - Number of bytes to transfer if there is a Data stage.

#### 6.1.2 tr\_struct\_t

TR struct represents a transfer. A TR will be allocated, filled and send to HCI when apps, class drivers or controller start a transfer. And HCI decomposes the TR into transactions and then does the actual transmission.

##### Synopsis

```
typedef struct tr_struct
{
    struct tr_struct*      next;
    uint32_t               status;
    uint32_t               tr_index;
    uint8_t *              tx_buffer;
    uint8_t *              rx_buffer;
    uint32_t               tx_length;
    uint32_t               rx_length;
```

```

bool                send_phase;
usb_setup_t         setup_packet;
uint32_t            transfered_length;
tr_callback          callback;
void*                callback_param;
void*                hw_transaction_head; /* used only for EHCI */
void*                hw_transaction_tail; /* used only for EHCI */
uint8_t             occupied;
uint8_t             setup_status;
uint8_t             no_of_itds_sitds;
uint8_t             setup_first_phase;
} tr_struct_t;

```

## Fields

next	- A pointer to save next TR address
status	- Save TR status.
tr_index	- Transfer number on current usb_host_ptr.
tx_buffer	- To be sent data buffer address.
rx_buffer	- To be received data buffer address.
tx_length	- Send data length. For control transfers, the length of data for the data phase.
rx_length	- Recv data length. For control transfers, the length of data for the data phase.
send_phase	- TR dir. For control transfers: Send/Receive.
setup_packet	- Setup packet raw data.
transfered_length	- The data length has been transferred.
callback	- The callback function to be invoked when a transfer is completed or an error is to be reported.
callback_param	- The second parameter to be passed into the callback function when it is invoked.
hw_transaction_head	- HW struct head pointer, used only for EHCI.
hw_transaction_tail	- HW struct tail pointer, used only for EHCI.
occupied	- Is used or not.
setup_status	- Setup transfer status.
no_of_itds_sitds	- The numbers of itds or sitds for the TR.
setup_first_phase	- Is the setup packet is sent.

### 6.1.3 pipe\_init\_struct\_t

Pipe init struct is used to set pipe params when calling usb\_host\_open\_pipe().

## Synopsis

```
typedef struct pipe_init_struct
{
    void*                dev_instance;
    uint32_t             flags;
    uint16_t             max_packet_size;
    uint16_t             nak_count;
    uint8_t              interval;
    uint8_t              endpoint_number;
    uint8_t              direction;
    uint8_t              pipetype;
} pipe_init_struct_t;
```

## Fields

dev_instance	- The device instance of this pipe.
flags	- Pipe flags.
max_packet_size	- Max pipe's packet size.
nak_count	- Max NAK retry count. MUST be zero for interrupt.
interval	- Interval for polling pipe for data transfer.
endpoint_number	- The device's ep number of this pipe.
direction	- The pipe direction. #define USB_RECV (0) #define USB_SEND (1)
pipetype	- The transfer type of this pipe. #define USB_CONTROL_PIPE (0x00) #define USB_ISOCHRONOUS_PIPE (0x01) #define USB_BULK_PIPE (0x02) #define USB_INTERRUPT_PIPE (0x03)

### 6.1.4 pipe\_struct\_t

The struct is used to establish interface between host and device. Used by usb\_host\_open\_pipe().

## Synopsis

```
typedef struct pipe_struct
{
    struct pipe_struct *next;
    tr_struct_t*        tr_list_ptr;
    void*               dev_instance;
```

```

uint32_t          flags;
uint16_t          max_packet_size
uint16_t          nak_count;
uint8_t           interval;
uint8_t           endpoint_number;
uint8_t           direction;
uint8_t           pipetype;
uint8_t           trs_per_uframe;
uint8_t           open;
uint8_t           nextdata01;
} pipe_struct_t;

```

## Fields

next	- A pointer to save next pipe struct address
tr_list_ptr	- List of TRs linked to this pipe.
dev_instance	- The device instance of this pipe.
flags	- After all data transferred, should we terminate the transfer with a zero length packet if the last packet size == max_packet_size?
max_packet_size	- Max pipe's packet size.
nak_count	- Max NAK retry count. MUST be zero for interrupt.
interval	- Interval for polling pipe for data transfer.
endpoint_number	- The device's ep number of this pipe.
direction	- The pipe direction.
	#define USB_RECV (0)
	#define USB_SEND (1)
pipetype	- The transfer type of this pipe.
	#define USB_CONTROL_PIPE (0x00)
	#define USB_ISOCHRONOUS_PIPE (0x01)
	#define USB_BULK_PIPE (0x02)
	#define USB_INTERRUPT_PIPE (0x03)
trs_per_uframe	- Number of transaction per frame, only high-speed high-bandwidth pipes.
open	- Is opened or not.
nextdata01	- Endpoint data toggling bit.

## 6.1.5 usb\_host\_driver\_info\_t

Information for one class or device driver, Used by usb\_host\_register\_driver\_info().

## Synopsis

```
typedef struct driver_info
{
    uint8_t            idVendor[2];
    uint8_t            idProduct[2];
    uint8_t            bDeviceClass;
    uint8_t            bDeviceSubClass;
    uint8_t            bDeviceProtocol;
    uint8_t            reserved;
    event_callback      attach_call;
} usb_host_driver_info_t;
```

## Fields

idVendor[2]	- Vendor ID per USB-IF.
idProduct[2]	- Product ID per manufacturer.
bDeviceClass	- Class code, 0xFF if any.
bDeviceSubClass	- Sub-Class code, 0xFF if any.
bDeviceProtocol	- Protocol, 0xFF if any.
reserved	- Alignment padding .
attach_call	- The function to call when above information matches the one in device's descriptors.

### 6.1.6 dev\_instance\_t

The struct stands by a device. When a device attached, a struct will be created. And when a device detached, the struct will be removed. Used by `usb_host_dev_mng_attach()`.

## Synopsis

```
typedef struct dev_instance
{
    struct dev_instance *next;
    usb_host_handle      host;
    hub_device_struct_t* hub_instance;
    uint8_t              speed;
    uint8_t              hub_no;
    uint8_t              port_no;
    uint8_t              address;
    uint16_t             cfg_value;
    uint8_t              ctrl_retries;
    uint8_t              stall_retries;
    uint8_t              new_config;
```

```

uint8_t          num_of_interfaces;
uint8_t          rerserved1[1];
uint16_t         state;
usb_pipe_handle  control_pipe;
tr_callback      control_callback;
void*            control_callback_param;
device_descriptor_t dev_descriptor;
uint8_t          rerserved2[2];
uint8_t          buffer[9];
uint8_t          rerserved3[3];
void*            lpConfiguration;
usb_device_configuration_struct_t configuration;
usb_device_interface_info_struct_t
interface_info[USBCFG_HOST_MAX_INTERFACE_PER_CONFIGURATION];
uint8_t          attached;
uint8_t          pre_detached;
uint8_t          to_be_detached;
uint8_t          target_address;
uint8_t          level;
} dev_instance_t;

```

## Fields

next	- A pointer to save next struct address
host	- The host of the device attached.
hub_instance	- HUB instance, this device linked.
speed	- Device speed. 0 – full-speed transfer 1 – low-speed transfer 2 – high-speed transfer
hub_no	- The HUB number, root hub is 0.
port_no	- The HUB's port(1~8), this device linked.
address	- Device address(1~127).
cfg_value	- Configure value.
ctrl_retries	- Retry count of control when transfer failed.
stall_retries	- Retry count of control when ep stalled.
new_config	- non-zero = new config.
num_of_interfaces	- The interface numbers of device.
rerserved1[1]	- Rerserved.



state	- Device state.
control_pipe	- Control pipe handle.
control_callback	- The callback function to be invoked when a control transfer is completed or an error is to be reported.
control_callback_param	- The second parameter to be passed into the control_callback function when it is invoked.
dev_descriptor	- Device descriptor.
reserved2[2]	- Reserved.
buffer[9]	- enumeration buffer.
reserved3[3]	- Reserved.
lpConfiguration	- Configure descriptor pointer.
configuration	- Save the device configuration
interface_info	- Save information of the matched interface.
attached	- Device attached or not.
pre_detached	- Device pre-detached or not.
to_be_detached	- Device need to be detached or not.
target_address	- Temp address, assign for device.
level	- Device level.

### 6.1.7 usb\_host\_state\_struct\_t

This struct is used to keep usb host status.

#### Synopsis

```
typedef struct usb_host_generic_structure
{
    uint8_t                occupied;
    uint8_t                controller_id;
    usb_host_handle        controller_handle;
    const usb_host_api_functions_struct_t * host_controller_api;
    event_callback         unsupport_device_callback;
    void*                  device_list_ptr;
    struct driver_info *    device_info_table;
    os_mutex_handle        mutex;
    os_sem_handle          hub_sem;
    os_mutex_handle        hub_mutex;
    void*                  hub_link;
}
```

```

uint32_t          hub_task;
tr_struct_t       tr_list[MAX_TR_NUMBER];
uint32_t          tr_index;
void*             root_hub_ptr;
usb_host_service_struct_t services[MAX_HOST_SERVICE_NUMBER];
uint8_t           tr_user;

#ifdef USBCFG_OTG
usb_otg_handle     otg_handle;
#endif
} usb_host_state_struct_t;

```

## Fields

occupied	- Is used or not.
controller_id	- HW controller ID.
controller_handle	- HW controller handle.
host_controller_api	- HW controller API list.
unsupport_device_callback	- The callback function to be invoked when unsupported device attached.
device_list_ptr	- Device instance list connected to this host.
device_info_table	- Supported device info table.
mutex	- Mutex for host.
hub_sem	- Semaphore for HUB class driver.
hub_mutex	- Mutex for HUB class driver.
hub_link	- HUB instance list attached to this host
hub_task	- HUB task handle.
tr_list	- TRs for this host.
tr_index	- TR count is used.
root_hub_ptr	- Root HUB instance handle.
services	- services for pipe or specific event.
tr_user	- TR count in using.
otg_handle	- OTG handle.

## 6.2 HID class structures

### 6.2.1 usb\_hid\_class\_struct\_t

HID class struct.

## Synopsis

```
typedef struct _usb_hid_class
{
    usb_host_handle          host_handle;
    usb_device_instance_handle dev_handle;
    usb_interface_descriptor_handle intf_handle;
    bool                     in_setup;
    usb_pipe_handle          in_pipe;
    /* Here we store callback and parameter from higher level */
    tr_callback              ctrl_callback;
    void*                    ctrl_param;
    tr_callback              recv_callback;
    void*                    recv_param;
    uint32_t                 running;
} usb_hid_class_struct_t;
```

## Fields

host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
in_setup	- Class driver in setup phase or not.
in_pipe	- Pointer to in pipe.
ctrl_callback	- The function to call when control transfer callback.
ctrl_param	- The second parameter to be passed into the callback function when it is invoked.
recv_callback	- The function to call when data recved.
recv_param	- The second parameter to be passed into the callback function when it is invoked.
running	- It is running.

## 6.2.2 hid\_command\_t

The HID command structure.

## Synopsis

```
typedef struct _hid_command
{
    class_handle      class_ptr;
    tr_callback       callback_fn;
    void*             callback_param;
```

```
} hid_command_t;
```

## Fields

class_ptr	- Pointer to class call structure.
callback_fn	- Callback function.
callback_param	- Callback function parameter.

## 6.3 MSD class structures

### 6.3.1 usb\_mass\_class\_struct\_t

MSD class struct.

#### Synopsis

```
typedef struct
{
    usb_host_handle                host_handle;
    usb_device_instance_handle     dev_handle;
    usb_interface_descriptor_handle intf_handle;
    usb_pipe_handle               control_pipe;
    usb_pipe_handle               bulk_in_pipe;
    usb_pipe_handle               bulk_out_pipe;
    mass_queue_struct_t           queue;
    uint8_t                       interface_num;
    uint8_t                       alternate_setting;
    tr_callback                   ctrl_callback;
    void *                        ctrl_param;
    os_mutex_handle               mutex;
} usb_mass_class_struct_t;
```

## Fields

host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
control_pipe	- Control pipe handle.
bulk_in_pipe	- Bulk in pipe handle.
bulk_out_pipe	- Bulk out pipe handle.
queue	- The queue used to save cmd.
interface_num	- Interface number.
alternate_setting	- Interface alternate setting

ctrl_callback	- The function to call when control transfer callback.
ctrl_param	- The second parameter to be passed into the callback function when it is invoked.
mutex	- Mutex for msd class driver.

### 6.3.2 mass\_command\_struct\_t

The MSD command structure.

#### Synopsis

```
typedef struct
{
    class_handle                CLASS_PTR;
    uint32_t                    LUN;
    cbw_struct_t *              CBW_PTR;
    csw_struct_t *              CSW_PTR;
    void (_CODE_PTR_)           CALLBACK)
        (usb_status,
         void*,
         void*,
         uint32_t
        );

    void*                       DATA_BUFFER;
    uint32_t                     BUFFER_LEN;
    uint32_t                     BUFFER_SO_FAR;
    usb_class_mass_command_status STATUS;
    usb_class_mass_command_status PREV_STATUS;
    uint32_t                     TR_BUF_LEN;
    uint8_t                      RETRY_COUNT;
    uint8_t                      CBW_RETRY_COUNT;
    uint8_t                      DPHASE_RETRY_COUNT;
    uint8_t                      CSW_RETRY_COUNT;
    uint8_t                      IS_STALL_IN_DPHASE;
    uint8_t                      TR_INDEX;
} mass_command_struct_t;
```

#### Fields

class_ptr	- Pointer to class handle.
LUN	- Logical unit number on device.
callback_param	- Callback function parameter.
CBW_PTR	- Current CBW being constructed
CSW_PTR	- CSW for this command

<b>CALLBACK</b>	- Callback function
<b>DATA_BUFFER</b>	- Buffer for IN/OUT for the command
<b>BUFFER_LEN</b>	- Length of data buffer
<b>BUFFER_SOFAR</b>	- Number of bytes trans so far
<b>STATUS</b>	- Current status of this command
<b>PREV_STATUS</b>	- Previous status of this command
<b>TR_BUF_LEN</b>	- Length of the buffer received in currently executed TR
<b>RETRY_COUNT</b>	- Number of times this commad tried
<b>CBW_RETRY_COUNT</b>	- Number of times this commad tried
<b>DPHASE_RETRY_COUNT</b>	- Number of times this commad tried
<b>CSW_RETRY_COUNT</b>	- Number of times this commad tried
<b>IS_STALL_IN_DPHASE</b>	- Is stall happened in data dpase
<b>TR_INDEX</b>	- TR_INDEX of the TR used for search

## 6.4 CDC class structures

### 6.4.1 usb\_acm\_class\_intf\_struct\_t

CDC Control interface struct.

#### Synopsis

```
typedef struct {
    usb_cdc_desc_acm_t *      acm_desc;
    usb_cdc_desc_cm_t *      cm_desc;
    usb_cdc_desc_header_t *  header_desc;
    usb_cdc_desc_union_t *   union_desc;
    usb_cdc_uart_coding_t    uart_coding;
    usb_pipe_handle          interrupt_pipe;
    usb_cdc_acm_state_t      interrupt_buffer;
    usb_cdc_ctrl_state_t     ctrl_state;
    os_event_handle          acm_event;
    usb_host_handle          host_handle;
    usb_device_instance_handle dev_handle;
    usb_interface_descriptor_handle intf_handle;
    uint8_t                 intf_num;
    os_mutex_handle          mutex;
} usb_acm_class_intf_struct_t;
```

#### Fields

**acm\_desc** - ACM descriptor handle.

cm_desc	- CM descriptor handle.
header_desc	- Header descriptor handle.
union_desc	- Union descriptor handle.
uart_coding	- Current uart coding config.
interrupt_pipe	- Interrupt pipe handle.
interrupt_buffer	- Interrupt transfer buffer.
ctrl_state	- Control pipe state.
acm_event	- ACM event.
host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
intf_num	- Interface number.
mutex	- Mutex for ACM interface.

## 6.4.2 usb\_data\_class\_intf\_struct\_t

CDC Data interface struct.

### Synopsis

```
typedef struct {
    cdc_class_call_struct_t *
    uint8_t *
    uint8_t *
    uint8_t *
    uint32_t
    uint32_t
    uint32_t
    usb_pipe_handle
    usb_pipe_handle
    char *
    os_event_handle
    usb_host_handle
    usb_device_instance_handle
    usb_interface_descriptor_handle
    uint8_t
    os_mutex_handle
    tr_callback
    void *

    BOUND_CONTROL_INTERFACE;
    rx_buffer;
    RX_BUFFER_DRV;
    RX_BUFFER_APP;
    RX_BUFFER_SIZE;
    RX_READ;
    TX_SENT;
    in_pipe;
    out_pipe;
    device_name;
    data_event;
    host_handle;
    dev_handle;
    intf_handle;
    intf_num;
    mutex;
    ctrl_callback;
    ctrl_callback_param;
}
```

```

    _usb_cdc_callback    data_tx_cb;
    _usb_cdc_callback    data_rx_cb;
    bool                is_rx_xferring;
    bool                is_tx_xferring;
} usb_data_class_intf_struct_t;

```

## Fields

BOUND_CONTROL_INTERFACE	- Interface handle bound to data interface.
rx_buffer	- Recv buffer.
RX_BUFFER_DRV	- The buffer address, class received.
RX_BUFFER_APP	- The buffer address, app has read last.
RX_BUFFER_SIZE	- rx_buffer size.
RX_READ	- Recv data length.
TX_SENT	- Sent data length.
in_pipe	- In pipe handle.
out_pipe	- Out pipe handle.
device_name	- Device name.
data_event	- Event for data interface.
host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
intf_num	- Interface number.
mutex	- Mutex for data interface.
ctrl_callback	- The function to call when control transfer callback.
ctrl_callback_param	- The second parameter to be passed into the callback
function when it is invoked.	
data_tx_cb	- Data send callback.
data_rx_cb	- Data recv callback.
is_rx_xferring	- Is in recv data phase?
is_tx_xferring	- Is in send data phase?

### 6.4.3 cdc\_command\_t

The CDC command structure.



## Synopsis

```
typedef struct {
    cdc_class_call_struct_t *    CLASS_PTR;
    tr_callback                  CALLBACK_FN;
    void *                       CALLBACK_PARAM;
} cdc_command_t;
```

## Fields

CLASS\_PTR - Pointer to class call structure.

CALLBACK\_FN - Callback function.

CALLBACK\_PARAM - Callback function parameter.

## 6.5 PHDC class structures

### 6.5.1 usb\_phdc\_class\_struct\_t

PHDC class struct.

## Synopsis

```
typedef struct _usb_phdc_class_intf_struct_type
{
    usb_host_handle            host_handle;
    usb_device_instance_handle dev_handle;
    usb_interface_descriptor_handle intf_handle;
    /* Pipes */
    usb_pipe_handle            control_pipe;
    usb_pipe_handle            bulk_in_pipe;
    usb_pipe_handle            bulk_out_pipe;
    usb_pipe_handle            int_in_pipe;
    usb_phdc_desc_qos_metadata_list_t *qos_metadata_list;
    usb_phdc_desc_fcn_ext_t    *fcn_ext_desc;
    /* Callbacks */
    phdc_callback              send_callback;
    phdc_callback              recv_callback;
    phdc_callback              ctrl_callback;
    bool                       preamble_capability;
    uint8_t                    phdc_data_code;
    bool                       set_clear_request_pending;
    bool                       device_feature_set;
    uint8_t                    num_transf_bulk_in;
    uint8_t                    num_transf_bulk_out;
    os_mutex_handle            mutex;
    uint32_t                   running;
} usb_phdc_class_struct_t;
```

## Fields

host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
control_pipe	- Control pipe handle.
bulk_in_pipe	- Bulk in pipe handle.
bulk_out_pipe	- Bulk out pipe handle.
int_in_pipe	- Interrupt in pipe handle.
qos_metadata_list	- QoS and Metadata Linked-List
fcn_ext_desc	- Function extension descriptor
send_callback	- Send data callback
recv_callback	- Recv data callback
ctrl_callback	- Control transfer callback
preamble_capability	- Preamble capability
phdc_data_code	- PHDC data code
set_clear_request_pending	- Pending transfers flags
device_feature_set	- Metadata feature set for the device.
num_transf_bulk_in	- Number of in transfers.
num_transf_bulk_out	- Number of out transfers.
mutex	- Mutex for msd class driver.
running	- It is running.

### 6.5.2 usb\_phdc\_param\_t

The PHDC param structure.

#### Synopsis

```
typedef struct usb_phdc_param_type
{
    class_handle          class_ptr;
    tr_callback           callback_fn;
    void*                 callback_param;
    uint8_t               classRequestType;
    bool                  metadata;
    uint8_t               qos;
```

```

uint8_t          usb_phdc_status;
usb_status       usb_status;
uint8_t*         buff_ptr;
uint32_t         buff_size;
uint32_t         tr_index;
usb_pipe_handle  tr_pipe_handle;
} usb_phdc_param_t;

```

## Fields

class_ptr	- Pointer to class handle.
callback_fn	- Callback function
callback_param	- Callback function parameter.
classRequestType	- the type of the request (only for PHDC Ctrl requests).
metadata	- Boolean for metadata transfers
qos	- QoS for receive transfers (only for PHDC Recv request).
usb_phdc_status	- USB PHDC status code.
usb_status	- USB status code.
buff_ptr	- data buffer (only for PHDC Send/Recv requests).
buff_size	- length of buffer (only for PHDC Send/Recv requests).
tr_index	- USB transaction index. Used to identify the Send/Recv transaction.
tr_pipe_handle	- TR pipe handle.

## 6.6 AUDIO class structures

### 6.6.1 audio\_control\_struct\_t

Audio control subclass structure.

#### Synopsis

```

typedef struct {
    usb_host_handle          host_handle;
    usb_device_instance_handle dev_handle;
    usb_interface_descriptor_handle intf_handle;
    usb_audio_ctrl_desc_header_t* header_desc;
    usb_audio_ctrl_desc_it_t* it_desc;
    usb_audio_ctrl_desc_ot_t* ot_desc;
    usb_audio_ctrl_desc_fu_t* fu_desc;
    uint32_t                 in_setup;
    tr_callback               ctrl_callback;
}

```

```

void*
usb_pipe_handle
usb_audio_control_status_t
tr_callback
void*
tr_callback
void*
uint8_t
} audio_control_struct_t;

ctrl_param;
interrupt_pipe;
interrupt_buffer;
interrupt_callback;
interrupt_callback_param;
user_callback;
user_param;
ifnum;

```

## Fields

ctrl_state	- Control pipe state.
acm_event	- ACM event.
host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
header_desc	- Header descriptor handle.
it_desc	- IT descriptor handle.
ot_desc	- OT descriptor handle.
fu_desc	- FU descriptor handle.
in_setup	- Is in setup phase.
ctrl_callback	- The function to call when control transfer callback.
ctrl_param when it is invoked.	- The second parameter to be passed into the callback function
interrupt_pipe	- Interrupt pipe handle.
interrupt_buffer	- Interrupt transfer buffer.
interrupt_callback	- The function to call when interrupt transfer callback.
interrupt_callback_param function when it is invoked.	- The second parameter to be passed into the callback
user_callback	- App callback function.
user_param	- App callback param.
ifnum	- Interface number.

## 6.6.2 usb\_data\_class\_intf\_struct\_t

AUDIO stream interface struct.

## Synopsis

```
typedef struct {
    usb_host_handle                host_handle;
    usb_device_instance_handle     dev_handle;
    usb_interface_descriptor_handle intf_handle;
    usb_audio_stream_desc_specific_as_if_t* as_itf_desc;
    usb_audio_stream_desc_format_type_t* frm_type_desc;
    usb_audio_stream_desc_specific_iso_endp_t* iso_endp_spec_desc;
    usb_pipe_handle                iso_in_pipe;
    usb_pipe_handle                iso_out_pipe;
    tr_callback                    recv_callback;
    void*                          recv_param;
    tr_callback                    send_callback;
    void*                          send_param;
    os_event_handle                stream_event;
    uint8_t                        iso_ep_num;
} audio_stream_struct_t;
```

## Fields

host_handle	- Pointer to USB host.
dev_handle	- Pointer to device.
intf_handle	- Pointer to interface.
as_itf_desc	- AS interface descriptor.
frm_type_desc	- Format type descriptor.
iso_endp_spec_desc	- ISO ep descriptor.
iso_in_pipe	– ISO pipe in handle.
iso_out_pipe	– ISO pipe out handle.
recv_callback	- Recv stream callback
recv_param	– Recv stream callback param
send_callback	- Send stream callback
send_param	– Send stream callback param
stream_event	– Event for stream interface.
iso_ep_num	– ISO ep number.

### 6.6.3 audio\_command\_t

The AUDIO command structure.

## Synopsis

```

typedef struct {
    class_handle      class_control_handle;
    class_handle      class_stream_handle;
    tr_callback       callback_fn;
    void*             callback_param;
} audio_command_t;

```

## Fields

class_control_handle	- Pointer to class control interface structure.
class_stream_handle	- Pointer to class stream interface structure
callback_fn	- Callback function.
callback_param	- Callback function parameter.

## Chapter 7 OS Adapter

OS adapter is used to provide unified OS API to USB Stack.

### 7.1 OS adapter overview

OS adapter provides many function modules, including Event, MsgQ, Mutex, and Sem.

### 7.2 API overview

#### 7.2.1 Task management

Task management includes create, delete, suspend, resume.

**Table 8 Task management APIs**

No.	API function	Description
1	OS_Task_create()	Create a task, and return value is task id.
2	OS_Task_delete()	Delete a task.
3	OS_Task_suspend()	Suspend a task.
4	OS_Task_resume()	Resume a task.

#### 7.2.2 Event

Event model includes create, destroy, set, wait, check and clear.

**Table 9 Event APIs**

No.	API function	Description
1	OS_Event_create()	Create a Event, and return value is event handle.
2	OS_Event_destroy()	Destroy a event.
3	OS_Event_set()	Set Event.
4	OS_Event_check_bit()	Check Event
5	OS_Event_clear()	Clear Event.
6	OS_Event_wait()	Wait Event.

### 7.2.3 MsgQ

MsgQ model includes create, destroy, send, receive.

**Table 10 MsgQ APIs**

No.	API function	Description
1	OS_MsgQ_create()	Create a MsgQ, and return value is MsgQ handle.
2	OS_MsgQ_destroy()	Destroy a MsgQ.
3	OS_MsgQ_send()	Send a msg
4	OS_MsgQ_recv()	Recv a msg

### 7.2.4 Mutex

Mutex model includes create, destroy, lock, unlock.

**Table 11 Mutex APIs**

No.	API function	Description
1	OS_Mutex_create()	Create a Mutex, and return value is Mutex handle.
2	OS_Mutex_destroy()	Destroy a Mutex.
3	OS_Mutex_lock()	Lock a Mutex.
4	OS_Mutex_unlock()	Unlock a Mutex.

### 7.2.5 Sem

Sem model includes create, destroy, post, wait.

**Table 12 Sem APIs**

No.	API function	Description
1	OS_Sem_create()	Create a Sem, and return value is Sem handle.
2	OS_Sem_destroy()	Destroy a Sem.
3	OS_Sem_post()	Post a Sem.
4	OS_Sem_wait()	Wait a Sem.



## 7.2.6 Mem

Mem model includes alloc, free.

**Table 13 Mem APIs**

No.	API function	Description
1	OS_Mem_alloc()	Alloc memory.
2	OS_Mem_alloc_zero()	Alloc memory, and set space to 0.
3	OS_mem_alloc_uncached() ()	Alloc uncached memory.
4	OS_mem_alloc_uncached_ _zero()	Alloc uncached memety, and set space to 0.
5	OS_Mem_free()	Free memory.
6	OS_Mem_zero()	Set memory space to 0.
7	OS_Mem_copy()	Copy src memory to dst memory.

## 7.2.7 Interrupt

Interrupt model includes install isr.

**Table 14 Int APIs**

No.	API function	Description
1	OS_install_isr()	Install isr function.
2	OS_intr_init()	Init interrupt priority, and disable or enable interrupt.

#### **How to Reach Us:**

##### **Home Page:**

[www.freescale.com](http://www.freescale.com)

##### **Web Support:**

[www.freescale.com/support](http://www.freescale.com/support)

Information in this document is provided solely to enable system and software implementers to use Freescale products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits based on the information in this document.

Freescale reserves the right to make changes without further notice to any products herein. Freescale makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including "typicals," must be validated for each customer application by customer's technical experts. Freescale does not convey any license under its patent rights nor the rights of others. Freescale sells products pursuant to standard terms and conditions of sale, which can be found at the following address: [freescale.com/SalesTermsandConditions](http://freescale.com/SalesTermsandConditions).

Freescale, Kinetis, and the Freescale logo are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. All other product or service names are the property of their respective owners.

©2014 Freescale Semiconductor, Inc.