

---

# USB Stack Host Reference Manual

Freescall Semiconductor, Inc.

Document Number:  
KSDK20USBHAPIRM  
Rev. 0  
Jan 2016





# Contents

## Chapter Overview

<b>1.1</b>	<b>USB Host Initialization flow</b> . . . . .	<b>3</b>
<b>1.2</b>	<b>USB Host peripheral attach/detach flow</b> . . . . .	<b>5</b>

## Chapter Definitions and structures

<b>2.1</b>	<b>Overview</b> . . . . .	<b>9</b>
<b>2.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>10</b>
2.2.1	struct usb_version_t . . . . .	10
<b>2.3</b>	<b>Typedef Documentation</b> . . . . .	<b>10</b>
2.3.1	usb_device_handle . . . . .	10
<b>2.4</b>	<b>Enumeration Type Documentation</b> . . . . .	<b>11</b>
2.4.1	usb_status_t . . . . .	11
2.4.2	usb_controller_index_t . . . . .	11

## Chapter USB Host driver

<b>3.1</b>	<b>Overview</b> . . . . .	<b>13</b>
<b>3.2</b>	<b>Data Structure Documentation</b> . . . . .	<b>17</b>
3.2.1	struct usb_host_ep_t . . . . .	17
3.2.2	struct usb_host_interface_t . . . . .	17
3.2.3	struct usb_host_configuration_t . . . . .	18
3.2.4	struct usb_host_pipe_t . . . . .	18
3.2.5	struct usb_host_transfer_t . . . . .	19
3.2.6	struct usb_host_pipe_init_t . . . . .	19
3.2.7	struct usb_host_cancel_param_t . . . . .	20
3.2.8	struct usb_host_device_instance_t . . . . .	20
3.2.9	struct usb_host_process_feature_param_t . . . . .	21
3.2.10	struct usb_host_process_descriptor_param_t . . . . .	21
3.2.11	struct usb_host_get_interface_param_t . . . . .	21
3.2.12	struct usb_host_get_status_param_t . . . . .	22
3.2.13	struct usb_host_set_interface_param_t . . . . .	22
3.2.14	struct usb_host_synch_frame_param_t . . . . .	22

# Contents

Section Number	Title	Page Number
3.2.15	struct usb_host_instance_t . . . . .	22
<b>3.3</b>	<b>Typedef Documentation . . . . .</b>	<b>23</b>
3.3.1	host_callback_t . . . . .	23
3.3.2	transfer_callback_t . . . . .	24
3.3.3	host_inner_transfer_callback_t . . . . .	24
<b>3.4</b>	<b>Enumeration Type Documentation . . . . .</b>	<b>24</b>
3.4.1	usb_host_event_t . . . . .	24
3.4.2	usb_host_dev_info_t . . . . .	25
3.4.3	usb_host_device_enumeration_status_t . . . . .	25
3.4.4	usb_host_interface_state_t . . . . .	25
3.4.5	usb_host_device_state_t . . . . .	26
3.4.6	usb_host_request_type_t . . . . .	26
<b>3.5</b>	<b>Function Documentation . . . . .</b>	<b>26</b>
3.5.1	USB_HostInit . . . . .	26
3.5.2	USB_HostDeinit . . . . .	27
3.5.3	USB_HostHelperGetPeripheralInformation . . . . .	27
3.5.4	USB_HostHelperParseAlternateSetting . . . . .	28
3.5.5	USB_HostRemoveDevice . . . . .	28
3.5.6	USB_HostKhciTaskFunction . . . . .	29
3.5.7	USB_HostEhciTaskFunction . . . . .	29
3.5.8	USB_HostKhciIsrFunction . . . . .	29
3.5.9	USB_HostEhciIsrFunction . . . . .	30
3.5.10	USB_HostOpenPipe . . . . .	30
3.5.11	USB_HostClosePipe . . . . .	30
3.5.12	USB_HostSend . . . . .	31
3.5.13	USB_HostSendSetup . . . . .	31
3.5.14	USB_HostRecv . . . . .	32
3.5.15	USB_HostCancelTransfer . . . . .	33
3.5.16	USB_HostMallocTransfer . . . . .	33
3.5.17	USB_HostFreeTransfer . . . . .	34
3.5.18	USB_HostRequestControl . . . . .	34
3.5.19	USB_HostOpenDeviceInterface . . . . .	35
3.5.20	USB_HostCloseDeviceInterface . . . . .	35
3.5.21	USB_HostGetVersion . . . . .	36
3.5.22	USB_HostAttachDevice . . . . .	36
3.5.23	USB_HostDetachDevice . . . . .	36
3.5.24	USB_HostDetachDeviceInternal . . . . .	37
3.5.25	USB_HostGetDeviceAttachState . . . . .	37
3.5.26	USB_HostValidateDevice . . . . .	37
<b>3.6</b>	<b>USB Host Controller driver . . . . .</b>	<b>39</b>
3.6.1	Overview . . . . .	39

# Contents

Section Number	Title	Page Number
3.6.2	Data Structure Documentation . . . . .	39
3.6.3	Enumeration Type Documentation . . . . .	40
3.6.4	USB Host Controller KHCI driver . . . . .	41
3.6.5	USB Host Controller EHCI driver . . . . .	48
<b>Chapter</b>	<b>USB Class driver</b>	
<b>4.1</b>	<b>Overview . . . . .</b>	<b>61</b>
<b>4.2</b>	<b>USB CDC Class driver . . . . .</b>	<b>62</b>
4.2.1	Overview . . . . .	62
4.2.2	USB Host CDC Initialization . . . . .	62
4.2.3	USB Host CDC De-initialization . . . . .	64
4.2.4	USB Host CDC Send data . . . . .	64
4.2.5	USB Host CDC Receive data . . . . .	64
4.2.6	Data Structure Documentation . . . . .	67
4.2.7	Function Documentation . . . . .	72
<b>4.3</b>	<b>USB HID Class driver . . . . .</b>	<b>82</b>
4.3.1	Overview . . . . .	82
4.3.2	USB Host HID Initialization . . . . .	82
4.3.3	USB Host HID Deinitialization . . . . .	83
4.3.4	USB Host HID Send data . . . . .	83
4.3.5	USB Host HID Receive data . . . . .	83
4.3.6	Data Structure Documentation . . . . .	85
4.3.7	Function Documentation . . . . .	87
<b>4.4</b>	<b>USB MSC Class driver . . . . .</b>	<b>97</b>
4.4.1	Overview . . . . .	97
4.4.2	USB Host MSC Initialization . . . . .	97
4.4.3	USB Host MSC Deinitialization . . . . .	98
4.4.4	USB Host MSC UFI Command . . . . .	98
4.4.5	Data Structure Documentation . . . . .	101
4.4.6	Function Documentation . . . . .	105
<b>4.5</b>	<b>USB AUDIO Class driver . . . . .</b>	<b>126</b>
4.5.1	Overview . . . . .	126
4.5.2	USB Host audio Initialization . . . . .	126
4.5.3	USB Host audio De-initialization . . . . .	128
4.5.4	USB Host audio Send data . . . . .	128
4.5.5	USB Host audio Receive data . . . . .	128
4.5.6	Data Structure Documentation . . . . .	131
4.5.7	Function Documentation . . . . .	136
<b>4.6</b>	<b>USB PHDC Class driver . . . . .</b>	<b>143</b>

# Contents

Section Number	Title	Page Number
4.6.1	Overview . . . . .	143
4.6.2	USB Host PHDC Initialization . . . . .	143
4.6.3	USB Host PHDC Deinitialization . . . . .	144
4.6.4	USB Host PHDC Send data . . . . .	144
4.6.5	USB Host PHDC Receive data . . . . .	144
4.6.6	Data Structure Documentation . . . . .	146
4.6.7	Function Documentation . . . . .	148

## Chapter USB OS Adapter

5.1	Overview . . . . .	155
5.2	Enumeration Type Documentation . . . . .	157
5.2.1	usb_osa_status_t . . . . .	157
5.2.2	usb_osa_event_mode_t . . . . .	157
5.3	Function Documentation . . . . .	157
5.3.1	USB_OsaMemoryAllocate . . . . .	157
5.3.2	USB_OsaMemoryFree . . . . .	157
5.3.3	USB_OsaEventCreate . . . . .	158
5.3.4	USB_OsaEventDestroy . . . . .	158
5.3.5	USB_OsaEventSet . . . . .	159
5.3.6	USB_OsaEventWait . . . . .	160
5.3.7	USB_OsaEventCheck . . . . .	161
5.3.8	USB_OsaEventClear . . . . .	162
5.3.9	USB_OsaSemCreate . . . . .	162
5.3.10	USB_OsaSemDestroy . . . . .	163
5.3.11	USB_OsaSemPost . . . . .	163
5.3.12	USB_OsaSemWait . . . . .	164
5.3.13	USB_OsaMutexCreate . . . . .	164
5.3.14	USB_OsaMutexDestroy . . . . .	165
5.3.15	USB_OsaMutexLock . . . . .	165
5.3.16	USB_OsaMutexUnlock . . . . .	166
5.3.17	USB_OsaMsgqCreate . . . . .	166
5.3.18	USB_OsaMsgqDestroy . . . . .	167
5.3.19	USB_OsaMsgqSend . . . . .	167
5.3.20	USB_OsaMsgqRecv . . . . .	168
5.3.21	USB_OsaMsgqCheck . . . . .	168

## Chapter 1

### Overview

The USB host stack is composed of USB class drivers. The USB class drivers include the USB common host driver and USB controller driver, which consists of the xHCI driver. Note that the xHCI represents either the EHCI or the KHCI, not the XHCI for USB 3.0.

To support different RTOSes with the same code base, the OSA is used inside the USB stack to wrap the differences between RTOSes. Note that the OSA is not supported for use in the USB application. Therefore, from the USB application's view point, the OSA is invisible.

The USB host stack must work with a dedicated application in which the following tasks should be done:

- Configure the USB clock
- Initialize/configure the USB host stack
- Choose the proper configuration when one peripheral is connected on callback event received and decide if one peripheral could be supported by this application
- Initialize class
- Choose the proper interface setting and configure the peripheral if needed
- Initialize the transfer request
- Handle the transfer result through the callback

The architecture and components of the USB host stack are shown below:

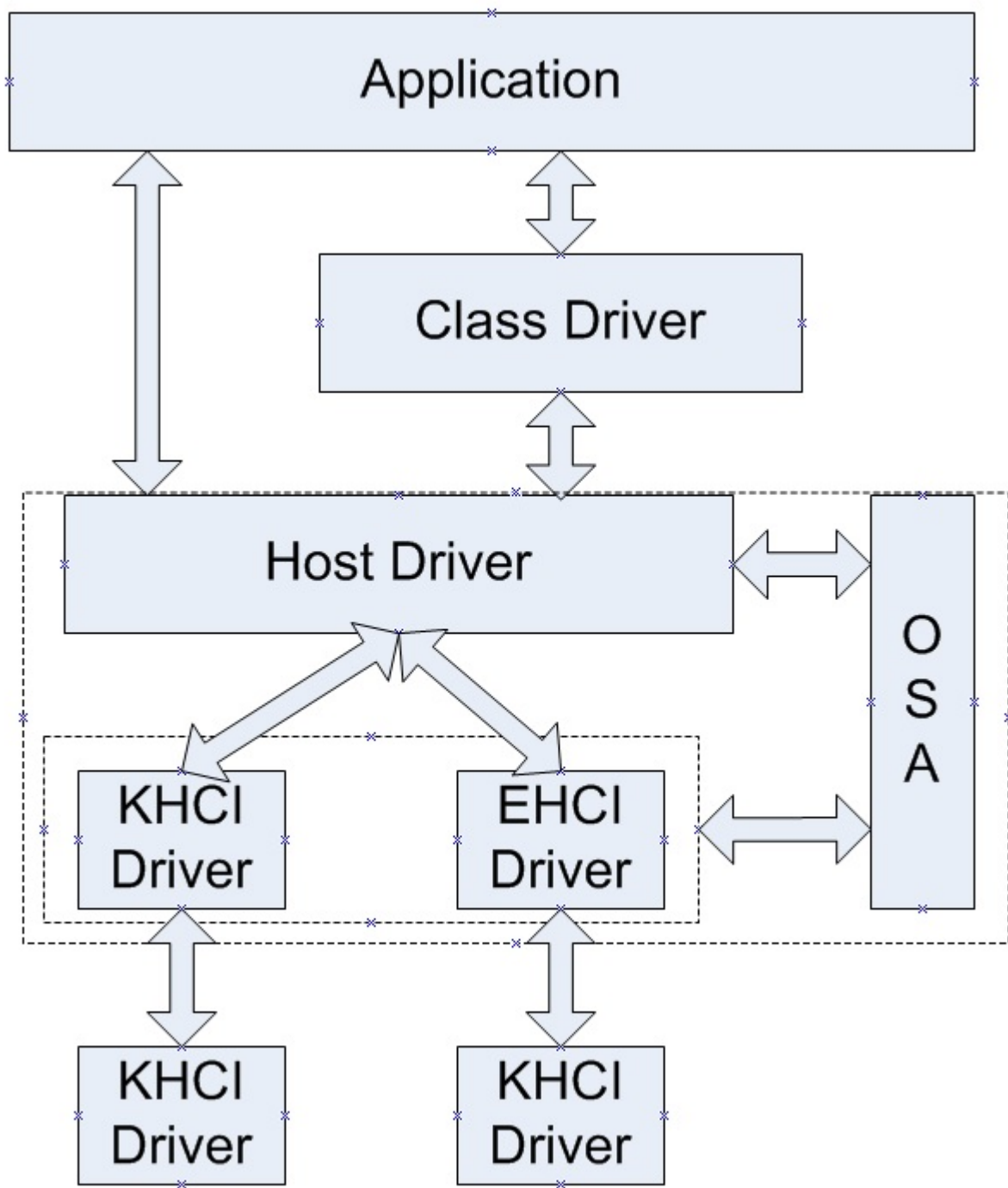


Figure 1: Host stack architecture

The interface between the KHCI/EHCI Driver and the Common Controller driver is internal and is simplified in this document.



## 1.1 USB Host Initialization flow

The host stack works as follows:

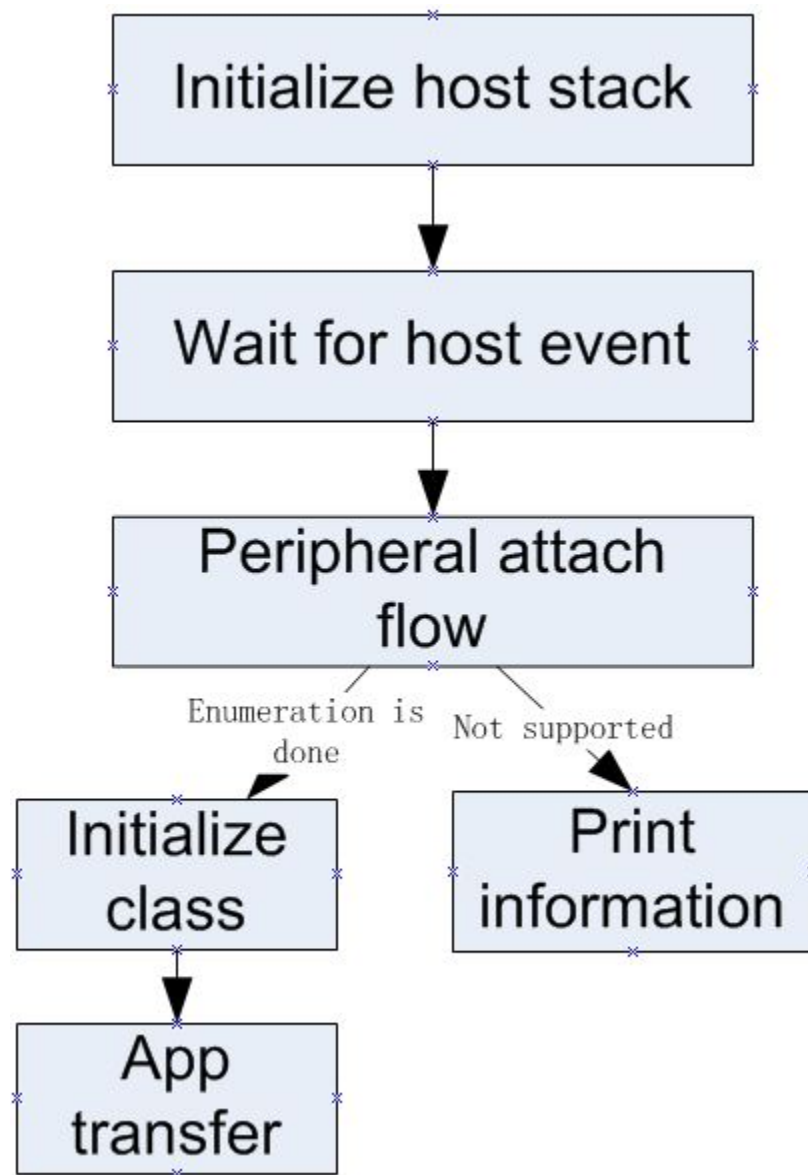


Figure 1.1.1: Host stack work flow

The host stack initialization work flow is as follows:

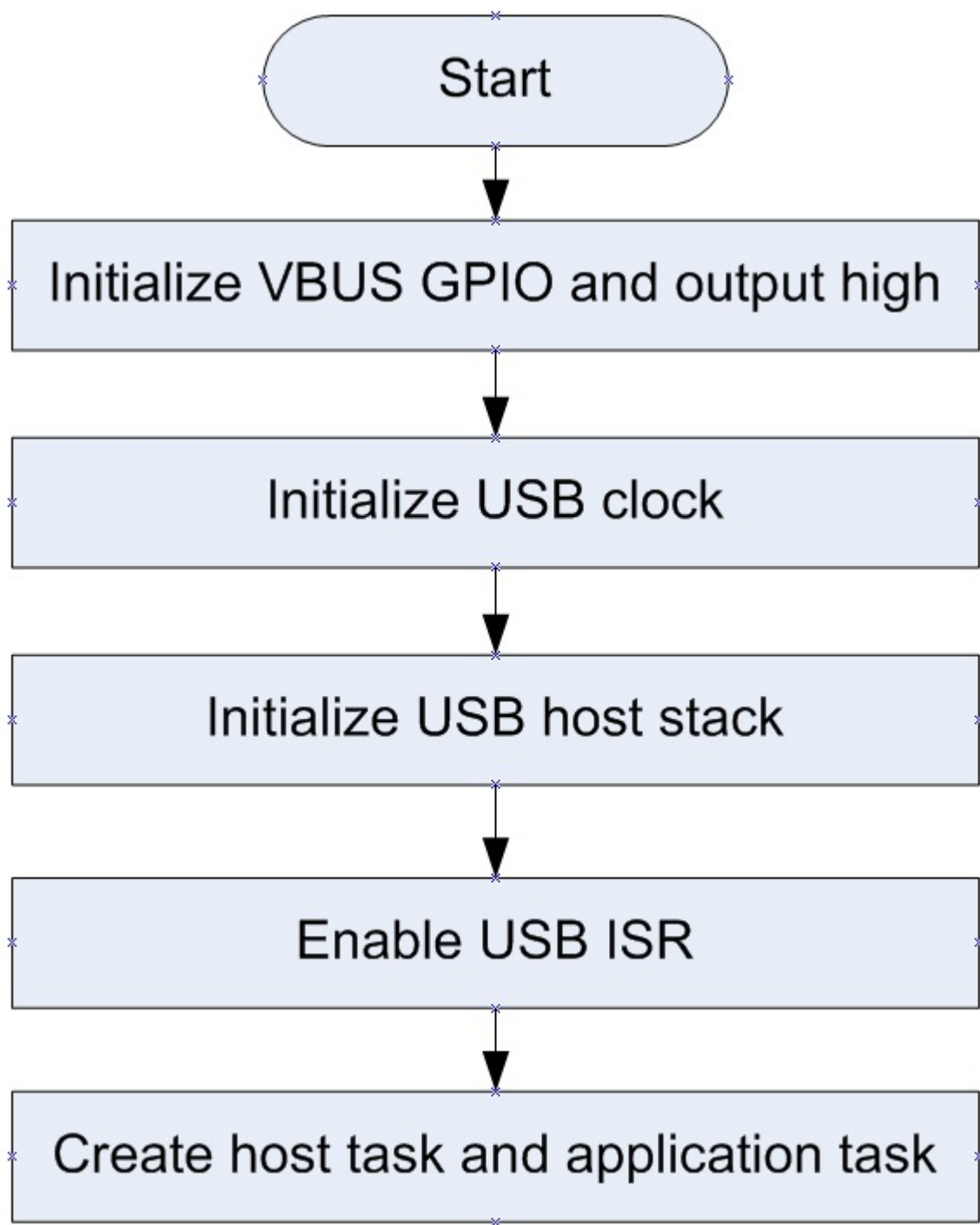


Figure 1.1.2: Host stack initialization flow

- If the platform uses a GPIO to control the VBUS, initialize the GPIO and the output high.
- Initialize the USB host clock.
- Call the `USB_HostInit` to initialize the USB host stack.
- Set the USB interrupt priority and enable the interrupt.
- Create the host task with the task API `USB_HostKhciTaskFunction` or the `USB_HostEhciTask-`

Function. Create an application task if necessary.

### 1.2 USB Host peripheral attach/detach flow

The peripheral attach/detach/unsupported event notifies the application through the callback function that it is registered by the USB\_HostInit.

The peripheral attach/detach flow is as follows:

## USB Host peripheral attach/detach flow

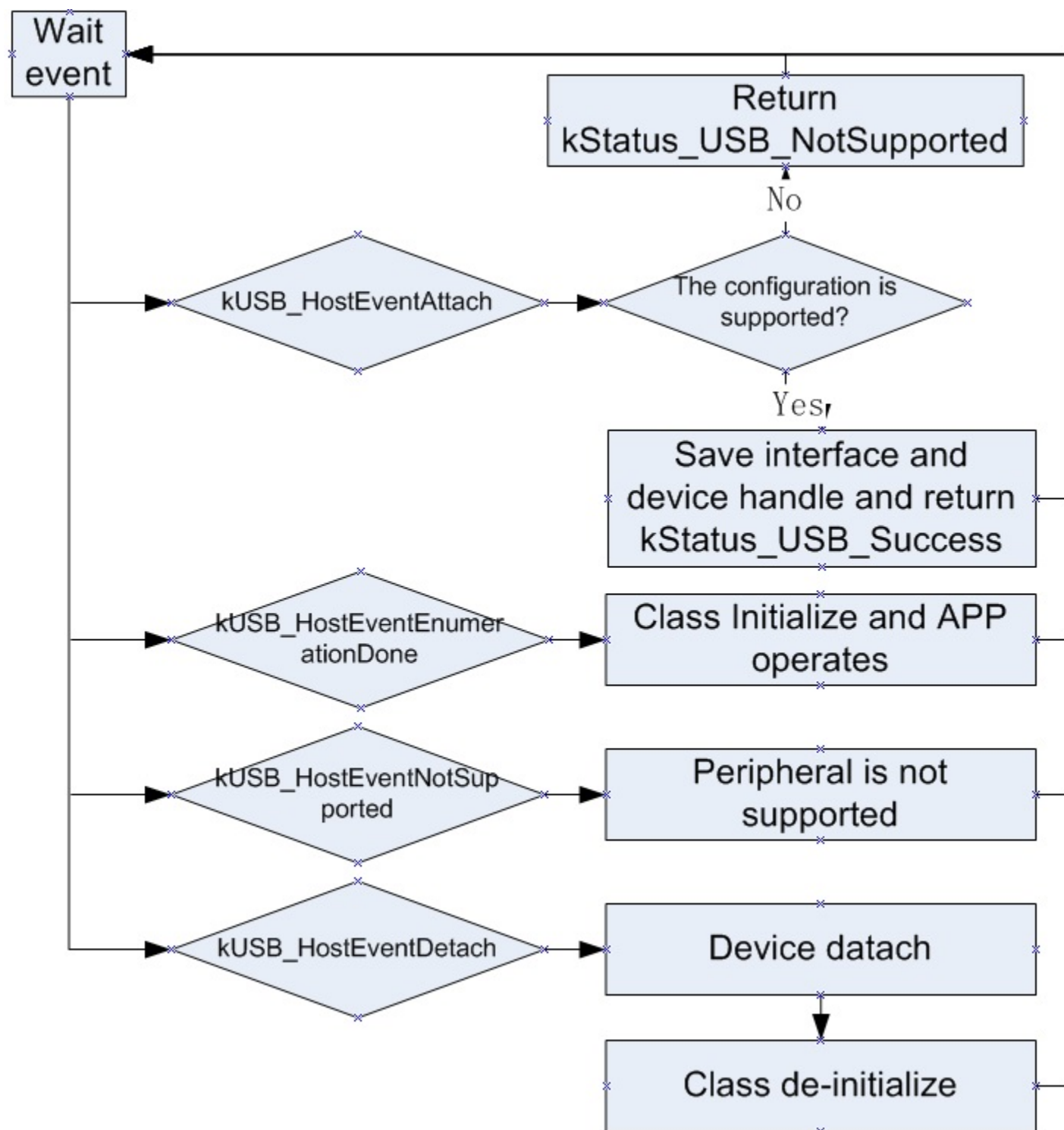


Figure 1.2.1: Host stack attach flow

The parameters of the callback contain the device handle, configuration handle, and event code. The key point is the configuration handle. All interface information within this configuration is included. The

application should make use of the information to decide if this configuration is supported. Note that, if the application returns `kStatus_USB_NotSupported`, the USB host stack checks the next configuration descriptor of the peripheral until the application returns the `kStatus_USB_Success` or all configuration descriptors are checked. If there is no supported configuration found in the peripheral descriptor, the `kUSB_HostEventNotSupported` event is notified to the application through a callback function registered by the `usb_host_init`.

There are four events in the callback. See the `host_event_t`:

- `kUSB_HostEventAttach` for attaching the peripheral
- `kUSB_HostEventDetach` for detaching the peripheral
- `kUSB_HostEventEnumerationDone` for a supported peripheral enumeration
- `kUSB_HostEventNotSupported` for detaching the peripheral

For example:

- Use case 1: The device has one configuration and is supported by the host application. The event flow is as follows:
  - (1) `kUSB_HostEventAttach` event; An application chooses the configuration and returns the `kStatus_USB_Success`.
  - (2) `kUSB_HostEventEnumerationDone` event; An application starts to initialize the class and run.
- Use case 2: The device has two configurations and is not supported by the host application. The event flow is as follows:
  - (1) `kUSB_HostEventAttach` event; An application chooses the first configuration and returns the `kStatus_USB_NotSupported`.
  - (2) `kUSB_HostEventAttach` event; An application chooses the second configuration and returns the `kStatus_USB_NotSupported`.
  - (3) `kUSB_HostEventNotSupported` event; An application prints the device not supported information.



## Chapter 2

### Definitions and structures

#### 2.1 Overview

This lists the common definitions and structures for the USB stack.

#### Data Structures

- struct [usb\\_version\\_t](#)  
*USB stack version fields. [More...](#)*

#### Macros

- #define [USB\\_STACK\\_VERSION\\_MAJOR](#) (0x01U)  
*Defines USB stack major version.*
- #define [USB\\_STACK\\_VERSION\\_MINOR](#) (0x00U)  
*Defines USB stack minor version.*
- #define [USB\\_STACK\\_VERSION\\_BUGFIX](#) (0x00U)  
*Defines USB stack bug fix version.*
- #define [USB\\_MAKE\\_VERSION](#)(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))  
*USB stack version definition.*

#### Typedefs

- typedef void \* [usb\\_host\\_handle](#)  
*USB host handle type define.*
- typedef void \* [usb\\_device\\_handle](#)  
*USB device handle type define.*
- typedef void \* [usb\\_otg\\_handle](#)  
*USB OTG handle type define.*

### Enumerations

- enum `usb_status_t` {  
    `kStatus_USB_Success` = 0x00U,  
    `kStatus_USB_Error`,  
    `kStatus_USB_Busy`,  
    `kStatus_USB_InvalidHandle`,  
    `kStatus_USB_InvalidParameter`,  
    `kStatus_USB_InvalidRequest`,  
    `kStatus_USB_ControllerNotFound`,  
    `kStatus_USB_InvalidControllerInterface`,  
    `kStatus_USB_NotSupported`,  
    `kStatus_USB_Retry`,  
    `kStatus_USB_TransferStall`,  
    `kStatus_USB_TransferFailed`,  
    `kStatus_USB_AllocFail`,  
    `kStatus_USB_LackSwapBuffer`,  
    `kStatus_USB_TransferCancel`,  
    `kStatus_USB_BandwidthFail`,  
    `kStatus_USB_MSDDStatusFail` }  
    *USB error code.*
- enum `usb_controller_index_t` {  
    `kUSB_ControllerKhci0` = 0U,  
    `kUSB_ControllerKhci1`,  
    `kUSB_ControllerEhci0`,  
    `kUSB_ControllerEhci1` }  
    *USB controller ID.*

## 2.2 Data Structure Documentation

### 2.2.1 struct `usb_version_t`

#### Data Fields

- `uint8_t` `major`  
    *Major.*
- `uint8_t` `minor`  
    *Minor.*
- `uint8_t` `bugfix`  
    *Bug fix.*

## 2.3 Typedef Documentation

### 2.3.1 typedef `void* usb_device_handle`

For device stack it is the whole device handle; for host stack it is the attached device instance handle



## 2.4 Enumeration Type Documentation

### 2.4.1 enum usb\_status\_t

Enumerator

*kStatus\_USB\_Success* Success.  
*kStatus\_USB\_Error* Failed.  
*kStatus\_USB\_Busy* Busy.  
*kStatus\_USB\_InvalidHandle* Invalid handle.  
*kStatus\_USB\_InvalidParameter* Invalid parameter.  
*kStatus\_USB\_InvalidRequest* Invalid request.  
*kStatus\_USB\_ControllerNotFound* Controller cannot be found.  
*kStatus\_USB\_InvalidControllerInterface* Invalid controller interface.  
*kStatus\_USB\_NotSupported* Configuration is not supported.  
*kStatus\_USB\_Retry* Enumeration get configuration retry.  
*kStatus\_USB\_TransferStall* Transfer stalled.  
*kStatus\_USB\_TransferFailed* Transfer failed.  
*kStatus\_USB\_AllocFail* Allocation failed.  
*kStatus\_USB\_LackSwapBuffer* Insufficient swap buffer for KHCI.  
*kStatus\_USB\_TransferCancel* The transfer cancelled.  
*kStatus\_USB\_BandwidthFail* Allocate bandwidth failed.  
*kStatus\_USB\_MSDStatusFail* For MSD, the CSW status means fail.

### 2.4.2 enum usb\_controller\_index\_t

Enumerator

*kUSB\_ControllerKhci0* KHCI 0U.  
*kUSB\_ControllerKhci1* KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.  
*kUSB\_ControllerEhci0* EHCI 0U.  
*kUSB\_ControllerEhci1* EHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.



## Chapter 3

# USB Host driver

### 3.1 Overview

The USB host driver implements USB host basic functions, such as the device enumeration, USB standard request, and send/receive data. It is the middle layer between the class driver and the controller driver. It provides the same APIs for different controller drivers.

/\*!

### Modules

- [USB Host Controller driver](#)

### Data Structures

- struct [usb\\_host\\_ep\\_t](#)  
*USB host endpoint information structure. [More...](#)*
- struct [usb\\_host\\_interface\\_t](#)  
*USB host interface information structure. [More...](#)*
- struct [usb\\_host\\_configuration\\_t](#)  
*USB host configuration information structure. [More...](#)*
- struct [usb\\_host\\_pipe\\_t](#)  
*USB host pipe common structure. [More...](#)*
- struct [usb\\_host\\_transfer\\_t](#)  
*USB host transfer structure. [More...](#)*
- struct [usb\\_host\\_pipe\\_init\\_t](#)  
*USB host pipe information structure for opening pipe. [More...](#)*
- struct [usb\\_host\\_cancel\\_param\\_t](#)  
*Cancel transfer parameter structure. [More...](#)*
- struct [usb\\_host\\_device\\_instance\\_t](#)  
*Device instance. [More...](#)*
- struct [usb\\_host\\_process\\_feature\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_CLEAR\_FEATURE and USB\_REQSET\_STANDARD\_SET\_FEATURE. [More...](#)*
- struct [usb\\_host\\_process\\_descriptor\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_GET\_DESCRIPTOR and USB\_REQSET\_STANDARD\_SET\_DESCRIPTOR. [More...](#)*
- struct [usb\\_host\\_get\\_interface\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_GET\_INTERFACE. [More...](#)*
- struct [usb\\_host\\_get\\_status\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_GET\_STATUS. [More...](#)*
- struct [usb\\_host\\_set\\_interface\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_SET\_INTERFACE. [More...](#)*
- struct [usb\\_host\\_synch\\_frame\\_param\\_t](#)  
*For USB\_REQSET\_STANDARD\_SYNCH\_FRAME. [More...](#)*

## Overview

- struct [usb\\_host\\_instance\\_t](#)  
*USB host instance structure. [More...](#)*

## Typedefs

- typedef void \* [usb\\_host\\_class\\_handle](#)  
*USB host class handle type define.*
- typedef void \* [usb\\_host\\_controller\\_handle](#)  
*USB host controller handle type define.*
- typedef void \* [usb\\_host\\_configuration\\_handle](#)  
*USB host configuration handle type define.*
- typedef void \* [usb\\_host\\_interface\\_handle](#)  
*USB host interface handle type define.*
- typedef void \* [usb\\_host\\_pipe\\_handle](#)  
*USB host pipe handle type define.*
- typedef [usb\\_status\\_t](#)(\* [host\\_callback\\_t](#))([usb\\_device\\_handle](#) deviceHandle, [usb\\_host\\_configuration\\_handle](#) configurationHandle, uint32\_t eventCode)  
*Host callback function typedef.*
- typedef void(\* [transfer\\_callback\\_t](#))(void \*param, uint8\_t \*data, uint32\_t dataLen, [usb\\_status\\_t](#) status)  
*Transfer callback function typedef.*
- typedef void(\* [host\\_inner\\_transfer\\_callback\\_t](#))(void \*param, struct \_usb\_host\_transfer \*transfer, [usb\\_status\\_t](#) status)  
*Host stack inner transfer callback function typedef.*

## Enumerations

- enum [usb\\_host\\_event\\_t](#) {  
    [kUSB\\_HostEventAttach](#) = 1U,  
    [kUSB\\_HostEventDetach](#),  
    [kUSB\\_HostEventEnumerationDone](#),  
    [kUSB\\_HostEventNotSupported](#) }  
*Event codes for device attach/detach.*
- enum [usb\\_host\\_dev\\_info\\_t](#) {  
    [kUSB\\_HostGetDeviceAddress](#) = 1U,  
    [kUSB\\_HostGetDeviceHubNumber](#),  
    [kUSB\\_HostGetDevicePortNumber](#),  
    [kUSB\\_HostGetDeviceSpeed](#),  
    [kUSB\\_HostGetDeviceHSHubNumber](#),  
    [kUSB\\_HostGetDeviceHSHubPort](#),  
    [kUSB\\_HostGetDeviceLevel](#),  
    [kUSB\\_HostGetHostHandle](#),  
    [kUSB\\_HostGetDeviceControlPipe](#),  
    [kUSB\\_HostGetDevicePID](#),  
    [kUSB\\_HostGetDeviceVID](#),  
    [kUSB\\_HostGetHubThinkTime](#) }  
*USB host device information code.*

- enum `usb_host_device_enumeration_status_t` {  
`kStatus_DEV_Notinit` = 0,  
`kStatus_DEV_Initial`,  
`kStatus_DEV_GetDes8`,  
`kStatus_DEV_SetAddress`,  
`kStatus_DEV_GetDes`,  
`kStatus_DEV_GetCfg9`,  
`kStatus_DEV_GetCfg`,  
`kStatus_DEV_SetCfg`,  
`kStatus_DEV_EnumDone`,  
`kStatus_DEV_AppUsed` }

*States of device instances enumeration.*

- enum `usb_host_interface_state_t` {  
`kStatus_interface_Attached` = 1,  
`kStatus_interface_Opened`,  
`kStatus_interface_Detached` }

*States of device's interface.*

- enum `usb_host_device_state_t` {  
`kStatus_device_Detached` = 0,  
`kStatus_device_Attached`,  
`kStatus_device_Released` }

*States of device.*

- enum `usb_host_request_type_t` {  
`kRequestDevice` = 1U,  
`kRequestInterface`,  
`kRequestEndpoint` }

*Request type.*

## Functions

- `usb_status_t USB_HostAttachDevice` (`usb_host_handle` hostHandle, `uint8_t` speed, `uint8_t` hubNumber, `uint8_t` portNumber, `uint8_t` level, `usb_device_handle` \*deviceHandle)  
*Calls this function when device attach.*
- `usb_status_t USB_HostDetachDevice` (`usb_host_handle` hostHandle, `uint8_t` hubNumber, `uint8_t` portNumber)  
*Call this function when device detaches.*
- `usb_status_t USB_HostDetachDeviceInternal` (`usb_host_handle` hostHandle, `usb_device_handle` deviceHandle)  
*Call this function when device detaches.*
- `uint8_t USB_HostGetDeviceAttachState` (`usb_device_handle` deviceHandle)  
*Gets the the device attach/detach state.*
- `usb_status_t USB_HostValidateDevice` (`usb_host_handle` hostHandle, `usb_device_handle` deviceHandle)  
*Determine whether the device is attached.*

## USB host APIs Part 1

The following APIs are recommended for application use.

## Overview

- `usb_status_t USB_HostInit` (`uint8_t controllerId`, `usb_host_handle *hostHandle`, `host_callback_t callbackFn`)  
*Initializes the USB host stack.*
- `usb_status_t USB_HostDeinit` (`usb_host_handle hostHandle`)  
*Deinitializes the USB host stack.*
- `usb_status_t USB_HostHelperGetPeripheralInformation` (`usb_device_handle deviceHandle`, `uint32_t infoCode`, `uint32_t *infoValue`)  
*Gets the device information.*
- `usb_status_t USB_HostHelperParseAlternateSetting` (`usb_host_interface_handle interfaceHandle`, `uint8_t alternateSetting`, `usb_host_interface_t *interface`)  
*Parses the alternate interface descriptor.*
- `usb_status_t USB_HostRemoveDevice` (`usb_host_handle hostHandle`, `usb_device_handle deviceHandle`)  
*Removes the attached device.*
- `void USB_HostKhciTaskFunction` (`void *hostHandle`)  
*KHCI task function.*
- `void USB_HostEhciTaskFunction` (`void *hostHandle`)  
*EHCI task function.*
- `void USB_HostKhciIsrFunction` (`void *hostHandle`)  
*Device KHCI ISR function.*
- `void USB_HostEhciIsrFunction` (`void *hostHandle`)  
*Device EHCI ISR function.*

## USB host APIs Part 2.

The following APIs are not recommended for application use.

They are mainly used in the class driver.

- `usb_status_t USB_HostOpenPipe` (`usb_host_handle hostHandle`, `usb_host_pipe_handle *pipeHandle`, `usb_host_pipe_init_t *pipeInit`)  
*Opens the USB host pipe.*
- `usb_status_t USB_HostClosePipe` (`usb_host_handle hostHandle`, `usb_host_pipe_handle pipeHandle`)  
*Closes the USB host pipe.*
- `usb_status_t USB_HostSend` (`usb_host_handle hostHandle`, `usb_host_pipe_handle pipeHandle`, `usb_host_transfer_t *transfer`)  
*Sends data to a pipe.*
- `usb_status_t USB_HostSendSetup` (`usb_host_handle hostHandle`, `usb_host_pipe_handle pipeHandle`, `usb_host_transfer_t *transfer`)  
*Sends a setup transfer to the pipe.*
- `usb_status_t USB_HostRecv` (`usb_host_handle hostHandle`, `usb_host_pipe_handle pipeHandle`, `usb_host_transfer_t *transfer`)  
*Receives the data from the pipe.*
- `usb_status_t USB_HostCancelTransfer` (`usb_host_handle hostHandle`, `usb_host_pipe_handle pipeHandle`, `usb_host_transfer_t *transfer`)  
*Cancel the pipe's transfers.*
- `usb_status_t USB_HostMallocTransfer` (`usb_host_handle hostHandle`, `usb_host_transfer_t **transfer`)  
*Allocates a transfer resource.*

- `usb_status_t USB_HostFreeTransfer (usb_host_handle hostHandle, usb_host_transfer_t *transfer)`  
*Frees a transfer resource.*
- `usb_status_t USB_HostRequestControl (usb_device_handle deviceHandle, uint8_t usbRequest, usb_host_transfer_t *transfer, void *param)`  
*Requests the USB standard request.*
- `usb_status_t USB_HostOpenDeviceInterface (usb_device_handle deviceHandle, usb_host_interface_handle interfaceHandle)`  
*Opens the interface.*
- `usb_status_t USB_HostCloseDeviceInterface (usb_device_handle deviceHandle, usb_host_interface_handle interfaceHandle)`  
*Closes an interface.*
- `void USB_HostGetVersion (uint32_t *version)`  
*Gets a host stack version function.*

## 3.2 Data Structure Documentation

### 3.2.1 struct usb\_host\_ep\_t

#### Data Fields

- `usb_descriptor_endpoint_t * epDesc`  
*Endpoint descriptor pointer.*
- `uint8_t * epExtension`  
*Endpoint extended descriptor pointer.*
- `uint16_t epExtensionLength`  
*Extended descriptor length.*

### 3.2.2 struct usb\_host\_interface\_t

#### Data Fields

- `usb_host_ep_t epList [USB_HOST_CONFIG_INTERFACE_MAX_EP]`  
*Endpoint array.*
- `usb_descriptor_interface_t * interfaceDesc`  
*Interface descriptor pointer.*
- `uint8_t * interfaceExtension`  
*Interface extended descriptor pointer.*
- `uint16_t interfaceExtensionLength`  
*Extended descriptor length.*
- `uint8_t interfaceIndex`  
*The interface index.*
- `uint8_t alternateSettingNumber`  
*The interface alternate setting value.*
- `uint8_t epCount`  
*Interface's endpoint number.*

### 3.2.3 struct usb\_host\_configuration\_t

#### Data Fields

- [usb\\_host\\_interface\\_t interfaceList](#) [USB\_HOST\_CONFIG\_CONFIGURATION\_MAX\_INTERF-  
ACE]  
*Interface array.*
- [usb\\_descriptor\\_configuration\\_t \\* configurationDesc](#)  
*Configuration descriptor pointer.*
- [uint8\\_t \\* configurationExtension](#)  
*Configuration extended descriptor pointer.*
- [uint16\\_t configurationExtensionLength](#)  
*Extended descriptor length.*
- [uint8\\_t interfaceCount](#)  
*The configuration's interface number.*

### 3.2.4 struct usb\_host\_pipe\_t

#### Data Fields

- [struct \\_usb\\_host\\_pipe \\* next](#)  
*Link the idle pipes.*
- [usb\\_device\\_handle deviceHandle](#)  
*This pipe's device's handle.*
- [uint16\\_t currentCount](#)  
*For KHCI transfer.*
- [uint16\\_t nakCount](#)  
*Maximum NAK count.*
- [uint16\\_t maxPacketSize](#)  
*Maximum packet size.*
- [uint16\\_t interval](#)  
*FS/LS: frame unit; HS: micro-frame unit.*
- [uint8\\_t open](#)  
*0 - closed, 1 - open*
- [uint8\\_t nextdata01](#)  
*Data toggle.*
- [uint8\\_t endpointAddress](#)  
*Endpoint address.*
- [uint8\\_t direction](#)  
*Pipe direction.*
- [uint8\\_t pipeType](#)  
*Pipe type, for example USB\_ENDPOINT\_BULK.*
- [uint8\\_t numberPerUframe](#)  
*Transaction number per micro-frame.*



### 3.2.5 struct usb\_host\_transfer\_t

#### Data Fields

- struct \_usb\_host\_transfer \* [next](#)  
*The next transfer structure.*
- uint8\_t \* [transferBuffer](#)  
*Transfer data buffer.*
- uint32\_t [transferLength](#)  
*Transfer data length.*
- uint32\_t [transferSofar](#)  
*Have transferred length.*
- [host\\_inner\\_transfer\\_callback\\_t](#) [callbackFn](#)  
*Transfer callback function.*
- void \* [callbackParam](#)  
*Transfer callback parameter.*
- [usb\\_host\\_pipe\\_t](#) \* [transferPipe](#)  
*Transfer pipe pointer.*
- [usb\\_setup\\_struct\\_t](#) [setupPacket](#)  
*Set up packet buffer.*
- uint8\_t [direction](#)  
*Transfer direction; it's values are USB\_OUT or USB\_IN.*
- uint8\_t [setupStatus](#)  
*Set up the transfer status.*

### 3.2.6 struct usb\_host\_pipe\_init\_t

#### Data Fields

- void \* [devInstance](#)  
*Device instance handle.*
- uint16\_t [nakCount](#)  
*Maximum NAK retry count.*
- uint16\_t [maxPacketSize](#)  
*Pipe's maximum packet size.*
- uint8\_t [interval](#)  
*Pipe's interval.*
- uint8\_t [endpointAddress](#)  
*Endpoint address.*
- uint8\_t [direction](#)  
*Endpoint direction.*
- uint8\_t [pipeType](#)  
*Endpoint type, the value is USB\_ENDPOINT\_INTERRUPT, USB\_ENDPOINT\_CONTROL, USB\_ENDPOINT\_ISOCHRONOUS, USB\_ENDPOINT\_BULK.*
- uint8\_t [numberPerUframe](#)  
*Transaction number for each micro-frame.*

## Data Structure Documentation

### 3.2.6.0.0.1 Field Documentation

#### 3.2.6.0.0.1.1 uint16\_t usb\_host\_pipe\_init\_t::nakCount

MUST be zero for interrupt

### 3.2.7 struct usb\_host\_cancel\_param\_t

#### Data Fields

- [usb\\_host\\_pipe\\_handle](#) pipeHandle  
*Cancelling pipe handle.*
- [usb\\_host\\_transfer\\_t](#) \* transfer  
*Cancelling transfer.*

### 3.2.8 struct usb\_host\_device\_instance\_t

#### Data Fields

- struct \_usb\_host\_device\_instance \* [next](#)  
*Next device, or NULL.*
- [usb\\_host\\_handle](#) hostHandle  
*Host handle.*
- [usb\\_host\\_configuration\\_t](#) configuration  
*Parsed configuration information for the device.*
- [usb\\_descriptor\\_device\\_t](#) deviceDescriptor  
*Standard device descriptor.*
- [usb\\_host\\_pipe\\_handle](#) controlPipe  
*Device's control pipe.*
- [uint8\\_t](#) \* [configurationDesc](#)  
*Configuration descriptor pointer.*
- [uint16\\_t](#) [configurationLen](#)  
*Configuration descriptor length.*
- [uint16\\_t](#) [configurationValue](#)  
*Configuration index.*
- [uint8\\_t](#) [interfaceStatus](#) [USB\_HOST\_CONFIG\_CONFIGURATION\_MAX\_INTERFACE]  
*Interfaces' status, see [usb\\_host\\_interface\\_state\\_t](#).*
- [uint8\\_t](#) [enumBuffer](#) [9]  
*Buffer for enumeration.*
- [uint8\\_t](#) [state](#)  
*Device state for enumeration.*
- [uint8\\_t](#) [enumRetries](#)  
*Re-enumeration when error in control transfer.*
- [uint8\\_t](#) [stallRetries](#)  
*Re-transfer when stall.*
- [uint8\\_t](#) [speed](#)  
*Device speed.*

- uint8\_t [allocatedAddress](#)  
*Temporary address for the device.*
- uint8\_t [setAddress](#)  
*The address has been set to the device successfully, 1 - 127.*
- uint8\_t [deviceAttachState](#)  
*See the `usb_host_device_state_t`.*

### 3.2.8.0.0.2 Field Documentation

#### 3.2.8.0.0.2.1 uint8\_t usb\_host\_device\_instance\_t::allocatedAddress

When set address request succeeds, setAddress is a value, 1 - 127

## 3.2.9 struct usb\_host\_process\_feature\_param\_t

### Data Fields

- uint8\_t [requestType](#)  
*See the `usb_host_request_type_t`.*
- uint8\_t [featureSelector](#)  
*Set/cleared feature.*
- uint8\_t [interfaceOrEndpoint](#)  
*Interface or end pointer.*

## 3.2.10 struct usb\_host\_process\_descriptor\_param\_t

### Data Fields

- uint8\_t [descriptorType](#)  
*See the `usb_spec.h`, such as the `USB_DESCRIPTOR_TYPE_DEVICE`.*
- uint8\_t [descriptorIndex](#)  
*The descriptor index is used to select a specific descriptor (only for configuration and string descriptors) when several descriptors of the same type are implemented in a device.*
- uint8\_t [languageId](#)  
*It specifies the language ID for string descriptors or is reset to zero for other descriptors.*
- uint8\_t \* [descriptorBuffer](#)  
*Buffer pointer.*
- uint16\_t [descriptorLength](#)  
*Buffer data length.*

## 3.2.11 struct usb\_host\_get\_interface\_param\_t

### Data Fields

- uint8\_t [interface](#)

## Data Structure Documentation

- *Interface number.*  
uint8\_t \* [alternateInterfaceBuffer](#)  
*Save the transfer result.*

### 3.2.12 struct usb\_host\_get\_status\_param\_t

#### Data Fields

- uint8\_t [requestType](#)  
*See the [usb\\_host\\_request\\_type\\_t](#).*
- uint8\_t [interfaceOrEndpoint](#)  
*Interface number or the end pointer number.*
- uint8\_t \* [statusBuffer](#)  
*Save the transfer result.*

### 3.2.13 struct usb\_host\_set\_interface\_param\_t

#### Data Fields

- uint8\_t [alternateSetting](#)  
*Alternate setting value.*
- uint8\_t [interface](#)  
*Interface number.*

### 3.2.14 struct usb\_host\_synch\_frame\_param\_t

#### Data Fields

- uint8\_t [endpoint](#)  
*Endpoint number.*
- uint8\_t \* [frameNumberBuffer](#)  
*Frame number data buffer.*

### 3.2.15 struct usb\_host\_instance\_t

#### Data Fields

- void \* [controllerHandle](#)  
*The low level controller handle.*
- [host\\_callback\\_t](#) [deviceCallback](#)  
*Device attach/detach callback.*
- [usb\\_osa\\_mutex\\_handle](#) [hostMutex](#)  
*Host layer mutex.*

- `usb_host_transfer_t transferList` [USB\_HOST\_CONFIG\_MAX\_TRANSFERS]  
*Transfer resource.*
- `usb_host_transfer_t * transferHead`  
*Idle transfer head.*
- `const`  
`usb_host_controller_interface_t * controllerTable`  
*KHCI/EHCI interface.*
- `void * deviceList`  
*Device list.*
- `uint8_t addressBitMap` [16]  
*Used for address allocation.*
- `uint8_t occupied`  
*0 - the instance is not occupied; 1 - the instance is occupied*
- `uint8_t controllerId`  
*The controller ID.*

### 3.2.15.0.0.3 Field Documentation

#### 3.2.15.0.0.3.1 `uint8_t usb_host_instance_t::addressBitMap[16]`

The first bit is the address 1, second bit is the address 2

## 3.3 Typedef Documentation

### 3.3.1 `typedef usb_status_t(* host_callback_t)(usb_device_handle deviceHandle, usb_host_configuration_handle configurationHandle, uint32_t eventCode)`

This callback function is used to notify application device attach/detach event. This callback pointer is passed when initializing the host.

Parameters

<i>deviceHandle</i>	The device handle, which indicates the attached device.
<i>configuration-Handle</i>	The configuration handle contains the attached device's configuration information.
<i>event_code</i>	The callback event code; See the enumeration <code>host_event_t</code> .

Returns

A USB error code or `kStatus_USB_Success`.

## Enumeration Type Documentation

Return values

<i>kStatus_USB_Success</i>	Application handles the attached device successfully.
<i>kStatus_USB_Not-Supported</i>	Application don't support the attached device.
<i>kStatus_USB_Error</i>	Application handles the attached device falsely.

### 3.3.2 typedef void(\* transfer\_callback\_t)(void \*param, uint8\_t \*data, uint32\_t dataLen, usb\_status\_t status)

This callback function is used to notify the upper layer the result of the transfer. This callback pointer is passed when calling the send/receive APIs.

Parameters

<i>param</i>	The parameter pointer, which is passed when calling the send/receive APIs.
<i>data</i>	The data buffer pointer.
<i>data_len</i>	The result data length.
<i>status</i>	A USB error code or kStatus_USB_Success.

### 3.3.3 typedef void(\* host\_inner\_transfer\_callback\_t)(void \*param, struct \_usb\_host\_transfer \*transfer, usb\_status\_t status)

This callback function is used to notify the upper layer the result of a transfer. This callback pointer is passed when initializing the structure [usb\\_host\\_transfer\\_t](#).

Parameters

<i>param</i>	The parameter pointer, which is passed when calling the send/receive APIs.
<i>transfer</i>	The transfer information; See the structure <a href="#">usb_host_transfer_t</a> .
<i>status</i>	A USB error code or kStatus_USB_Success.

## 3.4 Enumeration Type Documentation

### 3.4.1 enum usb\_host\_event\_t

Enumerator

***kUSB\_HostEventAttach*** Device is attached.  
***kUSB\_HostEventDetach*** Device is detached.

***kUSB\_HostEventEnumerationDone*** Device's enumeration is done and the device is supported.

***kUSB\_HostEventNotSupported*** Device's enumeration is done and the device is not supported.

### 3.4.2 enum usb\_host\_dev\_info\_t

Enumerator

***kUSB\_HostGetDeviceAddress*** Device's address.

***kUSB\_HostGetDeviceHubNumber*** Device's first hub address.

***kUSB\_HostGetDevicePortNumber*** Device's first hub port number.

***kUSB\_HostGetDeviceSpeed*** Device's speed.

***kUSB\_HostGetDeviceHSHubNumber*** Device's first high-speed hub address.

***kUSB\_HostGetDeviceHSHubPort*** Device's first high-speed hub number.

***kUSB\_HostGetDeviceLevel*** Device's hub level.

***kUSB\_HostGetHostHandle*** Device's host handle.

***kUSB\_HostGetDeviceControlPipe*** Device's control pipe handle.

***kUSB\_HostGetDevicePID*** Device's PID.

***kUSB\_HostGetDeviceVID*** Device's VID.

***kUSB\_HostGetHubThinkTime*** Device's hub total think time.

### 3.4.3 enum usb\_host\_device\_enumeration\_status\_t

Enumerator

***kStatus\_DEV\_Notinit*** Device is invalid.

***kStatus\_DEV\_Initial*** Device has been processed by host driver.

***kStatus\_DEV\_GetDes8*** Enumeration process: get 8 bytes' device descriptor.

***kStatus\_DEV\_SetAddress*** Enumeration process: set device address.

***kStatus\_DEV\_GetDes*** Enumeration process: get device descriptor.

***kStatus\_DEV\_GetCfg9*** Enumeration process: get 9 bytes' configuration descriptor.

***kStatus\_DEV\_GetCfg*** Enumeration process: get configuration descriptor.

***kStatus\_DEV\_SetCfg*** Enumeration process: set configuration.

***kStatus\_DEV\_EnumDone*** Enumeration is done.

***kStatus\_DEV\_AppUsed*** This device has been used by application.

### 3.4.4 enum usb\_host\_interface\_state\_t

Enumerator

***kStatus\_interface\_Attached*** Interface's default status.

***kStatus\_interface\_Opened*** Interface is used by application.

***kStatus\_interface\_Detached*** Interface is not used by application.

## Function Documentation

### 3.4.5 enum usb\_host\_device\_state\_t

Enumerator

*kStatus\_device\_Detached* Device is used by application.  
*kStatus\_device\_Attached* Device's default status.  
*kStatus\_device\_Released* Device's resource is released.

### 3.4.6 enum usb\_host\_request\_type\_t

Enumerator

*kRequestDevice* Control request object is device.  
*kRequestInterface* Control request object is interface.  
*kRequestEndpoint* Control request object is endpoint.

## 3.5 Function Documentation

### 3.5.1 usb\_status\_t USB\_HostInit ( uint8\_t controllerId, usb\_host\_handle \* hostHandle, host\_callback\_t callbackFn )

This function initializes the USB host module specified by the controllerId.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t.
out	<i>hostHandle</i>	Return the host handle.
in	<i>callbackFn</i>	Host callback function notifies device attach/detach.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Invalid_Handle</i>	The host_handle_ptr is a NULL pointer.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller according to the controller ID.



<i>kStatus_USB_AllocFail</i>	Allocation memory fail.
<i>kStatus_USB_Error</i>	Host mutex create fail; KHCI/EHCI mutex or KHCI/EHCI event create fail, or, KHCI/EHCI IP initialize fail.

### 3.5.2 **usb\_status\_t** USB\_HostDeinit ( **usb\_host\_handle** *hostHandle* )

This function deinitializes the USB host module specified by the hostHandle.

Parameters

in	<i>hostHandle</i>	the host handle.
----	-------------------	------------------

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle is a NULL pointer.
<i>kStatus_USB_Error</i>	Controller deinitialization fail.

### 3.5.3 **usb\_status\_t** USB\_HostHelperGetPeripheralInformation ( **usb\_device\_handle** *deviceHandle*, **uint32\_t** *infoCode*, **uint32\_t \*** *infoValue* )

This function gets the device information.

Parameters

in	<i>deviceHandle</i>	Removing device handle.
in	<i>infoCode</i>	See the enumeration host_dev_info_t.
out	<i>infoValue</i>	Return the information value.

Return values

<i>kStatus_USB_Success</i>	Close successfully.
<i>kStatus_USB_Invalid-Parameter</i>	The deviceHandle or info_value is a NULL pointer.

## Function Documentation

<i>kStatus_USB_Error</i>	The info_code is not the host_dev_info_t value.
--------------------------	-------------------------------------------------

### 3.5.4 **usb\_status\_t USB\_HostHelperParseAlternateSetting ( usb\_host\_interface\_handle *interfaceHandle*, uint8\_t *alternateSetting*, usb\_host\_interface\_t \* *interface* )**

This function parses the alternate interface descriptor and returns an interface information through the structure [usb\\_host\\_interface\\_t](#).

Parameters

in	<i>interface-Handle</i>	The whole interface handle.
in	<i>alternate-Setting</i>	Alternate setting value.
out	<i>interface</i>	Return interface information.

Return values

<i>kStatus_USB_Success</i>	Close successfully.
<i>kStatus_USB_Invalid-Handle</i>	The interfaceHandle is a NULL pointer.
<i>kStatus_USB_Invalid-Parameter</i>	The alternateSetting is 0.
<i>kStatus_USB_Error</i>	The interface descriptor is wrong.

### 3.5.5 **usb\_status\_t USB\_HostRemoveDevice ( usb\_host\_handle *hostHandle*, usb\_device\_handle *deviceHandle* )**

This function removes the attached device. This function should not be used all the time.

Parameters

in	<i>hostHandle</i>	The host handle.
----	-------------------	------------------

in	<i>deviceHandle</i>	Removing device handle.
----	---------------------	-------------------------

Return values

<i>kStatus_USB_Success</i>	Remove successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or deviceHandle is a NULL pointer.
<i>kStatus_USB_Invalid-Parameter</i>	The deviceHandle instance don't belong to hostHandle instance.

### 3.5.6 void USB\_HostKhciTaskFunction ( void \* *hostHandle* )

The function is used to handle the KHCI controller message. In the BM environment, this function should be called periodically in the main function. In the RTOS environment, this function should be used as a function entry to create a task.

Parameters

in	<i>hostHandle</i>	The host handle.
----	-------------------	------------------

### 3.5.7 void USB\_HostEhciTaskFunction ( void \* *hostHandle* )

The function is used to handle the EHCI controller message. In bare metal environment, this function should be called periodically in the main function. In the RTOS environment, this function should be used as a function entry to create a task.

Parameters

in	<i>hostHandle</i>	The host handle.
----	-------------------	------------------

### 3.5.8 void USB\_HostKhciIsrFunction ( void \* *hostHandle* )

The function is the KHCI interrupt service routine.

Parameters

---

## Function Documentation

in	<i>hostHandle</i>	The host handle.
----	-------------------	------------------

### 3.5.9 void USB\_HostEhciSrFunction ( void \* *hostHandle* )

The function is the EHCI interrupt service routine.

Parameters

in	<i>hostHandle</i>	The host handle.
----	-------------------	------------------

### 3.5.10 usb\_status\_t USB\_HostOpenPipe ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle \* *pipeHandle*, usb\_host\_pipe\_init\_t \* *pipeInit* )

This function opens a pipe according to the pipe\_init\_ptr parameter.

Parameters

in	<i>hostHandle</i>	The host handle.
out	<i>pipeHandle</i>	The pipe handle pointer used to return the pipe handle.
in	<i>pipeInit</i>	Used to initialize the pipe.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or pipe_handle_ptr is a NULL pointer.
<i>kStatus_USB_Error</i>	There is no idle pipe. Or, there is no idle QH for EHCI. Or, bandwidth allocate fail for EHCI.

### 3.5.11 usb\_status\_t USB\_HostClosePipe ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle *pipeHandle* )

This function closes a pipe and frees the related resources.

## Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>pipeHandle</i>	The closing pipe handle.

## Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or pipeHandle is a NULL pointer.

### 3.5.12 usb\_status\_t USB\_HostSend ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )

This function requests to send the transfer to the specified pipe.

## Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>pipeHandle</i>	The sending pipe handle.
in	<i>transfer</i>	The transfer information.

## Return values

<i>kStatus_USB_Success</i>	Send successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle, pipeHandle or transfer is a NULL pointer.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.
<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI.

### 3.5.13 usb\_status\_t USB\_HostSendSetup ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )

This function request to send the setup transfer to the specified pipe.

## Function Documentation

### Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>pipeHandle</i>	The sending pipe handle.
in	<i>transfer</i>	The transfer information.

### Return values

<i>kStatus_USB_Success</i>	Send successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle, pipeHandle or transfer is a NULL pointer.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.
<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI.

### 3.5.14 **usb\_status\_t USB\_HostRecv ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function requests to receive the transfer from the specified pipe.

### Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>pipeHandle</i>	The receiving pipe handle.
in	<i>transfer</i>	The transfer information.

### Return values

<i>kStatus_USB_Success</i>	Receive successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle, pipeHandle or transfer is a NULL pointer.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.

<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI.
--------------------------	-----------------------------------------

### 3.5.15 **usb\_status\_t USB\_HostCancelTransfer ( usb\_host\_handle *hostHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function cancels all pipe's transfers when the parameter transfer is NULL or cancels the transfers altogether.

Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>pipeHandle</i>	The receiving pipe handle.
in	<i>transfer</i>	The transfer information.

Return values

<i>kStatus_USB_Success</i>	Cancel successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or pipeHandle is a NULL pointer.

### 3.5.16 **usb\_status\_t USB\_HostMallocTransfer ( usb\_host\_handle *hostHandle*, usb\_host\_transfer\_t \*\* *transfer* )**

This function allocates a transfer. This transfer is used to pass data information to a low level stack.

Parameters

in	<i>hostHandle</i>	The host handle.
out	<i>transfer</i>	Return the transfer.

Return values

<i>kStatus_USB_Success</i>	Allocate successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or transfer is a NULL pointer.

## Function Documentation

<i>kStatus_USB_Error</i>	There is no idle transfer.
--------------------------	----------------------------

### 3.5.17 **usb\_status\_t USB\_HostFreeTransfer ( usb\_host\_handle *hostHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function frees a transfer. This transfer is used to pass data information to a low level stack.

#### Parameters

in	<i>hostHandle</i>	The host handle.
in	<i>transfer</i>	Release the transfer.

#### Return values

<i>kStatus_USB_Success</i>	Free successfully.
<i>kStatus_USB_Invalid-Handle</i>	The hostHandle or transfer is a NULL pointer.

### 3.5.18 **usb\_status\_t USB\_HostRequestControl ( usb\_device\_handle *deviceHandle*, uint8\_t *usbRequest*, usb\_host\_transfer\_t \* *transfer*, void \* *param* )**

This function sends the USB standard request packet.

#### Parameters

in	<i>deviceHandle</i>	The device handle for control transfer.
in	<i>usbRequest</i>	A USB standard request code. Se the usb_spec.h.
in	<i>transfer</i>	The used transfer.
in	<i>param</i>	The parameter structure is different for different request, see usb_host-framework.h.

#### Return values

<i>kStatus_USB_Success</i>	Send successfully.
----------------------------	--------------------



<i>kStatus_USB_Invalid-Handle</i>	The deviceHandle is a NULL pointer.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.
<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI, Or, the request is not standard request.

### 3.5.19 **usb\_status\_t USB\_HostOpenDeviceInterface ( usb\_device\_handle deviceHandle, usb\_host\_interface\_handle interfaceHandle )**

This function opens the interface. It is used to notify the host driver the interface is used by APP or class driver.

Parameters

in	<i>deviceHandle</i>	Removing device handle.
in	<i>interface-Handle</i>	Opening interface handle.

Return values

<i>kStatus_USB_Success</i>	Open successfully.
<i>kStatus_USB_Invalid-Handle</i>	The deviceHandle or interfaceHandle is a NULL pointer.

### 3.5.20 **usb\_status\_t USB\_HostCloseDeviceInterface ( usb\_device\_handle deviceHandle, usb\_host\_interface\_handle interfaceHandle )**

This function opens an interface. It is used to notify the host driver the interface is not used by APP or class driver.

Parameters

in	<i>deviceHandle</i>	Removing device handle.
in	<i>interface-Handle</i>	Opening interface handle.

## Function Documentation

Return values

<i>kStatus_USB_Success</i>	Close successfully.
<i>kStatus_USB_Invalid-Handle</i>	The deviceHandle is a NULL pointer.

### 3.5.21 void USB\_HostGetVersion ( uint32\_t \* *version* )

The function is used to get the host stack version.

Parameters

out	<i>version</i>	The version structure pointer to keep the host stack version.
-----	----------------	---------------------------------------------------------------

### 3.5.22 usb\_status\_t USB\_HostAttachDevice ( usb\_host\_handle *hostHandle*, uint8\_t *speed*, uint8\_t *hubNumber*, uint8\_t *portNumber*, uint8\_t *level*, usb\_device\_handle \* *deviceHandle* )

Parameters

<i>hostHandle</i>	Host instance handle.
<i>speed</i>	Device speed.
<i>hubNumber</i>	Device hub no. root device's hub no. is 0.
<i>portNumber</i>	Device port no. root device's port no. is 0.
<i>level</i>	Device level. root device's level is 1.
<i>deviceHandle</i>	Return device handle.

Returns

kStatus\_USB\_Success or error codes.

### 3.5.23 usb\_status\_t USB\_HostDetachDevice ( usb\_host\_handle *hostHandle*, uint8\_t *hubNumber*, uint8\_t *portNumber* )

## Parameters

<i>hostHandle</i>	Host instance handle.
<i>hubNumber</i>	Device hub no. root device's hub no. is 0.
<i>portNumber</i>	Device port no. root device's port no. is 0.

## Returns

kStatus\_USB\_Success or error codes.

### 3.5.24 **usb\_status\_t USB\_HostDetachDeviceInternal ( usb\_host\_handle *hostHandle*, usb\_device\_handle *deviceHandle* )**

## Parameters

<i>hostHandle</i>	Host instance handle.
<i>deviceHandle</i>	Device handle.

## Returns

kStatus\_USB\_Success or error codes.

### 3.5.25 **uint8\_t USB\_HostGetDeviceAttachState ( usb\_device\_handle *deviceHandle* )**

## Parameters

<i>deviceHandle</i>	Device handle.
---------------------	----------------

## Returns

0x01 - attached; 0x00 - detached.

### 3.5.26 **usb\_status\_t USB\_HostValidateDevice ( usb\_host\_handle *hostHandle*, usb\_device\_handle *deviceHandle* )**

## Function Documentation

### Parameters

<i>hostHandle</i>	Host instance pointer.
<i>deviceHandle</i>	Device handle.

### Returns

kStatus\_USB\_Success or error codes.

## 3.6 USB Host Controller driver

### 3.6.1 Overview

The USB Host controller driver implements the real send/receive function. Implementations are different for different controllers. There two supported controller drivers are KHCI and EHCI.

/\*!

### Modules

- [USB Host Controller EHCI driver](#)
- [USB Host Controller KHCI driver](#)

### Data Structures

- struct [usb\\_host\\_controller\\_interface\\_t](#)  
*USB host controller interface structure. [More...](#)*

### Enumerations

- enum [usb\\_host\\_controller\\_control\\_t](#) {  
    [kUSB\\_HostCancelTransfer](#) = 1U,  
    [kUSB\\_HostBusControl](#),  
    [kUSB\\_HostGetFrameNumber](#),  
    [kUSB\\_HostUpdateControlEndpointAddress](#),  
    [kUSB\\_HostUpdateControlPacketSize](#) }  
*USB host controller control code.*
- enum [usb\\_host\\_bus\\_control\\_t](#) {  
    [kUSB\\_HostBusReset](#) = 1U,  
    [kUSB\\_HostBusRestart](#),  
    [kUSB\\_HostBusEnableAttach](#),  
    [kUSB\\_HostBusDisableAttach](#) }  
*USB host controller bus control code.*

### 3.6.2 Data Structure Documentation

#### 3.6.2.1 struct usb\_host\_controller\_interface\_t

#### Data Fields

- [usb\\_status\\_t](#)(\* [controllerCreate](#) )(uint8\_t controllerId, [usb\\_host\\_handle](#) upperLayerHandle, [usb\\_host\\_controller\\_handle](#) \*controllerHandle)  
*Create a controller instance function prototype.*

## USB Host Controller driver

- `usb_status_t(* controllerDestory )(usb_host_controller_handle controllerHandle)`  
*Destroy a controller instance function prototype.*
- `usb_status_t(* controllerOpenPipe )(usb_host_controller_handle controllerHandle, usb_host_pipe_handle *pipeHandle, usb_host_pipe_init_t *pipeInit)`  
*Open a controller pipe function prototype.*
- `usb_status_t(* controllerClosePipe )(usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle)`  
*Close a controller pipe function prototype.*
- `usb_status_t(* controllerWritePipe )(usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle, usb_host_transfer_t *transfer)`  
*Write data to a pipe function prototype.*
- `usb_status_t(* controllerReadPipe )(usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle, usb_host_transfer_t *transfer)`  
*Read data from a pipe function prototype.*
- `usb_status_t(* controllerIoctl )(usb_host_controller_handle controllerHandle, uint32_t ioctlEvent, void *ioctlParam)`  
*Control a controller function prototype.*

### 3.6.3 Enumeration Type Documentation

#### 3.6.3.1 enum usb\_host\_controller\_control\_t

Enumerator

***kUSB\_HostCancelTransfer*** Cancel transfer code.  
***kUSB\_HostBusControl*** Bus control code.  
***kUSB\_HostGetFrameNumber*** Get frame number code.  
***kUSB\_HostUpdateControlEndpointAddress*** Update control endpoint address.  
***kUSB\_HostUpdateControlPacketSize*** Update control endpoint maximum packet size.

#### 3.6.3.2 enum usb\_host\_bus\_control\_t

Enumerator

***kUSB\_HostBusReset*** Reset bus.  
***kUSB\_HostBusRestart*** Restart bus.  
***kUSB\_HostBusEnableAttach*** Enable attach.  
***kUSB\_HostBusDisableAttach*** Disable attach.

### 3.6.4 USB Host Controller KHCI driver

#### 3.6.4.1 Overview

The KHCI host controller driver implements send/receive data through the KHCI IP.

/\*!

#### Data Structures

- struct `ptr_usb_host_khci_state_struct_t`  
*KHCI controller driver instance structure. [More...](#)*

#### Macros

- #define `KHCICFG_THSLD_DELAY` 0x65  
*The value programmed into the threshold register must reserve enough time to ensure the worst case transaction completes.*

#### USB host KHCI APIs

- `usb_status_t USB_HostKhciCreate` (`uint8_t` controllerId, `usb_host_handle` hostHandle, `usb_host_controller_handle` \*controllerHandle)  
*Creates the USB host KHCI instance.*
- `usb_status_t USB_HostKhciDestory` (`usb_host_controller_handle` controllerHandle)  
*Destroys the USB host KHCI instance.*
- `usb_status_t USB_HostKhciOpenPipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` \*pipeHandlePointer, `usb_host_pipe_init_t` \*pipeInitPointer)  
*Opens the USB host pipe.*
- `usb_status_t USB_HostKhciClosePipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle)  
*Closes the USB host pipe.*
- `usb_status_t USB_HostKhciWritePipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle, `usb_host_transfer_t` \*transfer)  
*Sends data to the pipe.*
- `usb_status_t USB_HostKhciReadpipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle, `usb_host_transfer_t` \*transfer)  
*Receives data from the pipe.*
- `usb_status_t USB_HostKciIoctl` (`usb_host_controller_handle` controllerHandle, `uint32_t` ioctlEvent, `void` \*ioctlParam)  
*Controls the KHCI.*

## USB Host Controller driver

### 3.6.4.2 Data Structure Documentation

#### 3.6.4.2.1 struct usb\_khci\_host\_state\_struct\_t

##### Data Fields

- volatile USB\_Type \* [usbRegBase](#)  
*The base address of the register.*
- void \* [hostHandle](#)  
*Related host handle.*
- [usb\\_host\\_pipe\\_t](#) \* [pipeDescriptorBasePointer](#)  
*Pipe descriptor base pointer.*
- [usb\\_osa\\_event\\_handle](#) [khciEventPointer](#)  
*KHCI event.*
- [usb\\_osa\\_mutex\\_handle](#) [khciMutex](#)  
*KHCI mutex.*
- [usb\\_host\\_transfer\\_t](#) \* [periodicListPointer](#)  
*KHCI periodic list pointer, which link is an interrupt and an ISO transfer request.*
- [usb\\_host\\_transfer\\_t](#) \* [asyncListPointer](#)  
*KHCI async list pointer, which link controls and bulk transfer request.*
- [khci\\_xfer\\_sts\\_t](#) [sXferSts](#)  
*KHCI transfer status structure for the DAM ALIGN workaround.*
- uint8\_t \* [khciSwapBufPointer](#)  
*KHCI swap buffer pointer for the DAM ALIGN workaround.*
- volatile uint32\_t [trState](#)  
*KHCI transfer state.*
- uint8\_t [asyncListActive](#)  
*KHCI async list is active.*
- uint8\_t [periodicListActive](#)  
*KHCI periodic list is active.*
- uint8\_t [rxBd](#)  
*RX buffer descriptor toggle bits.*
- uint8\_t [txBd](#)  
*TX buffer descriptor toggle bits.*
- uint8\_t [deviceSpeed](#)  
*Device speed.*
- int8\_t [deviceAttached](#)  
*Device attach/detach state.*

### 3.6.4.3 Macro Definition Documentation

#### 3.6.4.3.1 #define KHCICFG\_THSLD\_DELAY 0x65

In general, the worst case transaction is an IN token followed by a data packet from the target followed by the response from the host. The actual time required is a function of the maximum packet size on the bus. Set the KHCICFG\_THSLD\_DELAY to 0x65 can meet the worst case.



#### 3.6.4.4 Function Documentation

**3.6.4.4.1** `usb_status_t USB_HostKhciCreate ( uint8_t controllerId, usb_host_handle hostHandle,  
usb_host_controller_handle * controllerHandle )`

This function initializes the USB host KHCI controller driver.

## USB Host Controller driver

### Parameters

<i>controllerId</i>	The controller ID of the USB IP. See the enumeration <code>usb_controller_index_t</code> .
<i>hostHandle</i>	the host level handle.
<i>controller-Handle</i>	Return the controller instance handle.

### Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory failed.
<i>kStatus_USB_Error</i>	Host mutex create failed, KHCI mutex or KHCI event create failed. Or, KHCI IP initialize failed.

#### 3.6.4.4.2 `usb_status_t USB_HostKhciDestory ( usb_host_controller_handle controllerHandle )`

This function deinitializes the USB host KHCI controller driver.

### Parameters

<i>controller-Handle</i>	The controller handle.
--------------------------	------------------------

### Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
----------------------------	---------------------------------------

#### 3.6.4.4.3 `usb_status_t USB_HostKhciOpenPipe ( usb_host_controller_handle controllerHandle, usb_host_pipe_handle * pipeHandlePointer, usb_host_pipe_init_t * pipelnitPointer )`

This function opens a pipe according to the `pipe_init_ptr` parameter.

### Parameters

<i>controller-Handle</i>	The controller handle.
--------------------------	------------------------

<i>pipeHandle-Pointer</i>	The pipe handle pointer used to return the pipe handle.
<i>pipeInitPointer</i>	It is used to initialize the pipe.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Error</i>	There is no idle pipe.

#### 3.6.4.4.4 **usb\_status\_t USB\_HostKhciClosePipe ( usb\_host\_controller\_handle *controllerHandle*, usb\_host\_pipe\_handle *pipeHandle* )**

This function closes a pipe and frees the related resources.

Parameters

<i>controller-Handle</i>	The controller handle.
<i>pipeHandle</i>	The closing pipe handle.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
----------------------------	---------------------------------------

#### 3.6.4.4.5 **usb\_status\_t USB\_HostKhciWritePipe ( usb\_host\_controller\_handle *controllerHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function requests to send the transfer to the specified pipe.

Parameters

<i>controller-Handle</i>	The controller handle.
<i>pipeHandle</i>	The sending pipe handle.
<i>transfer</i>	The transfer information.

## USB Host Controller driver

Return values

<i>kStatus_USB_Success</i>	Send successful.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.

### 3.6.4.4.6 **usb\_status\_t** USB\_HostKhciReadpipe ( **usb\_host\_controller\_handle** *controllerHandle*, **usb\_host\_pipe\_handle** *pipeHandle*, **usb\_host\_transfer\_t** \* *transfer* )

This function requests to receive the transfer from the specified pipe.

Parameters

<i>controller-Handle</i>	The controller handle.
<i>pipeHandle</i>	The receiving pipe handle.
<i>transfer</i>	The transfer information.

Return values

<i>kStatus_USB_Success</i>	Receive successful.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.

### 3.6.4.4.7 **usb\_status\_t** USB\_HostKciloctl ( **usb\_host\_controller\_handle** *controllerHandle*, **uint32\_t** *ioctlEvent*, **void \*** *ioctlParam* )

This function controls the KHCI.

Parameters

<i>controller-Handle</i>	The controller handle.
<i>ioctlEvent</i>	See the enumeration <code>host_bus_control_t</code> .
<i>ioctlParam</i>	The control parameter.

Return values

<i>kStatus_USB_Success</i>	Cancel successful.
<i>kStatus_USB_Invalid-Handle</i>	The controllerHandle is a NULL pointer.

## USB Host Controller driver

### 3.6.5 USB Host Controller EHCI driver

#### 3.6.5.1 Overview

The EHCI host controller driver implements send/receive data through the EHCI IP.

#!/\*

#### Data Structures

- struct [usb\\_host\\_ehci\\_pipe\\_t](#)  
*EHCI pipe structure. [More...](#)*
- struct [usb\\_host\\_ehci\\_qh\\_t](#)  
*EHCI QH structure. [More...](#)*
- struct [usb\\_host\\_ehci\\_qtd\\_t](#)  
*EHCI QTD structure. [More...](#)*
- struct [usb\\_host\\_ehci\\_itd\\_t](#)  
*EHCI ITD structure. [More...](#)*
- struct [usb\\_host\\_ehci\\_sitd\\_t](#)  
*EHCI SITD structure. [More...](#)*
- struct [usb\\_host\\_ehci\\_iso\\_t](#)  
*EHCI ISO structure; An ISO pipe has an instance of this structure to keep the ISO pipe-specific information. [More...](#)*
- struct [usb\\_host\\_ehci\\_instance\\_t](#)  
*EHCI instance structure. [More...](#)*

#### Macros

- #define [USB\\_HOST\\_EHCI\\_ISO\\_NUMBER](#) USB\_HOST\_CONFIG\_EHCI\_MAX\_ITD  
*The maximum supported ISO pipe number.*
- #define [USB\\_HOST\\_EHCI\\_PORT\\_CONNECT\\_DEBOUNCE\\_DELAY](#) (101U)  
*Check the port connect state delay if the state is unstable.*
- #define [USB\\_HOST\\_EHCI\\_PORT\\_RESET\\_DELAY](#) (11U)  
*Delay for port reset.*
- #define [USB\\_HOST\\_EHCI\\_ISO\\_BOUNCE\\_FRAME\\_NUMBER](#) (2U)  
*The SITD inserts a frame interval for putting more SITD continuously.*
- #define [USB\\_HOST\\_EHCI\\_ISO\\_BOUNCE\\_UFRAME\\_NUMBER](#) (16U)  
*The ITD inserts a micro-frame interval for putting more ITD continuously.*
- #define [USB\\_HOST\\_EHCI\\_CONTROL\\_BULK\\_TIME\\_OUT\\_VALUE](#) (20U)  
*Control or bulk transaction timeout value (unit: 100 ms)*

#### Enumerations

- enum [host\\_ehci\\_device\\_state\\_t](#) {  
    [kEHCIDevicePhyAttached](#) = 1,  
    [kEHCIDeviceAttached](#),  
    [kEHCIDeviceDetached](#) }  
*EHCI state for device attachment/detachment.*

## USB host EHCI APIs

- `usb_status_t USB_HostEhciCreate` (`uint8_t` controllerId, `usb_host_handle` upperLayerHandle, `usb_host_controller_handle` \*controllerHandle)  
*Creates the USB host EHCI instance.*
- `usb_status_t USB_HostEhciDestory` (`usb_host_controller_handle` controllerHandle)  
*Destroys the USB host EHCI instance.*
- `usb_status_t USB_HostEhciOpenPipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` \*pipeHandle, `usb_host_pipe_init_t` \*pipeInit)  
*Opens the USB host pipe.*
- `usb_status_t USB_HostEhciClosePipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle)  
*Closes the USB host pipe.*
- `usb_status_t USB_HostEhciWritePipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle, `usb_host_transfer_t` \*transfer)  
*Sends data to the pipe.*
- `usb_status_t USB_HostEhciReadpipe` (`usb_host_controller_handle` controllerHandle, `usb_host_pipe_handle` pipeHandle, `usb_host_transfer_t` \*transfer)  
*Receives data from the pipe.*
- `usb_status_t USB_HostEhciIoctl` (`usb_host_controller_handle` controllerHandle, `uint32_t` ioctl-Event, `void` \*ioctlParam)  
*Controls the EHCI.*

## 3.6.5.2 Data Structure Documentation

### 3.6.5.2.1 struct usb\_host\_ehci\_pipe\_t

#### Data Fields

- `usb_host_pipe_t` pipeCommon  
*Common pipe information.*
- `void` \* ehciQh  
*Control/bulk/interrupt: QH; ISO: `usb_host_ehci_iso_t`.*
- `uint16_t` uframeInterval  
*Micro-frame interval value.*
- `uint16_t` startFrame  
*Bandwidth start frame: its value is from 0 to frame\_list.*
- `uint16_t` dataTime  
*Bandwidth time value:*
- `uint16_t` startSplitTime  
*Start splitting the bandwidth time value:*
- `uint16_t` completeSplitTime  
*Complete splitting the bandwidth time value:*
- `uint8_t` startUframe  
*Bandwidth start micro-frame: its value is from 0 to 7.*
- `uint8_t` uframeSmask  
*Start micro-frame.*
- `uint8_t` uframeCmask  
*Complete micro-frame.*

## USB Host Controller driver

### 3.6.5.2.1.1 Field Documentation

#### 3.6.5.2.1.1.1 uint16\_t usb\_host\_ehci\_pipe\_t::dataTime

- When the host works as HS: it's the data bandwidth value.
- When the host works as FS/LS:
  - For FS/LS device, it's the data bandwidth value when transferring the data by FS/LS.
  - For HS device, it's the data bandwidth value when transferring the data by HS.

#### 3.6.5.2.1.1.2 uint16\_t usb\_host\_ehci\_pipe\_t::startSplitTime

- When the host works as HS, it is the start split bandwidth value.

#### 3.6.5.2.1.1.3 uint16\_t usb\_host\_ehci\_pipe\_t::completeSplitTime

- When host works as HS, it is the complete split bandwidth value.

#### 3.6.5.2.1.1.4 uint8\_t usb\_host\_ehci\_pipe\_t::uframeSmask

- When host works as an HS:
  - For FS/LS device, it's the interrupt or ISO transfer start-split mask.
    - \* For HS device, it's the interrupt transfer start micro-frame mask.
- When host works as FS/LS, it's the interrupt and ISO start micro-frame mask

#### 3.6.5.2.1.1.5 uint8\_t usb\_host\_ehci\_pipe\_t::uframeCmask

- When host works as HS:
  - For FS/LS device, it's the interrupt or ISO transfer complete-split mask.

### 3.6.5.2.2 struct usb\_host\_ehci\_qh\_t

See the USB EHCI specification

#### Data Fields

- uint32\_t [horizontalLinkPointer](#)  
*QH specification filed, queue head a horizontal link pointer.*
- uint32\_t [staticEndpointStates](#) [2]  
*QH specification filed, static endpoint state and configuration information.*
- uint32\_t [currentQtdPointer](#)  
*QH specification filed, current qTD pointer.*
- uint32\_t [nextQtdPointer](#)  
*QH specification filed, next qTD pointer.*
- uint32\_t [alternateNextQtdPointer](#)  
*QH specification filed, alternate next qTD pointer.*
- uint32\_t [transferOverlayResults](#) [6]  
*QH specification filed, transfer overlay configuration and transfer results.*
- [usb\\_host\\_ehci\\_pipe\\_t](#) \* [ehciPipePointer](#)



- *EHCI pipe pointer.*  
[usb\\_host\\_transfer\\_t \\* ehciTransferHead](#)  
*Transfer list head on this QH.*
- [usb\\_host\\_transfer\\_t \\* ehciTransferTail](#)  
*Transfer list tail on this QH.*
- [uint16\\_t timeOutValue](#)  
*Its maximum value is USB\_HOST\_EHCI\_CONTROL\_BULK\_TIME\_OUT\_VALUE.*
- [uint16\\_t timeOutLabel](#)  
*It's used to judge the transfer timeout.*

### 3.6.5.2.2.1 Field Documentation

#### 3.6.5.2.2.1.1 [uint16\\_t usb\\_host\\_ehci\\_qh\\_t::timeOutValue](#)

When the value is zero, the transfer times out.

#### 3.6.5.2.2.1.2 [uint16\\_t usb\\_host\\_ehci\\_qh\\_t::timeOutLabel](#)

The EHCI driver maintain the value

### 3.6.5.2.3 [struct usb\\_host\\_ehci\\_qtd\\_t](#)

See the USB EHCI specification.

#### Data Fields

- [uint32\\_t nextQtdPointer](#)  
*QTD specification filed, the next QTD pointer.*
- [uint32\\_t alternateNextQtdPointer](#)  
*QTD specification filed, alternate next QTD pointer.*
- [uint32\\_t transferResults](#) [2]  
*QTD specification filed, transfer results fields.*
- [uint32\\_t bufferPointers](#) [4]  
*QTD specification filed, transfer buffer fields.*

### 3.6.5.2.4 [struct usb\\_host\\_ehci\\_itd\\_t](#)

See the USB EHCI specification.

#### Data Fields

- [uint32\\_t nextLinkPointer](#)  
*ITD specification filed, the next linker pointer.*
- [uint32\\_t transactions](#) [8]  
*ITD specification filed, transactions information.*
- [uint32\\_t bufferPointers](#) [7]  
*ITD specification filed, transfer buffer fields.*

## USB Host Controller driver

- struct \_usb\_host\_ehci\_itd \* [nextItldPointer](#)  
*Next ITD pointer.*
- uint32\_t [frameEntryIndex](#)  
*The ITD inserted frame value.*
- uint32\_t [reserved](#) [6]  
*Reserved fields for 32 bytes align.*

### 3.6.5.2.5 struct usb\_host\_ehci\_sitd\_t

See the USB EHCI specification.

#### Data Fields

- uint32\_t [nextLinkPointer](#)  
*SITD specification filed, the next linker pointer.*
- uint32\_t [endpointStates](#) [2]  
*SITD specification filed, endpoint configuration information.*
- uint32\_t [transferResults](#) [3]  
*SITD specification filed, transfer result fields.*
- uint32\_t [backPointer](#)  
*SITD specification filed, back pointer.*
- uint16\_t [frameEntryIndex](#)  
*The SITD inserted frame value.*
- uint8\_t [nextSitdIndex](#)  
*The next SITD index; Get the next SITD pointer through adding base address with the index.*
- uint8\_t [reserved](#)  
*Reserved fields for 32 bytes align.*

#### 3.6.5.2.5.1 Field Documentation

##### 3.6.5.2.5.1.1 uint8\_t usb\_host\_ehci\_sitd\_t::nextSitdIndex

0xFF means invalid.

### 3.6.5.2.6 struct usb\_host\_ehci\_iso\_t

#### Data Fields

- struct \_usb\_host\_ehci\_iso \* [next](#)  
*Next instance pointer.*
- [usb\\_host\\_pipe\\_t](#) \* [ehciPipePointer](#)  
*This ISO's EHCI pipe pointer.*
- [usb\\_host\\_transfer\\_t](#) \* [ehciTransferHead](#)  
*Transfer list head on this ISO pipe.*
- [usb\\_host\\_transfer\\_t](#) \* [ehciTransferTail](#)  
*Transfer list head on this ISO pipe.*
- uint16\_t [lastLinkFrame](#)  
*It means that the inserted frame for ISO ITD/SITD.*

### 3.6.5.2.6.1 Field Documentation

#### 3.6.5.2.6.1.1 uint16\_t usb\_host\_ehci\_iso\_t::lastLinkFrame

0xFFFF is invalid. For ITD, it is a micro-frame value. For SITD, it is a frame value

### 3.6.5.2.7 struct usb\_host\_ehci\_instance\_t

#### Data Fields

- [usb\\_host\\_handle](#) hostHandle  
*Related host handle.*
- uint32\_t \* [ehciUnitBase](#)  
*Keep the QH/QTD/ITD/SITD buffer pointer for release.*
- [usb\\_host\\_ehci\\_qh\\_t](#) \* [ehciQhList](#)  
*Idle QH list pointer.*
- [usb\\_host\\_ehci\\_qtd\\_t](#) \* [ehciQtdHead](#)  
*Idle QTD list pointer head.*
- [usb\\_host\\_ehci\\_qtd\\_t](#) \* [ehciQtdTail](#)  
*Idle QTD list pointer tail (recently used qTD will be used at last)*
- [usb\\_host\\_ehci\\_itd\\_t](#) \* [ehciItedList](#)  
*Idle ITD list pointer.*
- [usb\\_host\\_ehci\\_sitd\\_t](#) \* [ehciSitdIndexBase](#)  
*SITD buffer's start pointer.*
- [usb\\_host\\_ehci\\_sitd\\_t](#) \* [ehciSitdList](#)  
*Idle SITD list pointer.*
- [usb\\_host\\_ehci\\_iso\\_t](#) \* [ehciIsoList](#)  
*Idle ISO list pointer.*
- USBHS\_Type \* [ehciIpBase](#)  
*EHCI IP base address.*
- [usb\\_host\\_ehci\\_qh\\_t](#) \* [shedFirstQh](#)  
*First async QH.*
- [usb\\_host\\_ehci\\_pipe\\_t](#) \* [ehciPipeIndexBase](#)  
*Pipe buffer's start pointer.*
- [usb\\_host\\_ehci\\_pipe\\_t](#) \* [ehciPipeList](#)  
*Idle pipe list pointer.*
- [usb\\_host\\_ehci\\_pipe\\_t](#) \* [ehciRunningPipeList](#)  
*Running pipe list pointer.*
- [usb\\_osa\\_mutex\\_handle](#) [ehciMutex](#)  
*EHCI mutex.*
- [usb\\_osa\\_event\\_handle](#) [taskEventHandle](#)  
*EHCI task event.*
- uint8\_t [controllerId](#)  
*EHCI controller ID.*
- uint8\_t [deviceAttached](#)  
*Device attach/detach state, see [host\\_ehci\\_device\\_state\\_t](#).*
- uint8\_t [firstDeviceSpeed](#)  
*The first device's speed, the controller's work speed.*
- uint8\_t [ehciItidNumber](#)  
*Idle ITD number.*

## USB Host Controller driver

- uint8\_t ehciSitdNumber  
*Idle SITD number.*
- uint8\_t ehciQtdNumber  
*Idle QTD number.*

### 3.6.5.3 Macro Definition Documentation

#### 3.6.5.3.1 #define USB\_HOST\_EHCI\_ISO\_BOUNCE\_FRAME\_NUMBER (2U)

There is an interval when an application sends two FS/LS ISO transfers. When the interval is less than the macro, the two transfers are continuous in the frame list. Otherwise, the two transfers are not continuous. For example:

- Use case 1: when inserting the SITD first, the inserted frame = the current frame value + this MACRO value.
- Use case 2: when inserting SITD is not first, choose between the last inserted frame value and the current frame value according to the following criteria: If the interval is less than the MACRO value, the new SITD is continuous with the last SITD. If not, the new SITD inserting frame = the current frame value + this MACRO value.

#### 3.6.5.3.2 #define USB\_HOST\_EHCI\_ISO\_BOUNCE\_UFRAME\_NUMBER (16U)

There is an interval when an application sends two HS ISO transfers. When the interval is less than the macro, the two transfers are continuous in the frame list. Otherwise, the two transfers are not continuous. For example:

- Use case 1: when inserting ITD first, the inserted micro-frame = the current micro-frame value + this MACRO value.
- Use case 2: when inserting ITD is not first, choose between the last inserted micro-frame value and the current micro-frame value according to the following criteria: If the interval is less than this MACRO value, the new ITD is continuous with the last ITD. If not, the new ITD inserting micro-frame = the current micro-frame value + this MACRO value.

### 3.6.5.4 Enumeration Type Documentation

#### 3.6.5.4.1 enum host\_ehci\_device\_state\_t

Enumerator

***kEHCIDevicePhyAttached*** Device is physically attached.  
***kEHCIDeviceAttached*** Device is attached and initialized.  
***kEHCIDeviceDetached*** Device is detached and de-initialized.

### 3.6.5.5 Function Documentation

#### 3.6.5.5.1 `usb_status_t USB_HostEhciCreate ( uint8_t controllerId, usb_host_handle upperLayerHandle, usb_host_controller_handle * controllerHandle )`

This function initializes the USB host EHCI controller driver.

## USB Host Controller driver

### Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. Please refer to the enumeration <code>usb_controller_index_t</code> .
in	<i>upperLayer-Handle</i>	the host level handle.
out	<i>controller-Handle</i>	return the controller instance handle.

### Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_AllocFail</i>	allocate memory fail.
<i>kStatus_USB_Error</i>	host mutex create fail, KHCI/EHCI mutex or KHCI/EHCI event create fail. Or, KHCI/EHCI IP initialize fail.

#### 3.6.5.5.2 `usb_status_t USB_HostEhciDestory ( usb_host_controller_handle controllerHandle )`

This function de-initializes the USB host EHCI controller driver.

### Parameters

in	<i>controller-Handle</i>	The controller handle.
----	--------------------------	------------------------

### Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
----------------------------	---------------------------------------

#### 3.6.5.5.3 `usb_status_t USB_HostEhciOpenPipe ( usb_host_controller_handle controllerHandle, usb_host_pipe_handle * pipeHandle, usb_host_pipe_init_t * pipelnit )`

This function opens a pipe according to the `pipe_init_ptr` parameter.

### Parameters

in	<i>controller-Handle</i>	The controller handle.
----	--------------------------	------------------------

out	<i>pipeHandle</i>	The pipe handle pointer, it is used to return the pipe handle.
in	<i>pipeInit</i>	It is used to initialize the pipe.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
<i>kStatus_USB_Error</i>	there is no idle pipe. Or, there is no idle QH for EHCI. Or, bandwidth allocate fail for EHCI.

#### 3.6.5.5.4 **usb\_status\_t USB\_HostEhciClosePipe ( usb\_host\_controller\_handle *controllerHandle*, usb\_host\_pipe\_handle *pipeHandle* )**

This function closes a pipe and releases related resources.

Parameters

in	<i>controller-Handle</i>	the controller handle.
in	<i>pipeHandle</i>	the closing pipe handle.

Return values

<i>kStatus_USB_Success</i>	The host is initialized successfully.
----------------------------	---------------------------------------

#### 3.6.5.5.5 **usb\_status\_t USB\_HostEhciWritePipe ( usb\_host\_controller\_handle *controllerHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function requests to send the transfer to the specified pipe.

Parameters

in	<i>controller-Handle</i>	The controller handle.
in	<i>pipeHandle</i>	The sending pipe handle.
in	<i>transfer</i>	The transfer information.

## USB Host Controller driver

Return values

<i>kStatus_USB_Success</i>	Sent successfully.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.
<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI.

### 3.6.5.5.6 **usb\_status\_t USB\_HostEhciReadpipe ( usb\_host\_controller\_handle *controllerHandle*, usb\_host\_pipe\_handle *pipeHandle*, usb\_host\_transfer\_t \* *transfer* )**

This function requests to receive the transfer from the specified pipe.

Parameters

in	<i>controller-Handle</i>	The controller handle.
in	<i>pipeHandle</i>	The receiving pipe handle.
in	<i>transfer</i>	The transfer information.

Return values

<i>kStatus_USB_Success</i>	Send successfully.
<i>kStatus_USB_LackSwap-Buffer</i>	There is no swap buffer for KHCI.
<i>kStatus_USB_Error</i>	There is no idle QTD/ITD/SITD for EHCI.

### 3.6.5.5.7 **usb\_status\_t USB\_HostEhciIoctl ( usb\_host\_controller\_handle *controllerHandle*, uint32\_t *ioctlEvent*, void \* *ioctlParam* )**

This function controls the EHCI.

Parameters

in	<i>controller-Handle</i>	The controller handle.
----	--------------------------	------------------------



in	<i>ioctlEvent</i>	See enumeration host_bus_control_t.
in	<i>ioctlParam</i>	The control parameter.

#### Return values

<i>kStatus_USB_Success</i>	Cancel successfully.
<i>kStatus_USB_Invalid-Handle</i>	The controllerHandle is a NULL pointer.





## Chapter 4

### USB Class driver

#### 4.1 Overview

##### Modules

- [USB AUDIO Class driver](#)
- [USB CDC Class driver](#)
- [USB HID Class driver](#)
- [USB MSC Class driver](#)
- [USB PHDC Class driver](#)

## **4.2 USB CDC Class driver**

### **4.2.1 Overview**

The Communication Class defines mechanisms for a device and host to identify which existing protocols to use. It also defines an architecture that is capable of supporting any communications devices. The communications device class and associated subclass specifications, such as ISDN and PSTN, provides information to guide implementers in using the USB logical structures for communications device. This section uses the PSTN as the subclass and describes the programming interface of the USB HOST CDC class driver. The USB HOST HID class driver handles the specific control requests for CDC class and transfers data to and from the device through the bulk pipe.

### **4.2.2 USB Host CDC Initialization**

When the CDC device is attached, the CDC initialization flow is as follows:

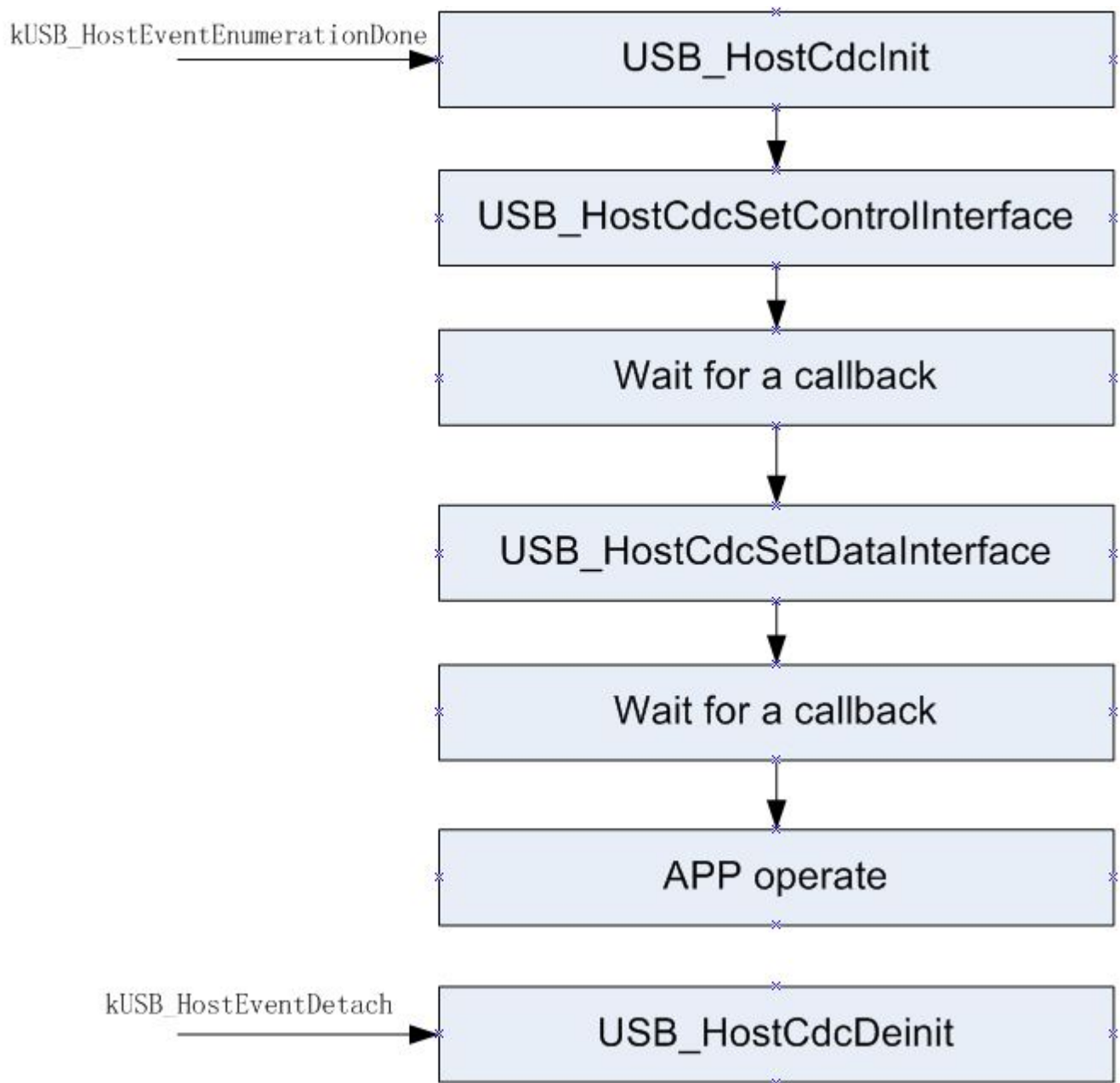


Figure 4.2.1: Host Cdc Initialization

The above figure describes the following steps:

- Call the `USB_HostCdcInit` to initialize the CDC class instance `#usb_host_cdc_instance_struct_t`. Save the class handle pointer into the `#usb_host_cdc_instance_struct_t`. The driver uses an instantiation of the `usb_host_cdc_instance_struct_t` structure to maintain the current state of a CDC instance module driver. This structure holds the USB host handle, the USB device handle and keeps track of transfer information, alternate setting, pipes, and interfaces that are enumerated for the attached CDC device.

## USB CDC Class driver

- Call the `USB_HostCdcSetControlInterface` to set the CDC class control interface, which opens the interface's pipes.
- Wait for the last step operation callback.
- Call the `USB_HostCdcSetDataInterface` to set the CDC class data interface, which opens the interface's pipes.
- Wait for the last step operation callback.
- Call the `USB_HostCdcDataRecv` to receive data from device, or call `USB_HostCdcDataSend` to send data to the device.
- Wait for the last step operation callback.
- Process data and receive or send again.

### 4.2.3 USB Host CDC De-initialization

An application can call the `USB_HostCdcDeinit` to deinitialize the CDC. This function cancels the transfer, closes the pipe, and frees the HID class instance.

There are two cases to call this function:

- The CDC device is detached and this function is called to free the resource.
- An application calls this function and calls the `USB_HostCdcInit` to re-initialize the CDC class.

### 4.2.4 USB Host CDC Send data

Provides the buffer pointer, the buffer length, the callback function, and the callback parameter and call `USB_HostCdcDataSend` to start asynchronous sending. Then the callback function is called with one transfer status parameter when the transfer succeeds or fails.

### 4.2.5 USB Host CDC Receive data

Provides the buffer pointer, the buffer length, the callback function, and the callback parameter and calls `USB_HostCdcDataRecv` to start asynchronous receiving. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

## Data Structures

- struct `usb_host_cdc_line_coding_struct_t`  
*CDC GetLineCoding structure according to the 6.3 in PSTN specification. [More...](#)*
- struct `usb_host_cdc_control_line_state_struct_t`  
*CDC GetLineCoding structure according to the 6.3 in PSTN specification. [More...](#)*
- struct `usb_host_cdc_acm_state_struct_t`  
*CDC SerialState structure according to the 6.5.4 in PSTN specification. [More...](#)*
- struct `usb_host_cdc_head_function_desc_struct_t`  
*CDC Header Functional Descriptor structure according to the 5.2.3 in CDC specification. [More...](#)*

- struct `usb_host_cdc_call_manage_desc_struct_t`  
CDC Call Management Functional Descriptor structure according to the 5.3.1 in PSTN specification. [More...](#)
- struct `usb_host_cdc_abstract_control_desc_struct_t`  
CDC Abstract Control Management Functional Descriptor structure according to the 5.3.2 in PSTN specification. [More...](#)
- struct `usb_host_cdc_direct_line_desc_struct_t`  
CDC Direct Line Management Functional Descriptor structure according to the 5.3.3 in PSTN specification. [More...](#)
- struct `usb_host_cdc_telephone_ringer_desc_struct_t`  
CDC Telephone Ringer Functional Descriptor structure according to the 5.3.4 in PSTN specification. [More...](#)
- struct `usb_host_cdc_tcLsr_desc_struct_t`  
CDC Telephone Call and Line State Reporting Capabilities Descriptor structure according to the 5.3.6 in PSTN specification. [More...](#)
- struct `usb_host_cdc_union_interface_desc_struct_t`  
CDC Header Functional Descriptor structure according to the 5.2.3 in CDC specification. [More...](#)
- struct `usb_host_cdc_tom_desc_struct_t`  
CDC Telephone Operational Modes Functional Descriptor structure according to the 5.3.5 in PSTN specification. [More...](#)
- struct `usb_host_cdc_common_desc_struct_t`  
CDC common Functional Descriptor structure. [More...](#)
- union `usb_cdc_func_desc_struct_t`  
CDC union Functional Descriptor structure for analyse class specific descriptor. [More...](#)

## Macros

- #define `USB_HOST_CDC_SET_LINE_CODING` 0x20U  
CDC class-specific request (SET\_LINE\_CODING)
- #define `USB_HOST_CDC_GET_LINE_CODING` 0x21U  
CDC class-specific request (GET\_LINE\_CODING)
- #define `USB_HOST_CDC_SET_CONTROL_LINE_STATE` 0x22U  
CDC class-specific request (SET\_CONTROL\_LINE\_STATE)
- #define `USB_HOST_ACM_UART_STATE_BITMAP_BTXCARRITER` 0x01U  
CDC class-specific notifications(SerialState) bitmap.
- #define `USB_HOST_ACM_UART_STATE_BITMAP_BRXCARRITER` 0x02U  
CDC class-specific notifications(SerialState) bitmap.
- #define `USB_HOST_ACM_UART_STATE_BITMAP_BBREAK` 0x04U  
CDC class-specific notifications(SerialState) bitmap.
- #define `USB_HOST_ACM_UART_STATE_BITMAP_BBRINGSIGNAL` 0x10U  
CDC class-specific notifications(SerialState) bitmap.
- #define `USB_HOST_CDC_CONTROL_LINE_STATE_DTR` 0x01U  
CDC class-specific request (SET\_CONTROL\_LINE\_STATE) bitmap.
- #define `USB_HOST_CDC_CONTROL_LINE_STATE_RTS` 0x02U  
CDC class-specific request (SET\_CONTROL\_LINE\_STATE) bitmap.
- #define `USB_HOST_DESC_SUBTYPE_HEADER` 0x00U  
CDC class-specific bDescriptor SubType in functional descriptors.
- #define `USB_HOST_DESC_SUBTYPE_CM` 0x01U  
CDC class-specific bDescriptor SubType in functional descriptors.

## USB CDC Class driver

- `#define USB_HOST_DESC_SUBTYPE_ACM 0x02U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_DLM 0x03U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_TR 0x04U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_TC_LSR 0x05U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_UNION 0x06U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_CS 0x07U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_DESC_SUBTYPE_TOM 0x08U`  
*CDC class-specific bDescriptor SubType in functional descriptors.*
- `#define USB_HOST_CDC_COMMUNICATIONS_CLASS_CODE 0x02U`  
*CDC class-specific code, Communications Interface Class Code.*
- `#define USB_HOST_CDC_SUBCLASS_ACM_CODE 0x02U`  
*CDC class-specific code, Communications Class Subclass Codes.*
- `#define USB_HOST_CDC_DATA_CLASS_CODE 0x0AU`  
*CDC class-specific code, Data Class Interface Codes.*

## USB CDC host class driver

- `usb_status_t USB_HostCdcInit (usb_device_handle deviceHandle, usb_host_class_handle *classHandle)`  
*Initializes the CDC instance.*
- `usb_status_t USB_HostCdcSetDataInterface (usb_host_class_handle classHandle, usb_host_interface_handle interfaceHandle, uint8_t alternateSetting, transfer_callback_t callbackFn, void *callbackParam)`  
*CDC set data interface callback and opens pipes.*
- `usb_status_t USB_HostCdcSetControlInterface (usb_host_class_handle classHandle, usb_host_interface_handle interfaceHandle, uint8_t alternateSetting, transfer_callback_t callbackFn, void *callbackParam)`  
*CDC set control interface callback and opens pipes.*
- `usb_status_t USB_HostCdcDeinit (usb_device_handle deviceHandle, usb_host_class_handle classHandle)`  
*Deinitializes the CDC instance.*
- `uint16_t USB_HostCdcGetPacketsize (usb_host_class_handle classHandle, uint8_t pipeType, uint8_t direction)`  
*Gets the pipe maximum packet size.*
- `usb_status_t USB_HostCdcDataRecv (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Receives data.*
- `usb_status_t USB_HostCdcDataSend (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Sends data.*
- `usb_status_t USB_HostCdcInterruptRecv (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`



*Interrupts the receive data.*

- `usb_status_t USB_HostCdcGetAcmLineCoding` (`usb_host_class_handle` classHandle, `usb_host_cdc_line_coding_struct_t` \*uartLineCoding, `transfer_callback_t` callbackFn, void \*callbackParam)

*CDC get line coding.*

- `usb_status_t USB_HostCdcSetAcmCtrlState` (`usb_host_class_handle` classHandle, `uint8_t` dtr, `uint8_t` rts, `transfer_callback_t` callbackFn, void \*callbackParam)

*CDC setControlLineState.*

- `usb_status_t USB_HostCdcGetAcmDescriptor` (`usb_host_class_handle` classHandle, `usb_host_cdc_head_function_desc_struct_t` \*\*headDesc, `usb_host_cdc_call_manage_desc_struct_t` \*\*callManageDesc, `usb_host_cdc_abstract_control_desc_struct_t` \*\*abstractControlDesc, `usb_host_cdc_union_interface_desc_struct_t` \*\*unionInterfaceDesc)

*CDC gets the ACM descriptor.*

- `usb_status_t USB_HostCdcControl` (`usb_host_class_handle` classHandle, `uint8_t` request\_type, `uint8_t` request, `uint8_t` wvalue\_l, `uint8_t` wvalue\_h, `uint16_t` wlength, `uint8_t` \*data, `transfer_callback_t` callbackFn, void \*callbackParam)

*CDC send control transfer common code.*

## 4.2.6 Data Structure Documentation

### 4.2.6.1 struct usb\_host\_cdc\_line\_coding\_struct\_t

#### Data Fields

- `uint32_t dwDTERate`  
*Data terminal rate, in bits per second.*
- `uint8_t bCharFormat`  
*Stop bits.*
- `uint8_t bParityType`  
*Parity.*
- `uint8_t bDataBits`  
*Data bits (5, 6, 7, 8 or 16).*

#### 4.2.6.1.0.1 Field Documentation

##### 4.2.6.1.0.1.1 uint8\_t usb\_host\_cdc\_line\_coding\_struct\_t::bDataBits

### 4.2.6.2 struct usb\_host\_cdc\_control\_line\_state\_struct\_t

#### Data Fields

- `uint16_t line_state`  
*D1, This signal corresponds to V.24 signal 105 and RS-232 signal RTS.*

#### 4.2.6.2.0.2 Field Documentation

##### 4.2.6.2.0.2.1 uint16\_t usb\_host\_cdc\_control\_line\_state\_struct\_t::line\_state

D0, This signal corresponds to V.24 signal 108/2 and RS-232 signal DTR

### 4.2.6.3 struct usb\_host\_cdc\_acm\_state\_struct\_t

#### Data Fields

- uint8\_t [reserved](#) [8]  
*Notify response by the device, this is used as notification header which is return by the device.*
- uint8\_t [bmstate](#)  
*UART State Bitmap Values.*
- uint8\_t [reserved1](#) [1]  
*Fix 4B align issue.*
- uint8\_t [reserved2](#) [2]  
*Fix 4B align issue.*

### 4.2.6.4 struct usb\_host\_cdc\_head\_function\_desc\_struct\_t

#### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE descriptor type.*
- uint8\_t [bDescriptorSubtype](#)  
*Header functional descriptor subtype.*
- uint8\_t [bcdCDC](#) [2]  
*USB Class Definitions for Communications Devices Specification release number in binary-coded decimal.*

#### 4.2.6.4.0.3 Field Documentation

4.2.6.4.0.3.1 uint8\_t usb\_host\_cdc\_head\_function\_desc\_struct\_t::bFunctionLength

4.2.6.4.0.3.2 uint8\_t usb\_host\_cdc\_head\_function\_desc\_struct\_t::bDescriptorType

4.2.6.4.0.3.3 uint8\_t usb\_host\_cdc\_head\_function\_desc\_struct\_t::bDescriptorSubtype

4.2.6.4.0.3.4 uint8\_t usb\_host\_cdc\_head\_function\_desc\_struct\_t::bcdCDC[2]

### 4.2.6.5 struct usb\_host\_cdc\_call\_manage\_desc\_struct\_t

#### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Call Management functional descriptor subtype.*
- uint8\_t [bmCapabilities](#)  
*The capabilities that this configuration supports.*
- uint8\_t [bDataInterface](#)

*Interface number of Data Class interface optionally used for call management.*

#### 4.2.6.5.0.4 Field Documentation

4.2.6.5.0.4.1 uint8\_t usb\_host\_cdc\_call\_manage\_desc\_struct\_t::bFunctionLength

4.2.6.5.0.4.2 uint8\_t usb\_host\_cdc\_call\_manage\_desc\_struct\_t::bDescriptorType

4.2.6.5.0.4.3 uint8\_t usb\_host\_cdc\_call\_manage\_desc\_struct\_t::bDescriptorSubtype

4.2.6.5.0.4.4 uint8\_t usb\_host\_cdc\_call\_manage\_desc\_struct\_t::bmCapabilities

4.2.6.5.0.4.5 uint8\_t usb\_host\_cdc\_call\_manage\_desc\_struct\_t::bDataInterface

#### 4.2.6.6 struct usb\_host\_cdc\_abstract\_control\_desc\_struct\_t

##### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Abstract Control Management functional descriptor subtype.*
- uint8\_t [bmCapabilities](#)  
*The capabilities that this configuration supports.*

#### 4.2.6.6.0.5 Field Documentation

4.2.6.6.0.5.1 uint8\_t usb\_host\_cdc\_abstract\_control\_desc\_struct\_t::bFunctionLength

4.2.6.6.0.5.2 uint8\_t usb\_host\_cdc\_abstract\_control\_desc\_struct\_t::bDescriptorType

4.2.6.6.0.5.3 uint8\_t usb\_host\_cdc\_abstract\_control\_desc\_struct\_t::bDescriptorSubtype

4.2.6.6.0.5.4 uint8\_t usb\_host\_cdc\_abstract\_control\_desc\_struct\_t::bmCapabilities

#### 4.2.6.7 struct usb\_host\_cdc\_direct\_line\_desc\_struct\_t

##### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Direct Line Management functional descriptor subtype,.*
- uint8\_t [bmCapabilities](#)  
*The capabilities that this configuration supports.*

## USB CDC Class driver

### 4.2.6.7.0.6 Field Documentation

4.2.6.7.0.6.1 uint8\_t usb\_host\_cdc\_direct\_line\_desc\_struct\_t::bFunctionLength

4.2.6.7.0.6.2 uint8\_t usb\_host\_cdc\_direct\_line\_desc\_struct\_t::bDescriptorType

4.2.6.7.0.6.3 uint8\_t usb\_host\_cdc\_direct\_line\_desc\_struct\_t::bDescriptorSubtype

4.2.6.7.0.6.4 uint8\_t usb\_host\_cdc\_direct\_line\_desc\_struct\_t::bmCapabilities

4.2.6.8 struct usb\_host\_cdc\_telephone\_ringer\_desc\_struct\_t

#### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Telephone Ringer functional descriptor subtype.*
- uint8\_t [bRingerVolSteps](#)  
*Number of discrete steps in volume supported by the ringer,.*
- uint8\_t [bNumRingerPatterns](#)  
*Number of ringer patterns supported.*

### 4.2.6.8.0.7 Field Documentation

4.2.6.8.0.7.1 uint8\_t usb\_host\_cdc\_telephone\_ringer\_desc\_struct\_t::bFunctionLength

4.2.6.8.0.7.2 uint8\_t usb\_host\_cdc\_telephone\_ringer\_desc\_struct\_t::bDescriptorType

4.2.6.8.0.7.3 uint8\_t usb\_host\_cdc\_telephone\_ringer\_desc\_struct\_t::bRingerVolSteps

4.2.6.8.0.7.4 uint8\_t usb\_host\_cdc\_telephone\_ringer\_desc\_struct\_t::bNumRingerPatterns

4.2.6.9 struct usb\_host\_cdc\_tclsr\_desc\_struct\_t

#### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Telephone Call State Reporting Capabilities descriptor subtype.*
- uint8\_t [bmCapabilities](#) [4]  
*Call and line state reporting capabilities of the device.*

#### 4.2.6.9.0.8 Field Documentation

4.2.6.9.0.8.1 uint8\_t usb\_host\_cdc\_tcLsr\_desc\_struct\_t::bFunctionLength

4.2.6.9.0.8.2 uint8\_t usb\_host\_cdc\_tcLsr\_desc\_struct\_t::bDescriptorType

4.2.6.9.0.8.3 uint8\_t usb\_host\_cdc\_tcLsr\_desc\_struct\_t::bDescriptorSubtype

4.2.6.9.0.8.4 uint8\_t usb\_host\_cdc\_tcLsr\_desc\_struct\_t::bmCapabilities[4]

#### 4.2.6.10 struct usb\_host\_cdc\_union\_interface\_desc\_struct\_t

##### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE descriptor type.*
- uint8\_t [bDescriptorSubtype](#)  
*Union Functional Descriptor SubType.*
- uint8\_t [bControlInterface](#)  
*USB Class Definitions for Communications Devices Specification release number in binary-coded decimal.*

#### 4.2.6.10.0.9 Field Documentation

4.2.6.10.0.9.1 uint8\_t usb\_host\_cdc\_union\_interface\_desc\_struct\_t::bFunctionLength

4.2.6.10.0.9.2 uint8\_t usb\_host\_cdc\_union\_interface\_desc\_struct\_t::bDescriptorType

4.2.6.10.0.9.3 uint8\_t usb\_host\_cdc\_union\_interface\_desc\_struct\_t::bDescriptorSubtype

4.2.6.10.0.9.4 uint8\_t usb\_host\_cdc\_union\_interface\_desc\_struct\_t::bControlInterface

#### 4.2.6.11 struct usb\_host\_cdc\_tom\_desc\_struct\_t

##### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE.*
- uint8\_t [bDescriptorSubtype](#)  
*Telephone Operational Modes functional descriptor subtype.*
- uint8\_t [bmCapabilities](#)  
*operational modes:.*

## USB CDC Class driver

### 4.2.6.11.0.10 Field Documentation

4.2.6.11.0.10.1 uint8\_t usb\_host\_cdc\_tom\_desc\_struct\_t::bFunctionLength

4.2.6.11.0.10.2 uint8\_t usb\_host\_cdc\_tom\_desc\_struct\_t::bDescriptorType

4.2.6.11.0.10.3 uint8\_t usb\_host\_cdc\_tom\_desc\_struct\_t::bDescriptorSubtype

4.2.6.11.0.10.4 uint8\_t usb\_host\_cdc\_tom\_desc\_struct\_t::bmCapabilities

### 4.2.6.12 struct usb\_host\_cdc\_common\_desc\_struct\_t

#### Data Fields

- uint8\_t [bFunctionLength](#)  
*Size of this descriptor in bytes.*
- uint8\_t [bDescriptorType](#)  
*CS\_INTERFACE descriptor type.*
- uint8\_t [bDescriptorSubtype](#)  
*Header functional descriptor subtype.*

### 4.2.6.12.0.11 Field Documentation

4.2.6.12.0.11.1 uint8\_t usb\_host\_cdc\_common\_desc\_struct\_t::bFunctionLength

4.2.6.12.0.11.2 uint8\_t usb\_host\_cdc\_common\_desc\_struct\_t::bDescriptorType

4.2.6.12.0.11.3 uint8\_t usb\_host\_cdc\_common\_desc\_struct\_t::bDescriptorSubtype

### 4.2.6.13 union usb\_cdc\_func\_desc\_struct\_t

## 4.2.7 Function Documentation

4.2.7.1 usb\_status\_t USB\_HostCdcInit ( usb\_device\_handle *deviceHandle*,  
usb\_host\_class\_handle \* *classHandle* )

This function allocates the resource for the CDC instance.

Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandle</i>	return Class handle.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory fail.

**4.2.7.2** `usb_status_t USB_HostCdcSetDataInterface ( usb_host_class_handle classHandle,  
usb_host_interface_handle interfaceHandle, uint8_t alternateSetting,  
transfer_callback_t callbackFn, void * callbackParam )`

Parameters

in	<i>classHandle</i>	The class handle.
in	<i>interface-Handle</i>	The interface handle.
in	<i>alternate-Setting</i>	The alternate setting value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.
<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

**4.2.7.3** `usb_status_t USB_HostCdcSetControlInterface ( usb_host_class_handle  
classHandle, usb_host_interface_handle interfaceHandle, uint8_t  
alternateSetting, transfer_callback_t callbackFn, void * callbackParam )`

## USB CDC Class driver

### Parameters

in	<i>classHandle</i>	The class handle.
in	<i>interface-Handle</i>	The interface handle.
in	<i>alternate-Setting</i>	The alternate setting value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.
<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

#### 4.2.7.4 **usb\_status\_t USB\_HostCdcDeinit ( usb\_device\_handle *deviceHandle*, usb\_host\_class\_handle *classHandle* )**

This function frees the resource for the CDC instance.

### Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandle</i>	The class handle.

### Return values

<i>kStatus_USB_Success</i>	The device is de-initialized successfully.
----------------------------	--------------------------------------------

#### 4.2.7.5 **uint16\_t USB\_HostCdcGetPacketsize ( usb\_host\_class\_handle *classHandle*, uint8\_t *pipeType*, uint8\_t *direction* )**



## Parameters

in	<i>classHandle</i>	The class handle.
in	<i>pipeType</i>	Its value is USB_ENDPOINT_CONTROL, USB_ENDPOINT_ISOC-HRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTERRUPT. See the usb_spec.h
in	<i>direction</i>	Pipe direction.

## Return values

0	The classHandle is NULL.
max	Packet size.

**4.2.7.6** `usb_status_t USB_HostCdcDataRecv ( usb_host_class_handle classHandle,  
uint8_t * buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *  
callbackParam )`

This function implements the CDC receiving data.

## Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLength</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostRecv.

## USB CDC Class driver

**4.2.7.7** `usb_status_t USB_HostCdcDataSend ( usb_host_class_handle classHandle,  
uint8_t * buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *  
callbackParam )`

This function implements the CDC sending data.

## Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLength</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

**4.2.7.8** `usb_status_t USB_HostCdcInterruptRecv ( usb_host_class_handle classHandle,  
uint8_t * buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *  
callbackParam )`

This function implements the interrupt receiving data.

## Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLength</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

## USB CDC Class driver

<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostRecv.

### 4.2.7.9 **usb\_status\_t USB\_HostCdcGetAcmLineCoding ( usb\_host\_class\_handle classHandle, usb\_host\_cdc\_line\_coding\_struct\_t \* uartLineCoding, transfer\_callback\_t callbackFn, void \* callbackParam )**

This function implements the CDC GetLineCoding request. See the PSTN specification.

Parameters

<i>classHandle</i>	The class handle.
<i>uartLine-Coding</i>	The line coding pointer.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

### 4.2.7.10 **usb\_status\_t USB\_HostCdcSetAcmCtrlState ( usb\_host\_class\_handle classHandle, uint8\_t dtr, uint8\_t rts, transfer\_callback\_t callbackFn, void \* callbackParam )**

This function implements the CDC etControlLineState request. See PSTN specification.

Parameters

<i>classHandle</i>	The class handle.
<i>dtr</i>	The DRS value.

<i>rts</i>	The RTS value.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

**4.2.7.11** `usb_status_t USB_HostCdcGetAcmDescriptor ( usb_host_class_handle classHandle, usb_host_cdc_head_function_desc_struct_t ** headDesc, usb_host_cdc_call_manage_desc_struct_t ** callManageDesc, usb_host_cdc_abstract_control_desc_struct_t ** abstractControlDesc, usb_host_cdc_union_interface_desc_struct_t ** unionInterfaceDesc )`

This function is hunting for the class-specific ACM descriptor in the configuration and gets the corresponding descriptor.

## Parameters

<i>classHandle</i>	The class handle.
<i>headDesc</i>	The head function descriptor pointer.
<i>callManage-Desc</i>	The call management functional descriptor pointer.
<i>abstract-ControlDesc</i>	The abstract control management functional pointer.
<i>unionInterface-Desc</i>	The union functional descriptor pointer.

## Return values

<i>kStatus_USB_Error</i>	Analyse descriptor error.
--------------------------	---------------------------

## USB CDC Class driver

**4.2.7.12** `usb_status_t USB_HostCdcControl ( usb_host_class_handle classHandle, uint8_t request_type, uint8_t request, uint8_t wvalue_l, uint8_t wvalue_h, uint16_t wlength, uint8_t * data, transfer_callback_t callbackFn, void * callbackParam )`

## Parameters

<i>classHandle</i>	The class handle.
<i>request_type</i>	Set up the packet request type.
<i>request</i>	Set up the packet request value.
<i>wvalue_l</i>	Set up the packet wvalue low byte.
<i>wvalue_h</i>	Set up the packet wvalue high byte.
<i>wlength</i>	Set up the packet wlength value.
<i>data</i>	Data buffer pointer
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Returns

An error code or kStatus\_USB\_Success.

### 4.3 USB HID Class driver

#### 4.3.1 Overview

The USB HID consists primarily of devices that are used by humans to control the operation of computer systems. Typical examples of HID class devices include keyboard and mouse. This section describes the programming interface of the USB HOST HID class driver. The USB HOST HID class driver handles the specific control requests for HID class and transfers data to and from the device through the interrupt pipe.

#### 4.3.2 USB Host HID Initialization

When the HID device is attached, the HID initialization flow is as follows:

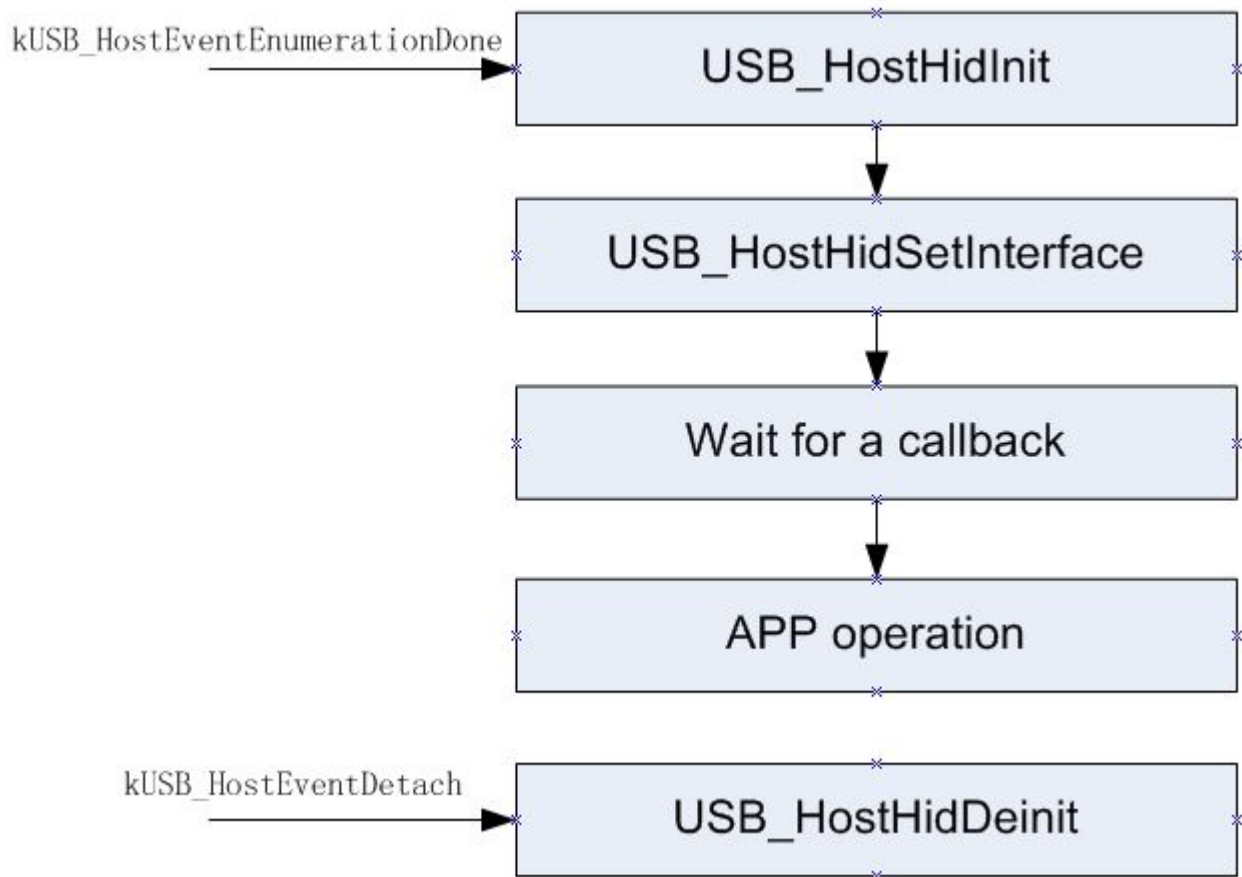


Figure 4.3.1: Host HID Initialization

The above picture describes the following steps:

- Call the `USB_HostHidInit` to initialize the HID class instance `usb_host_hid_instance_t` and the return class handle pointer to the hid class instance. The driver uses an instantiation of the `usb_host_hid_instance_t` structure to maintain the current state of a HID instance module driver. This structure



holds the USB host handle and the USB device handle and keeps track of transfer information, alternate setting, pipes and interfaces that are enumerated for attached HID device.

- Call the `USB_HostHidSetInterface` to set the HID class interface, which opens the interface's pipes.
- Wait the last step operation callback.
- Call the `USB_HostHidSetIdle` to set the HID device.
- Wait the last step operation callback.
- Call the `USB_HostHidGetReportDescriptor` to get the HID report descriptor.
- Wait the last step operation callback.
- Call the `USB_HostHidSetProtocol` to set protocol.
- Wait the last step operation callback.
- Call the `USB_HostHidRecv` to receive data from the device, or call `USB_HostHidSend` to send data to the device.
- Wait the last step operation callback.
- Process data and receive or send again.

### 4.3.3 USB Host HID Deinitialization

An application calls the `USB_HostHidDeinit` to deinitialize the HID. This function cancels the transfer, closes the pipe, and releases the HID class instance.

There are two use cases to call this function:

- The HID device is detached and this function is called to release the resource.
- An application calls this function and calls the `USB_HostHidInit` to reinitialize the HID class.

### 4.3.4 USB Host HID Send data

Provides the buffer pointer, the buffer length, the callback function, and the callback parameter and calls the `USB_HostHidSend` to start asynchronous sending. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

### 4.3.5 USB Host HID Receive data

Provides the buffer pointer, the buffer length, the callback function, and the callback parameter and calls the `USB_HostHidRecv` to start asynchronous receiving. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

## Data Structures

- struct `usb_host_hid_instance_t`  
*HID instance structure and HID `usb_host_class_handle` pointer to this structure. [More...](#)*
- struct `usb_host_hid_descriptor_t`

## USB HID Class driver

- *HID descriptor structure according to the 6.2.1 in HID specification. [More...](#)*
- struct [usb\\_host\\_hid\\_class\\_descriptor\\_t](#)  
*HID descriptor structure according to the 6.2.1 in HID specification. [More...](#)*

## Macros

- #define [USB\\_HOST\\_HID\\_GET\\_REPORT](#) (0x01U)  
*HID class-specific request (get report)*
- #define [USB\\_HOST\\_HID\\_GET\\_IDLE](#) (0x02U)  
*HID class-specific request (get idle)*
- #define [USB\\_HOST\\_HID\\_GET\\_PROTOCOL](#) (0x03U)  
*HID class-specific request (get protocol)*
- #define [USB\\_HOST\\_HID\\_SET\\_REPORT](#) (0x09U)  
*HID class-specific request (set report)*
- #define [USB\\_HOST\\_HID\\_SET\\_IDLE](#) (0x0AU)  
*HID class-specific request (set idle)*
- #define [USB\\_HOST\\_HID\\_SET\\_PROTOCOL](#) (0x0BU)  
*HID class-specific request (set protocol)*
- #define [USB\\_HOST\\_HID\\_CLASS\\_CODE](#) (3U)  
*HID class code.*
- #define [USB\\_HOST\\_HID\\_SUBCLASS\\_CODE\\_NONE](#) (0U)  
*HID sub-class code.*
- #define [USB\\_HOST\\_HID\\_SUBCLASS\\_CODE\\_BOOT](#) (1U)  
*HID sub-class code.*
- #define [USB\\_HOST\\_HID\\_PROTOCOL\\_KEYBOARD](#) (1U)  
*HID class protocol code.*
- #define [USB\\_HOST\\_HID\\_PROTOCOL\\_MOUSE](#) (2U)  
*HID class protocol code.*
- #define [USB\\_HOST\\_HID\\_PROTOCOL\\_NONE](#) (0U)  
*HID class protocol code.*
- #define [USB\\_HOST\\_HID\\_REQUEST\\_PROTOCOL\\_BOOT](#) (0U)  
*HID get/set protocol request data code.*
- #define [USB\\_HOST\\_HID\\_REQUEST\\_PROTOCOL\\_REPORT](#) (1U)  
*HID get/set protocol request data code.*

## USB host HID class APIs

- [usb\\_status\\_t](#) [USB\\_HostHidInit](#) ([usb\\_device\\_handle](#) deviceHandle, [usb\\_host\\_class\\_handle](#) \*classHandle)  
*Initializes the HID instance.*
- [usb\\_status\\_t](#) [USB\\_HostHidSetInterface](#) ([usb\\_host\\_class\\_handle](#) classHandle, [usb\\_host\\_interface\\_handle](#) interfaceHandle, [uint8\\_t](#) alternateSetting, [transfer\\_callback\\_t](#) callbackFn, void \*callbackParam)  
*Sets the interface.*
- [usb\\_status\\_t](#) [USB\\_HostHidDeinit](#) ([usb\\_device\\_handle](#) deviceHandle, [usb\\_host\\_class\\_handle](#) classHandle)  
*Deinitializes the the HID instance.*

- `uint16_t USB_HostHidGetPacketsize (usb_host_class_handle classHandle, uint8_t pipeType, uint8_t direction)`  
*Gets the pipe maximum packet size.*
- `usb_status_t USB_HostHidGetReportDescriptor (usb_host_class_handle classHandle, uint8_t *buffer, uint16_t buffer_len, transfer_callback_t callbackFn, void *callbackParam)`  
*HID get report descriptor.*
- `usb_status_t USB_HostHidRecv (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Receives data.*
- `usb_status_t USB_HostHidSend (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Sends data.*
- `usb_status_t USB_HostHidGetIdle (usb_host_class_handle classHandle, uint8_t reportId, uint8_t *idleRate, transfer_callback_t callbackFn, void *callbackParam)`  
*HID get idle.*
- `usb_status_t USB_HostHidSetIdle (usb_host_class_handle classHandle, uint8_t reportId, uint8_t idleRate, transfer_callback_t callbackFn, void *callbackParam)`  
*HID set idle.*
- `usb_status_t USB_HostHidGetProtocol (usb_host_class_handle classHandle, uint8_t *protocol, transfer_callback_t callbackFn, void *callbackParam)`  
*HID get protocol.*
- `usb_status_t USB_HostHidSetProtocol (usb_host_class_handle classHandle, uint8_t protocol, transfer_callback_t callbackFn, void *callbackParam)`  
*HID set protocol.*
- `usb_status_t USB_HostHidGetReport (usb_host_class_handle classHandle, uint8_t reportId, uint8_t reportType, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*HID get report.*
- `usb_status_t USB_HostHidSetReport (usb_host_class_handle classHandle, uint8_t reportId, uint8_t reportType, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*HID set report.*

## 4.3.6 Data Structure Documentation

### 4.3.6.1 struct usb\_host\_hid\_instance\_t

#### Data Fields

- `usb_host_handle hostHandle`  
*This instance's related host handle.*
- `usb_device_handle deviceHandle`  
*This instance's related device handle.*
- `usb_host_interface_handle interfaceHandle`  
*This instance's related interface handle.*
- `usb_host_pipe_handle controlPipe`  
*This instance's related device control pipe.*
- `usb_host_pipe_handle inPipe`

## USB HID Class driver

- *HID interrupt in pipe.*  
• [usb\\_host\\_pipe\\_handle outPipe](#)
- *HID interrupt out pipe.*  
• [transfer\\_callback\\_t inCallbackFn](#)  
*HID interrupt in transfer callback function pointer.*
- void \* [inCallbackParam](#)  
*HID interrupt in transfer callback parameter.*
- [transfer\\_callback\\_t outCallbackFn](#)  
*HID interrupt out transfer callback function pointer.*
- void \* [outCallbackParam](#)  
*HID interrupt out transfer callback parameter.*
- [transfer\\_callback\\_t controlCallbackFn](#)  
*HID control transfer callback function pointer.*
- void \* [controlCallbackParam](#)  
*HID control transfer callback parameter.*
- [usb\\_host\\_transfer\\_t \\* controlTransfer](#)  
*Ongoing control transfer.*
- uint16\_t [inPacketSize](#)  
*HID interrupt in maximum packet size.*
- uint16\_t [outPacketSize](#)  
*HID interrupt out maximum packet size.*

### 4.3.6.2 struct usb\_host\_hid\_descriptor\_t

#### Data Fields

- uint8\_t [bLength](#)  
*Total size of the HID descriptor.*
- uint8\_t [bDescriptorType](#)  
*Constant name specifying type of HID descriptor.*
- uint8\_t [bcdHID](#) [2]  
*Numeric expression identifying the HID Class Specification release.*
- uint8\_t [bCountryCode](#)  
*Numeric expression identifying country code of the localized hardware.*
- uint8\_t [bNumDescriptors](#)  
*Numeric expression specifying the number of class descriptors.*
- uint8\_t [bHidDescriptorType](#)  
*Constant name identifying type of class descriptor.*
- uint8\_t [wDescriptorLength](#) [2]  
*Numeric expression that is the total size of the Report descriptor.*

### 4.3.6.3 struct usb\_host\_hid\_class\_descriptor\_t

#### Data Fields

- uint8\_t [bHidDescriptorType](#)  
*Constant name specifying type of optional descriptor.*
- uint8\_t [wDescriptorLength](#) [2]  
*Numeric expression that is the total size of the optional descriptor.*

## 4.3.7 Function Documentation

**4.3.7.1** `usb_status_t USB_HostHidInit ( usb_device_handle deviceHandle,  
usb_host_class_handle * classHandle )`

This function allocate the resource for the HID instance.

## USB HID Class driver

### Parameters

in	<i>deviceHandle</i>	The device handle.
out	<i>classHandle</i>	Return class handle.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory fail.

### 4.3.7.2 **usb\_status\_t USB\_HostHidSetInterface ( usb\_host\_class\_handle *classHandle*, usb\_host\_interface\_handle *interfaceHandle*, uint8\_t *alternateSetting*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function binds the interface with the HID instance.

### Parameters

in	<i>classHandle</i>	The class handle.
in	<i>interface-Handle</i>	The interface handle.
in	<i>alternate-Setting</i>	The alternate setting value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.

<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

#### 4.3.7.3 **usb\_status\_t USB\_HostHidDeinit ( usb\_device\_handle *deviceHandle*, usb\_host\_class\_handle *classHandle* )**

This function frees the resources for the HID instance.

Parameters

in	<i>deviceHandle</i>	The device handle.
in	<i>classHandle</i>	The class handle.

Return values

<i>kStatus_USB_Success</i>	The device is de-initialized successfully.
----------------------------	--------------------------------------------

#### 4.3.7.4 **uint16\_t USB\_HostHidGetPacketsize ( usb\_host\_class\_handle *classHandle*, uint8\_t *pipeType*, uint8\_t *direction* )**

Parameters

in	<i>classHandle</i>	The class handle.
in	<i>pipeType</i>	Its value is USB_ENDPOINT_CONTROL, USB_ENDPOINT_ISOCHRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTERRUPT. See the usb_spec.h
in	<i>direction</i>	Pipe direction.

Return values

<i>0</i>	The classHandle is NULL.
<i>Maximum</i>	packet size.

#### 4.3.7.5 **usb\_status\_t USB\_HostHidGetReportDescriptor ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *buffer*, uint16\_t *buffer\_len*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the HID report descriptor request.

## USB HID Class driver

### Parameters

in	<i>classHandle</i>	The class handle.
out	<i>buffer</i>	The buffer pointer.
in	<i>buffer_len</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

**4.3.7.6** `usb_status_t USB_HostHidRecv ( usb_host_class_handle classHandle, uint8_t * buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the HID receiving data.

### Parameters

in	<i>classHandle</i>	The class handle.
out	<i>buffer</i>	The buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.



<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostRecv.

**4.3.7.7** `usb_status_t USB_HostHidSend ( usb_host_class_handle classHandle, uint8_t * buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the HID sending data.

Parameters

in	<i>classHandle</i>	The class handle.
in	<i>buffer</i>	The buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	Send request successfully.
<i>kStatus_USB_Invalid_Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

**4.3.7.8** `usb_status_t USB_HostHidGetIdle ( usb_host_class_handle classHandle, uint8_t reportId, uint8_t * idleRate, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the HID class-specific request (get idle).

Parameters

in	<i>classHandle</i>	The class handle.
----	--------------------	-------------------

## USB HID Class driver

in	<i>reportId</i>	Report ID.
out	<i>idleRate</i>	Return idle rate value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

#### 4.3.7.9 **usb\_status\_t USB\_HostHidSetIdle ( usb\_host\_class\_handle *classHandle*, uint8\_t *reportId*, uint8\_t *idleRate*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the HID class-specific request (set idle).

### Parameters

in	<i>classHandle</i>	The class handle.
in	<i>reportId</i>	Report ID.
in	<i>idleRate</i>	Idle rate value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

#### 4.3.7.10 **usb\_status\_t USB\_HostHidGetProtocol ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *protocol*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the HID class-specific request (get protocol).

## Parameters

in	<i>classHandle</i>	The class handle.
out	<i>protocol</i>	Return protocol value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

#### 4.3.7.11 **usb\_status\_t USB\_HostHidSetProtocol ( usb\_host\_class\_handle *classHandle*, uint8\_t *protocol*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the HID class-specific request (set protocol).

## Parameters

in	<i>classHandle</i>	The class handle.
in	<i>protocol</i>	Protocol value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

## USB HID Class driver

**4.3.7.12** `usb_status_t USB_HostHidGetReport ( usb_host_class_handle classHandle,  
uint8_t reportId, uint8_t reportType, uint8_t * buffer, uint32_t bufferLength,  
transfer_callback_t callbackFn, void * callbackParam )`

This function implements the HID class-specific request (get report).

## Parameters

in	<i>classHandle</i>	The class handle.
in	<i>reportId</i>	Report ID.
in	<i>reportType</i>	Report type.
out	<i>buffer</i>	The buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

#### 4.3.7.13 **usb\_status\_t USB\_HostHidSetReport ( usb\_host\_class\_handle *classHandle*, uint8\_t *reportId*, uint8\_t *reportType*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the HID class specific request (set report).

## Parameters

in	<i>classHandle</i>	The class handle.
in	<i>reportId</i>	Report ID.
in	<i>reportType</i>	Report type.
in	<i>buffer</i>	The buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## USB HID Class driver

### Return values

<i>kStatus_USB_Success</i>	Request successful.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

## 4.4 USB MSC Class driver

### 4.4.1 Overview

The USB Mass Storage Class (or USB MSC) defines the mass storage USB device. A typical example is a U-disk. This section describes the programming interface of the USB Host MSC class driver. The USB Host MSC class driver handles the specific control requests for MSC class and transfers data to and from the device through the interrupt pipe.

### 4.4.2 USB Host MSC Initialization

When the MSD device is attached, the MSD initialization flow is as follows:

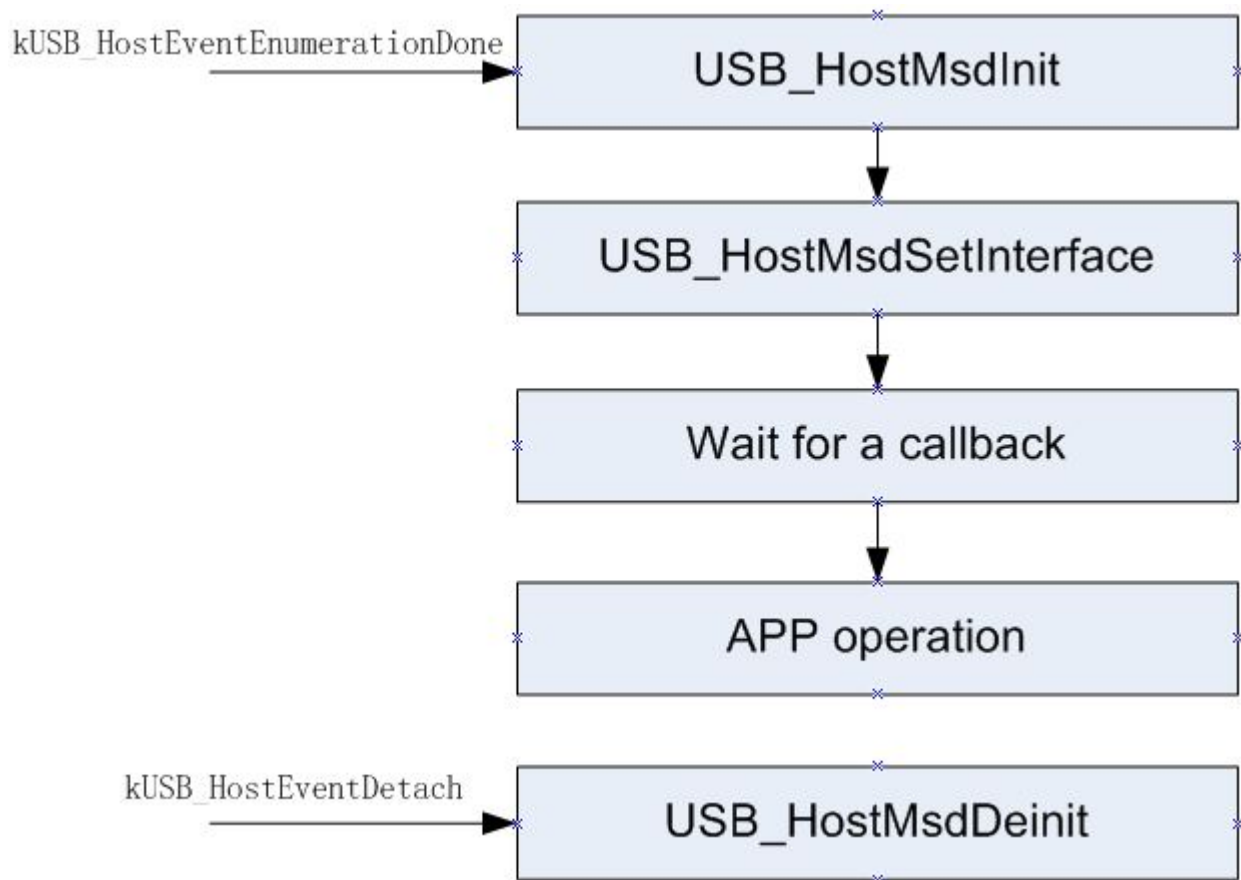


Figure 4.4.1: Host MSD Initialization

The above figure describes the following steps:

- Call the `USB_HostMsdInit` to initialize the MSD class instance `usb_host_msd_instance_t` and the return class handle pointer to the MSD class instance. The driver uses an instantiation of the `usb_host_msd_instance_t` structure to maintain the current state of a MSC instance module driver. This

## USB MSC Class driver

structure holds the USB host handle and the USB device handle and keeps track of transfer information, alternate setting, pipes and interfaces that are enumerated for attached MSC device.

- Call the `USB_HostMsdSetInterface` to set the MSD class interface, which opens the interface's pipes.
- Wait the last step operation callback.
- Test the MSD device: read capacity, write data, or read data.

### 4.4.3 USB Host MSC Deinitialization

An application calls the `USB_HostMsdDeinit` to deinitialize the MSD. This function cancels the transfer, closes the pipe, and releases the MSD class instance.

There are two use cases to call this function:

- The MSD device is detached and this function is called to free the resource.
- An application calls this function and then calls the `USB_HostMsdInit` to reinitialize the MSD class.

### 4.4.4 USB Host MSC UFI Command

Provides the buffer pointer, the buffer length, the callback function, the callback parameter, and other parameters and calls the `USB_HostMsdxx` to start an asynchronous MSD UFI command. Then, the callback function is called with one command status parameter when the command succeeds or fails. For example, `USB_HostMsdRead10` needs these parameters: buffer pointer, reading length, reading block number, callback function, callback parameter, logical unit number and start the block address.

## Data Structures

- struct `usb_host_cbw_t`  
*MSC Bulk-Only command block wrapper (CBW) [More...](#)*
- struct `usb_host_csw_t`  
*MSC Bulk-Only command status wrapper (CSW) [More...](#)*
- struct `usb_host_msd_command_t`  
*MSC UFI command information structure. [More...](#)*
- struct `usb_host_msd_instance_t`  
*MSD instance structure, MSD `usb_host_class_handle` pointer to this structure. [More...](#)*
- struct `usb_host_ufi_sense_data_t`  
*UFI standard sense data structure. [More...](#)*
- struct `usb_host_ufi_inquiry_data_t`  
*UFI standard inquiry data structure. [More...](#)*
- struct `usb_host_ufi_read_capacity_t`  
*UFI read capacity data structure. [More...](#)*



## Macros

- #define **USB\_HOST\_MSD\_RETRY\_MAX\_TIME** (1U)  
*retry time when transfer fail, when all the retries fail the transfer callback with error status*
- #define **USB\_HOST\_MSD\_BLOCK\_SIZE** (512U)  
*mass storage block size*
- #define **USB\_HOST\_MSD\_CLASS\_CODE** (8U)  
*MSD class code.*
- #define **USB\_HOST\_MSD\_SUBCLASS\_CODE\_UFI** (4U)  
*MSD sub-class code.*
- #define **USB\_HOST\_MSD\_SUBCLASS\_CODE\_SCSI** (6U)  
*MSD sub-class code.*
- #define **USB\_HOST\_MSD\_PROTOCOL\_BULK** (0x50U)  
*MSD protocol code.*
- #define **USB\_HOST\_HID\_MASS\_STORAGE\_RESET** (0xFFU)  
*MSD class-specific request (mass storage reset)*
- #define **USB\_HOST\_HID\_GET\_MAX\_LUN** (0xFEU)  
*MSD class-specific request (get maximum logical unit number)*

## Enumerations

- enum **usb\_host\_msd\_command\_status\_t**  
*UFI command process status.*

## USB host MSD class APIs

- **usb\_status\_t** **USB\_HostMsdInit** (**usb\_device\_handle** deviceHandle, **usb\_host\_class\_handle** \*classHandle)  
*Initializes the MSD instance.*
- **usb\_status\_t** **USB\_HostMsdSetInterface** (**usb\_host\_class\_handle** classHandle, **usb\_host\_interface\_handle** interfaceHandle, **uint8\_t** alternateSetting, **transfer\_callback\_t** callbackFn, void \*callbackParam)  
*Sets the interface.*
- **usb\_status\_t** **USB\_HostMsdDeinit** (**usb\_device\_handle** deviceHandle, **usb\_host\_class\_handle** classHandle)  
*Deinitializes the MSD instance.*
- **usb\_status\_t** **USB\_HostMsdMassStorageReset** (**usb\_host\_class\_handle** classHandle, **transfer\_callback\_t** callbackFn, void \*callbackParam)  
*Mass storage reset.*
- **usb\_status\_t** **USB\_HostMsdGetMaxLun** (**usb\_host\_class\_handle** classHandle, **uint8\_t** \*logicalUnitNumber, **transfer\_callback\_t** callbackFn, void \*callbackParam)  
*Gets the maximum logical unit number.*
- **usb\_status\_t** **USB\_HostMsdRead10** (**usb\_host\_class\_handle** classHandle, **uint8\_t** logicalUnit, **uint32\_t** blockAddress, **uint8\_t** \*buffer, **uint32\_t** bufferLength, **uint32\_t** blockNumber, **transfer\_callback\_t** callbackFn, void \*callbackParam)  
*Mass storage read (10).*

- `usb_status_t USB_HostMsRead12` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint32_t` blockAddress, `uint8_t` \*buffer, `uint32_t` bufferLength, `uint32_t` blockNumber, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage read (12).*
- `usb_status_t USB_HostMsWrite10` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint32_t` blockAddress, `uint8_t` \*buffer, `uint32_t` bufferLength, `uint32_t` blockNumber, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage write (10).*
- `usb_status_t USB_HostMsWrite12` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint32_t` blockAddress, `uint8_t` \*buffer, `uint32_t` bufferLength, `uint32_t` blockNumber, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage write (12).*
- `usb_status_t USB_HostMsReadCapacity` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage read capacity.*
- `usb_status_t USB_HostMsTestUnitReady` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage test unit ready.*
- `usb_status_t USB_HostMsRequestSense` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*mass storage request sense.*
- `usb_status_t USB_HostMsModeSelect` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage mode select.*
- `usb_status_t USB_HostMsModeSense` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` pageControl, `uint8_t` pageCode, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage mode sense.*
- `usb_status_t USB_HostMsInquiry` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage inquiry.*
- `usb_status_t USB_HostMsReadFormatCapacities` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage read format capacities.*
- `usb_status_t USB_HostMsFormatUnit` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` trackNumber, `uint16_t` interLeave, `uint8_t` \*buffer, `uint32_t` bufferLength, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage format unit.*
- `usb_status_t USB_HostMsPreventAllowRemoval` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` prevent, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage prevents/allows a medium removal.*
- `usb_status_t USB_HostMsWriteAndVerify` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint32_t` blockAddress, `uint8_t` \*buffer, `uint32_t` bufferLength, `uint32_t` blockNumber, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Mass storage write and verify.*
- `usb_status_t USB_HostMsStartStopUnit` (`usb_host_class_handle` classHandle, `uint8_t` logicalUnit, `uint8_t` loadEject, `uint8_t` start, `transfer_callback_t` callbackFn, void \*callbackParam)

- *Mass storage start stop unit.*  
**usb\_status\_t USB\_HostMsDVerify** (**usb\_host\_class\_handle** classHandle, **uint8\_t** logicalUnit, **uint32\_t** blockAddress, **uint16\_t** verificationLength, **transfer\_callback\_t** callbackFn, void \*callbackParam)
- *Mass storage verify.*  
**usb\_status\_t USB\_HostMsDRezeroUnit** (**usb\_host\_class\_handle** classHandle, **uint8\_t** logicalUnit, **transfer\_callback\_t** callbackFn, void \*callbackParam)
- *Mass storage rezero.*  
**usb\_status\_t USB\_HostMsDSeek10** (**usb\_host\_class\_handle** classHandle, **uint8\_t** logicalUnit, **uint32\_t** blockAddress, **transfer\_callback\_t** callbackFn, void \*callbackParam)
- *Mass storage seek(10).*  
**usb\_status\_t USB\_HostMsDSendDiagnostic** (**usb\_host\_class\_handle** classHandle, **uint8\_t** logicalUnit, **uint8\_t** selfTest, **transfer\_callback\_t** callbackFn, void \*callbackParam)
- *Mass storage send diagnostic.*

## 4.4.5 Data Structure Documentation

### 4.4.5.1 struct usb\_host\_cbw\_t

#### Data Fields

- **uint32\_t CBWSignature**  
*Signature that helps identify this data packet as a CBW.*
- **uint32\_t CBWTag**  
*A Command Block Tag sent by the host.*
- **uint32\_t CBWDataTransferLength**  
*The number of bytes of data that the host expects to transfer on the Bulk-In or Bulk-Out endpoint during the execution of this command.*
- **uint8\_t CBWFlags**  
*Bit 7 Direction - the device shall ignore this bit if the dCBWDataTransferLength field is zero, otherwise:  
0 = Data-Out from host to the device, 1 = Data-In from the device to the host.*
- **uint8\_t CBWLun**  
*The device Logical Unit Number (LUN) to which the command block is being sent.*
- **uint8\_t CBWCBLength**  
*The valid length of the CBWCB in bytes.*
- **uint8\_t CBWCB [16]**  
*The command block to be executed by the device.*

#### 4.4.5.1.0.12 Field Documentation

##### 4.4.5.1.0.12.1 uint32\_t usb\_host\_cbw\_t::CBWSignature

The signature field shall contain the value 43425355h (little endian), indicating a CBW

##### 4.4.5.1.0.12.2 uint32\_t usb\_host\_cbw\_t::CBWTag

The device shall echo the contents of this field back to the host in the dCSWTag field of the associated CSW

## USB MSC Class driver

### 4.4.5.1.0.12.3 `uint8_t usb_host_cbw_t::CBWFlags`

Bit 6 Obsolete. The host shall set this bit to zero. Bits 5..0 Reserved - the host shall set these bits to zero.

### 4.4.5.1.0.12.4 `uint8_t usb_host_cbw_t::CBWCBLength`

This defines the valid length of the command block. The only legal values are 1 through 16 (01h through 10h).

## 4.4.5.2 `struct usb_host_csw_t`

### Data Fields

- `uint32_t CSWSignature`  
*Signature that helps identify this data packet as a CSW.*
- `uint32_t CSWTag`  
*The device shall set this field to the value received in the `dCBWTag` of the associated CBW.*
- `uint32_t CSWDataResidue`  
*the difference between the amount of data expected as stated in the `dCBWDataTransferLength` and the actual amount of relevant data processed by the device.*
- `uint8_t CSWStatus`  
*bCSWStatus indicates the success or failure of the command.*

### 4.4.5.2.0.13 Field Documentation

#### 4.4.5.2.0.13.1 `uint32_t usb_host_csw_t::CSWSignature`

The signature field shall contain the value 53425355h (little endian), indicating CSW.

#### 4.4.5.2.0.13.2 `uint32_t usb_host_csw_t::CSWDataResidue`

#### 4.4.5.2.0.13.3 `uint8_t usb_host_csw_t::CSWStatus`

00h - Command passed. 01h - Command Failed. 02h - Phase error. others - Reserved.

## 4.4.5.3 `struct usb_host_msd_command_t`

### Data Fields

- `usb_host_cbw_t cbwBlock`  
*CBW data block.*
- `usb_host_csw_t cswBlock`  
*CSW data block.*
- `uint8_t * dataBuffer`  
*Data buffer pointer.*
- `uint32_t dataLength`  
*Data buffer length.*
- `uint32_t dataSofar`  
*Successful transfer data length.*

- `usb_host_transfer_t * transfer`  
*The transfer is used for processing the UFI command.*
- `uint8_t retryTime`  
*The UFI command residual retry time, when it reduce to zero the UFI command fail.*
- `uint8_t dataDirection`  
*The data direction, its value is `USB_OUT` or `USB_IN`.*

#### 4.4.5.4 struct usb\_host\_msd\_instance\_t

##### Data Fields

- `usb_host_handle hostHandle`  
*This instance's related host handle.*
- `usb_device_handle deviceHandle`  
*This instance's related device handle.*
- `usb_host_interface_handle interfaceHandle`  
*This instance's related interface handle.*
- `usb_host_pipe_handle controlPipe`  
*This instance's related device control pipe.*
- `usb_host_pipe_handle outPipe`  
*MSD bulk out pipe.*
- `usb_host_pipe_handle inPipe`  
*MSD bulk in pipe.*
- `transfer_callback_t commandCallbackFn`  
*MSD UFI command callback function pointer.*
- `void * commandCallbackParam`  
*MSD UFI command callback parameter.*
- `transfer_callback_t controlCallbackFn`  
*MSD control transfer callback function pointer.*
- `void * controlCallbackParam`  
*MSD control transfer callback parameter.*
- `usb_host_transfer_t * controlTransfer`  
*Ongoing control transfer.*
- `usb_host_msd_command_t msdCommand`  
*Ongoing MSD UFI command information.*
- `uint8_t commandStatus`  
*UFI command process status, see `command_status_t`.*
- `uint8_t internalResetRecovery`  
*1 - class driver internal mass storage reset recovery is on-going; 0 - application call `USB_HostMsdMassStorageReset` to reset or there is no reset*

#### 4.4.5.5 struct usb\_host\_ufi\_sense\_data\_t

##### Data Fields

- `uint8_t errorCode`  
*This field shall contain a value of 70h to indicate current errors.*
- `uint8_t reserved1`  
*Reserved field.*

## USB MSC Class driver

- uint8\_t [senseKey](#)  
*Provide a hierarchy of error or command result information.*
- uint8\_t [information](#) [4]  
*This field is command-specific; it is typically used by some commands to return a logical block address denoting where an error occurred.*
- uint8\_t [additionalSenseLength](#)  
*The UFI device sets the value of this field to ten, to indicate that ten more bytes of sense data follow this field.*
- uint8\_t [reserved2](#) [4]  
*Reserved field.*
- uint8\_t [additionalSenseCode](#)  
*Provide a hierarchy of error or command result information.*
- uint8\_t [additionalSenseCodeQualifier](#)  
*Provide a hierarchy of error or command result information.*
- uint8\_t [reserved3](#) [4]  
*Reserved field.*

### 4.4.5.6 struct usb\_host\_ufi\_inquiry\_data\_t

#### Data Fields

- uint8\_t [peripheralDeviceType](#)  
*Identifies the device currently connected to the requested logical unit.*
- uint8\_t [removableMediaBit](#)  
*This shall be set to one to indicate removable media.*
- uint8\_t [version](#)  
*Version.*
- uint8\_t [responseDataFormat](#)  
*A value of 01h shall be used for UFI device.*
- uint8\_t [additionalLength](#)  
*Specify the length in bytes of the parameters.*
- uint8\_t [reserved1](#) [3]  
*Reserved field.*
- uint8\_t [vendorInformation](#) [8]  
*Contains 8 bytes of ASCII data identifying the vendor of the product.*
- uint8\_t [productIdentification](#) [16]  
*Contains 16 bytes of ASCII data as defined by the vendor.*
- uint8\_t [productRevisionLevel](#) [4]  
*Contains 4 bytes of ASCII data as defined by the vendor.*

### 4.4.5.7 struct usb\_host\_ufi\_read\_capacity\_t

#### Data Fields

- uint8\_t [lastLogicalBlockAddress](#) [4]  
*The logical block number.*
- uint8\_t [blockLengthInBytes](#) [4]  
*Block size.*

## 4.4.6 Function Documentation

**4.4.6.1** `usb_status_t USB_HostMsdlInit ( usb_device_handle deviceHandle,  
usb_host_class_handle * classHandle )`

This function allocates the resources for the MSD instance.

## USB MSC Class driver

### Parameters

in	<i>deviceHandle</i>	The device handle.
out	<i>classHandle</i>	Return class handle.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory fail.

**4.4.6.2** `usb_status_t USB_HostMsdSetInterface ( usb_host_class_handle classHandle,  
usb_host_interface_handle interfaceHandle, uint8_t alternateSetting,  
transfer_callback_t callbackFn, void * callbackParam )`

This function binds the interface with the MSD instance.

### Parameters

in	<i>classHandle</i>	The class handle.
in	<i>interface-Handle</i>	The interface handle.
in	<i>alternate-Setting</i>	The alternate setting value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Success</i>	Callback return status, the command succeeded.



<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.
<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

#### 4.4.6.3 **usb\_status\_t USB\_HostMsdeinit ( usb\_device\_handle *deviceHandle*, usb\_host\_class\_handle *classHandle* )**

This function frees the resource for the MSD instance.

Parameters

in	<i>deviceHandle</i>	The device handle.
in	<i>classHandle</i>	The class handle.

Return values

<i>kStatus_USB_Success</i>	The device is de-initialized successfully.
----------------------------	--------------------------------------------

#### 4.4.6.4 **usb\_status\_t USB\_HostMsMassStorageReset ( usb\_host\_class\_handle *classHandle*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the mass storage reset request.

Parameters

in	<i>classHandle</i>	The class handle.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

## USB MSC Class driver

<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

### 4.4.6.5 **usb\_status\_t USB\_HostMsdGetMaxLun ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *logicalUnitNumber*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the get maximum LUN request.

Parameters

in	<i>classHandle</i>	The class handle.
out	<i>logicalUnit-Number</i>	Return logical unit number value.
in	<i>callbackFn</i>	This callback is called after this function completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

### 4.4.6.6 **usb\_status\_t USB\_HostMsdRead10 ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint32\_t *blockAddress*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, uint32\_t *blockNumber*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI READ(10) command. This command requests that the UFI device transfer data to the host.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>blockNumber</i>	Read block number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.7 **usb\_status\_t USB\_HostMsdRead12 ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint32\_t *blockAddress*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, uint32\_t *blockNumber*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI READ(12) command and requests that the UFI device transfer data to the host.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.

## USB MSC Class driver

in	<i>blockAddress</i>	The start block address.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>blockNumber</i>	Read block number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.8 **usb\_status\_t USB\_HostMsdWrite10 ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint32\_t *blockAddress*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, uint32\_t *blockNumber*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI WRITE(10) command and requests that the UFI device write the data transferred by the host to the medium.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
in	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.

in	<i>blockNumber</i>	Write block number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.9 **usb\_status\_t USB\_HostMsdWrite12 ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint32\_t *blockAddress*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, uint32\_t *blockNumber*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI WRITE(12) command and requests that the UFI device write the data transferred by the host to the medium.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
in	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>blockNumber</i>	Write block number.
in	<i>callbackFn</i>	This callback is called after this command completes.

## USB MSC Class driver

in	<i>callbackParam</i>	The first parameter in the callback function.
----	----------------------	-----------------------------------------------

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.10 **usb\_status\_t USB\_HostMsReadCapacity ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI READ CAPACITY command and allows the host to request capacities of the currently installed medium.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.11 **usb\_status\_t USB\_HostMsdTestUnitReady ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI TEST UNIT READY command and checks if the UFI device is ready.

Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.12 **usb\_status\_t USB\_HostMsdRequestSense ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI REQUEST SENSE command, this command instructs the UFI device to transfer sense data to the host for the specified logical unit.

## USB MSC Class driver

### Parameters

in	<i>classHandle</i>	the class MSD handle.
in	<i>logicalUnit</i>	logical unit number.
out	<i>buffer</i>	buffer pointer.
in	<i>bufferLength</i>	the buffer length.
in	<i>callbackFn</i>	this callback is called after this command completes.
in	<i>callbackParam</i>	the first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	callback return status, the command fail.

#### 4.4.6.13 **usb\_status\_t USB\_HostMsdModeSelect ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI MODE SELECT command and allows the host to specify medium or device parameters to the UFI device.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.



in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.14 **usb\_status\_t USB\_HostMsdModeSense ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t *pageControl*, uint8\_t *pageCode*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI MODE SENSE command and allows the UFI device to report medium or device parameters to the host.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>pageControl</i>	The page control field specifies the type of mode parameters to return.
in	<i>pageCode</i>	Buffer pointer.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

## USB MSC Class driver

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

### 4.4.6.15 **usb\_status\_t USB\_HostMsdInquiry ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI INQUIRY command and requests that information regarding parameters of the UFI device itself be sent to the host.

#### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

#### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.

<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.16 **usb\_status\_t USB\_HostMsdReadFormatCapacities ( usb\_host\_class\_handle classHandle, uint8\_t logicalUnit, uint8\_t \* buffer, uint32\_t bufferLength, transfer\_callback\_t callbackFn, void \* callbackParam )**

This function implements the UFI READ FORMAT CAPACITIES command and allows the host to request a list of the possible capacities that can be formatted on the currently installed medium.

##### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
out	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

##### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	callback return status, the command fail.

**4.4.6.17** `usb_status_t USB_HostMsdFormatUnit ( usb_host_class_handle classHandle,  
uint8_t logicalUnit, uint8_t trackNumber, uint16_t interLeave, uint8_t * buffer,  
uint32_t bufferLength, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the UFI FORMAT UNIT command and the host sends this command to physically format one track of a diskette according to the selected options.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>trackNumber</i>	This specifies which track is to be formatted.
in	<i>interLeave</i>	This specifies the interleave that shall be used for formatting.
in	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.18 **usb\_status\_t USB\_HostMsdPreventAllowRemoval ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t *prevent*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI PREVENT-ALLOW MEDIUM REMOVAL command and notifies the FUI device to enable or disable the removal of the medium in the logical unit.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.

## USB MSC Class driver

in	<i>prevent</i>	Prevent or allow <ul style="list-style-type: none"><li>• 0: enable (allow) the removal of the medium</li><li>• 1: disable (prevent) removal of the medium</li></ul>
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

**4.4.6.19** `usb_status_t USB_HostMsdWriteAndVerify ( usb_host_class_handle classHandle, uint8_t logicalUnit, uint32_t blockAddress, uint8_t * buffer, uint32_t bufferLength, uint32_t blockNumber, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the UFI WRITE AND VERIFY command and requests that the UFI device writes the data transferred by the host to the medium, then verifies the data on the medium.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
in	<i>buffer</i>	Buffer pointer.
in	<i>bufferLength</i>	The buffer length.

in	<i>blockNumber</i>	Write and verify block number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.20 **usb\_status\_t USB\_HostMsdStartStopUnit ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint8\_t *loadEject*, uint8\_t *start*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI START-STOP UNIT command and instructs the UFI device to enable or disable media access operations .

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>loadEject</i>	A Load Eject (LoEj) bit of zero requests that no eject action be performed. A LoEj bit of one, with the Start bit cleared to zero, which instructs the UFI device to eject the media.
in	<i>start</i>	A Start bit of one instructs the UFI device to enable media access operations. A Start bit of zero instructs the UFI device to disable media access operations.

## USB MSC Class driver

in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.21 **usb\_status\_t USB\_HostMsdVerify ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, uint32\_t *blockAddress*, uint16\_t *verificationLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI VERIFY command and requests that the UFI device verify the data on the medium.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
in	<i>verification-Length</i>	The data length that need to be verified.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
----------------------------	-----------------------------------------



<i>kStatus_USB_Invalid_Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

#### 4.4.6.22 **usb\_status\_t USB\_HostMsdRezeroUnit ( usb\_host\_class\_handle *classHandle*, uint8\_t *logicalUnit*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the UFI REZERO UNIT command. This command positions the head of the drive to the cylinder 0.

Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid_Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.

## USB MSC Class driver

<i>kStatus_USB_Error</i>	Callback return status, the command fail.
--------------------------	-------------------------------------------

**4.4.6.23** `usb_status_t USB_HostMsdSeek10 ( usb_host_class_handle classHandle, uint8_t logicalUnit, uint32_t blockAddress, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the UFI SEEK(10) command and requests that the UFI device seek to the specified Logical Block Address.

### Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>blockAddress</i>	The start block address.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	Callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	Callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	Callback return status, the command fail.

**4.4.6.24** `usb_status_t USB_HostMsdSendDiagnostic ( usb_host_class_handle classHandle, uint8_t logicalUnit, uint8_t selfTest, transfer_callback_t callbackFn, void * callbackParam )`

This function implements the UFI SEND DIAGNOSTIC command. This command requests the UFI device to do a reset or perform a self-test.

## Parameters

in	<i>classHandle</i>	The class MSD handle.
in	<i>logicalUnit</i>	Logical unit number.
in	<i>selfTest</i>	0 = perform special diagnostic test; 1 = perform default self-test.
in	<i>callbackFn</i>	This callback is called after this command completes.
in	<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	The previous command is executing or there is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSend/USB_HostRecv.
<i>kStatus_USB_Success</i>	callback return status, the command succeed.
<i>kStatus_USB_MSD-StatusFail</i>	callback return status, the CSW status indicate this command fail.
<i>kStatus_USB_Error</i>	callback return status, the command fail.

## **4.5 USB AUDIO Class driver**

### **4.5.1 Overview**

The audio device class definition applies to all devices or functions embedded in composite devices that are used to manipulate audio, voice, and sound-related functionality. This includes both audio data (analog and digital) and the functionality that is used to directly control the audio environment, such as volume and tone Control. Typical examples of audio class devices include the USB audio speaker. This section describes the programming interface of the USB HOST audio class driver. The USB HOST audio class driver handles the specific control requests for audio class and transfers data to and from the device through the isochronous pipe.

### **4.5.2 USB Host audio Initialization**

When audio device is attached, audio initialization occurs as follows:

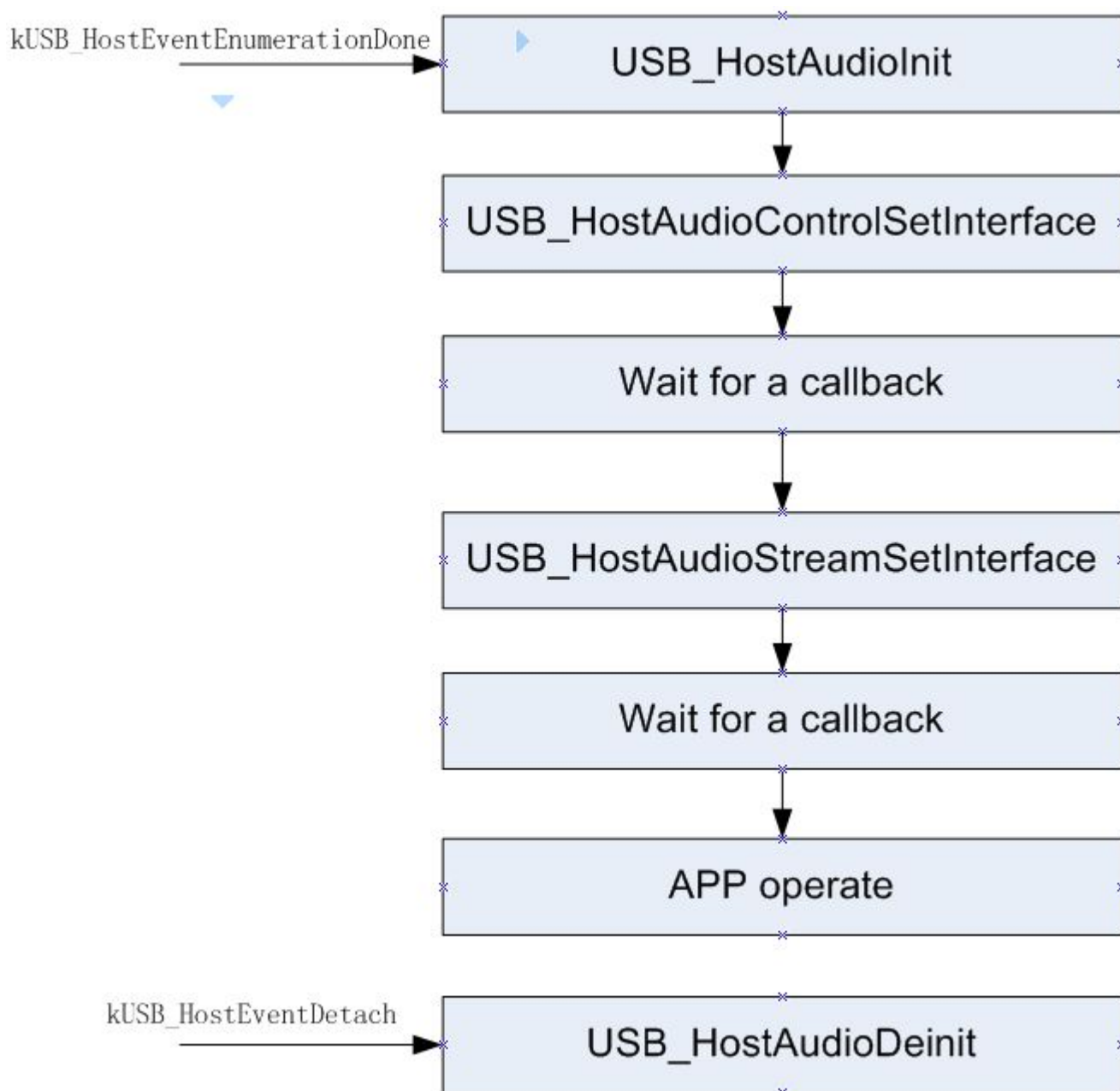


Figure 4.5.1: Host Audio Initialization

The above figure describes the following steps:

- Call the `USB_HostAudioInit` to initialize audio class instance `audio_instance_t` and the return class handle pointer to the audio class instance. The driver uses an instantiation of the `audio_instance_t` structure to maintain the current state of a audio instance module driver. This structure holds the USB host handle, the USB device handle, and keeps track of transfer information, alternate setting, pipes and interfaces that are enumerated for attached audio device.
- Call the `USB_HostAudioControlSetInterface` to set the audio class control interface, which opens

## USB AUDIO Class driver

- the interface's pipes.
- Wait the last step operation callback.
- Call the `USB_HostAudioStreamSetInterface` to set the audio class stream interface, which opens the interface's pipes.
- Wait the last step operation callback.
- Call the `USB_HostAudioStreamRecv` to receive isochronous data from the device, or call `USB_HostAudioStreamSend` to send isochronous data to the device.
- Wait the last step operation callback.
- Process data and receive or send again.

### 4.5.3 USB Host audio De-initialization

An application can call the `usb_host_audio_deinit` to deinitialize audio. This function cancels the transfer, closes the pipe, and releases the audio class instance.

There are two use cases when calling this function:

- The audio device is detached and this function is called to free the resource.
- The application calls this function and calls the `USB_HostAudioInit` to reinitialize the audio class.

### 4.5.4 USB Host audio Send data

Provides the buffer pointer, buffer length, the callback function, and the callback parameter and call the `USB_HostAudioStreamSend` to start asynchronous sending. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

### 4.5.5 USB Host audio Receive data

Provides the buffer pointer, buffer length, the callback function, and the callback parameter and calls the `USB_HostAudioStreamRecv` to start asynchronous receiving. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

## Data Structures

- struct `usb_audio_ctrl_header_desc_t`  
*Audio control interface header descriptor structure. [More...](#)*
- struct `usb_audio_ctrl_it_desc_t`  
*Audio control interface input terminal descriptor structure. [More...](#)*
- struct `usb_audio_ctrl_ot_desc_t`  
*Audio control interface output terminal descriptor structure. [More...](#)*
- struct `usb_audio_ctrl_fu_desc_t`  
*Audio control interface feature unit descriptor structure. [More...](#)*
- struct `usb_audio_stream_specific_iso_endp_desc_t`

- *Audio as isochronous audio data endpoint descriptor structure. [More...](#)*
- struct `usb_audio_stream_synch_endp_desc_t`  
*Audio standard as isochronous synch endpoint descriptor structure. [More...](#)*
- struct `usb_audio_stream_spepific_as_intf_desc_t`  
*Audio class-specific as interface descriptor structure. [More...](#)*
- struct `usb_audio_stream_format_type_desc_t`  
*audio Format type descriptor structure [More...](#)*
- struct `audio_instance_t`  
*Audio instance structure and audio usb\_host\_class\_handle pointer to this structure. [More...](#)*

## Macros

- #define `USB_AUDIO_CLASS_CODE` 1  
*Audio class code.*
- #define `USB_AUDIO_SUBCLASS_CODE_CONTROL` 1  
*Audio class control interface code.*
- #define `USB_AUDIO_SUBCLASS_CODE_AUDIOSTREAMING` 2  
*Audio class stream interface code.*
- #define `USB_AUDIO_GET_CUR_MUTE` 0x80  
*AUDIO class-specific feature unit get current mute command.*
- #define `USB_AUDIO_SET_CUR_MUTE` 0x00  
*AUDIO class-specific feature unit set current mute command.*
- #define `USB_AUDIO_GET_CUR_VOLUME` 0x81  
*AUDIO class-specific feature unit get current volume command.*
- #define `USB_AUDIO_SET_CUR_VOLUME` 0x01  
*AUDIO class-specific feature unit set current volume command.*
- #define `USB_AUDIO_GET_MIN_VOLUME` 0x82  
*AUDIO class-specific feature unit get minimum volume command.*
- #define `USB_AUDIO_SET_MIN_VOLUME` 0x02  
*AUDIO class-specific feature unit set minimum volume command.*
- #define `USB_AUDIO_GET_MAX_VOLUME` 0x83  
*AUDIO class-specific feature unit get maximum volume command.*
- #define `USB_AUDIO_SET_MAX_VOLUME` 0x03  
*AUDIO class-specific feature unit set maximum volume command.*
- #define `USB_AUDIO_GET_RES_VOLUME` 0x84  
*AUDIO class-specific feature unit get resolution volume command.*
- #define `USB_AUDIO_SET_RES_VOLUME` 0x04  
*AUDIO class-specific feature unit set resolution volume command.*
- #define `USB_AUDIO_GET_CUR_PITCH` 0x80  
*AUDIO class-specific endpoint get current pitch control command.*
- #define `USB_AUDIO_SET_CUR_PITCH` 0x00  
*AUDIO class-specific endpoint set current pitch control command.*
- #define `USB_AUDIO_GET_CUR_SAMPLING_FREQ` 0x81  
*AUDIO class-specific endpoint get current sampling frequency command.*
- #define `USB_AUDIO_SET_CUR_SAMPLING_FREQ` 0x01  
*AUDIO class-specific endpoint set current sampling frequency command.*
- #define `USB_AUDIO_GET_MIN_SAMPLING_FREQ` 0x82  
*AUDIO class-specific endpoint get minimum sampling frequency command.*
- #define `USB_AUDIO_SET_MIN_SAMPLING_FREQ` 0x02  
*AUDIO class-specific endpoint set minimum sampling frequency command.*

## USB AUDIO Class driver

- #define `USB_AUDIO_GET_MAX_SAMPLING_FREQ` 0x83  
*AUDIO class-specific endpoint get maximum sampling frequency command.*
- #define `USB_AUDIO_SET_MAX_SAMPLING_FREQ` 0x03  
*AUDIO class-specific endpoint set maximum sampling frequency command.*
- #define `USB_AUDIO_GET_RES_SAMPLING_FREQ` 0x84  
*AUDIO class-specific endpoint get resolution sampling frequency command.*
- #define `USB_AUDIO_SET_RES_SAMPLING_FREQ` 0x04  
*AUDIO class-specific endpoint set resolution sampling frequency command.*

## USB host audio class APIs

- `usb_status_t USB_HostAudioInit (usb_device_handle deviceHandle, usb_host_class_handle *classHandlePtr)`  
*Initializes the audio instance.*
- `usb_status_t USB_HostAudioDeinit (usb_device_handle deviceHandle, usb_host_class_handle classHandle)`  
*Deinitializes the Audio instance.*
- `usb_status_t USB_HostAudioStreamSetInterface (usb_host_class_handle classHandle, usb_host_interface_handle interfaceHandle, uint8_t alternateSetting, transfer_callback_t callbackFn, void *callbackParam)`  
*Sets the audio class stream interface.*
- `usb_status_t USB_HostAudioControlSetInterface (usb_host_class_handle classHandle, usb_host_interface_handle interfaceHandle, uint8_t alternateSetting, transfer_callback_t callbackFn, void *callbackParam)`  
*Sets the audio class control interface.*
- `uint16_t USB_HostAudioPacketSize (usb_host_class_handle classHandle, uint8_t pipeType, uint8_t direction)`  
*Gets the pipe maximum packet size.*
- `usb_status_t USB_HostAudioStreamRecv (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLen, transfer_callback_t callbackFn, void *callbackParam)`  
*Audio stream receive data.*
- `usb_status_t USB_HostAudioStreamSend (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLen, transfer_callback_t callbackFn, void *callbackParam)`  
*Audio stream send data.*
- `usb_status_t USB_HostAudioStreamGetCurrentAltsettingDescriptors (usb_host_class_handle classHandle, usb_audio_stream_specific_as_intf_desc_t **asIntfDesc, usb_audio_stream_format_type_desc_t **formatTypeDesc, usb_audio_stream_specific_iso_endp_desc_t **isoEndpDesc)`  
*Gets the audio stream current altsetting descriptor.*
- `usb_status_t USB_HostAudioFeatureUnitRequest (usb_host_class_handle classHandle, uint8_t channelNo, void *buf, uint32_t cmdCode, transfer_callback_t callbackFn, void *callbackParam)`  
*The USB audio feature unit request.*
- `usb_status_t USB_HostAudioEndpointRequest (usb_host_class_handle classHandle, void *buf, uint32_t cmdCode, transfer_callback_t callbackFn, void *callbackParam)`  
*The USB audio endpoint request.*



## 4.5.6 Data Structure Documentation

### 4.5.6.1 struct usb\_audio\_ctrl\_header\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the header descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of audio header descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of an audio header descriptor.*
- uint8\_t [bcdcdc](#) [2]  
*Audio Device Class Specification Release Number in Binary-Coded Decimal.*
- uint8\_t [wtotallength](#) [2]  
*Total number of bytes returned for the class-specific AudioControl interface descriptor.*
- uint8\_t [bincollection](#)  
*The number of AudioStreaming and MIDIStreaming interfaces in the Audio Interface Collection to which this AudioControl interface belongs to.*

#### 4.5.6.1.0.14 Field Documentation

##### 4.5.6.1.0.14.1 uint8\_t usb\_audio\_ctrl\_header\_desc\_t::wtotallength[2]

Includes the combined length of this descriptor header and all unit and terminal descriptors.

### 4.5.6.2 struct usb\_audio\_ctrl\_it\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the input terminal descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of audio input terminal descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of audio input terminal descriptor.*
- uint8\_t [bterminalid](#)  
*Constant uniquely identifying the Terminal within the audio function.*
- uint8\_t [wterminaltype](#) [2]  
*Constant characterizing the type of Terminal.*
- uint8\_t [bassocterminal](#)  
*ID of the Output Terminal to which this Input Terminal is associated.*
- uint8\_t [bnrchannels](#)  
*Number of logical output channels in the Terminal's output audio channel cluster.*
- uint8\_t [wchannelconfig](#) [2]  
*Describes the spatial location of the logical channels.*
- uint8\_t [ichannelnames](#)  
*Index of a string descriptor, describing the name of the first logical channel.*
- uint8\_t [iterminal](#)

## USB AUDIO Class driver

*Index of a string descriptor, describing the Input Terminal.*

### 4.5.6.2.0.15 Field Documentation

#### 4.5.6.2.0.15.1 uint8\_t usb\_audio\_ctrl\_it\_desc\_t::bterminalid

This value is used in all requests to address this Terminal

#### 4.5.6.2.0.15.2 uint8\_t usb\_audio\_ctrl\_it\_desc\_t::wchannelconfig[2]

### 4.5.6.3 struct usb\_audio\_ctrl\_ot\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the output terminal descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of audio output terminal descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of audio output terminal descriptor.*
- uint8\_t [bterminalid](#)  
*Constant uniquely identifying the Terminal within the audio function.*
- uint8\_t [wterminaltype](#) [2]  
*Constant characterizing the type of Terminal.*
- uint8\_t [bassocterminal](#)  
*Constant, identifying the Input Terminal to which this Output Terminal is associated.*
- uint8\_t [bsourceid](#)  
*ID of the Unit or Terminal to which this Terminal is connected.*
- uint8\_t [itterminal](#)  
*Index of a string descriptor, describing the Output Terminal.*

### 4.5.6.3.0.16 Field Documentation

#### 4.5.6.3.0.16.1 uint8\_t usb\_audio\_ctrl\_ot\_desc\_t::bterminalid

This value is used in all requests to address this Terminal

### 4.5.6.4 struct usb\_audio\_ctrl\_fu\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the output terminal descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of audio output terminal descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of audio output terminal descriptor.*
- uint8\_t [bunitid](#)  
*Constant uniquely identifying the unit within the audio function.*
- uint8\_t [bsourceid](#)

- *ID of the Unit or Terminal to which this Feature Unit is connected.*  
uint8\_t [bcontrolsiz](#)  
*Size in bytes of an element of the bmaControls.*

#### 4.5.6.4.0.17 Field Documentation

##### 4.5.6.4.0.17.1 uint8\_t usb\_audio\_ctrl\_fu\_desc\_t::bunitid

This value is used in all requests to address this unit

#### 4.5.6.5 struct usb\_audio\_stream\_specific\_iso\_endp\_desc\_t

##### Data Fields

- uint8\_t [blength](#)  
*Total size of the descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of the descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of the descriptor.*
- uint8\_t [bmattributes](#)  
*A bit in the range D6..0 set to 1 indicates that the mentioned Control is supported by this endpoint.*
- uint8\_t [blockdelayunits](#)  
*Indicates the units used for the wLockDelay field.*
- uint8\_t [wlockdelay](#) [2]  
*Indicates the time it takes this endpoint to reliably lock its internal clock recovery circuitry.*

#### 4.5.6.5.0.18 Field Documentation

##### 4.5.6.5.0.18.1 uint8\_t usb\_audio\_stream\_specific\_iso\_endp\_desc\_t::wlockdelay[2]

Units used depend on the value of the bLockDelayUnits field.

#### 4.5.6.6 struct usb\_audio\_stream\_synch\_endp\_desc\_t

##### Data Fields

- uint8\_t [blength](#)  
*Total size of the descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of the endpoint descriptor.*
- uint8\_t [bendpointaddress](#)  
*The address of the endpoint on the USB device described by this descriptor.*
- uint8\_t [bmattributes](#)  
*D3..2: Synchronization type, D1..0: Transfer type.*
- uint8\_t [wmaxpacketsize](#) [2]  
*Maximum packet size this endpoint is capable of sending or receiving when this configuration is selected.*
- uint8\_t [binterval](#)  
*Interval for polling endpoint for data transfers expressed in milliseconds.*

## USB AUDIO Class driver

- uint8\_t [brefresh](#)  
*This field indicates the rate at which an isochronous synchronization pipe provides new synchronization feedback data.*
- uint8\_t [bsynchaddress](#)  
*Must be reset to zero.*

### 4.5.6.7 struct usb\_audio\_stream\_spepific\_as\_intf\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of the descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of the descriptor.*
- uint8\_t [bterminallink](#)  
*The Terminal ID of the Terminal to which the endpoint of this interface is connected.*
- uint8\_t [bdelay](#)  
*Expressed in number of frames.*
- uint8\_t [wformattag](#) [2]  
*The Audio Data Format that has to be used to communicate with this interface.*

### 4.5.6.8 struct usb\_audio\_stream\_format\_type\_desc\_t

#### Data Fields

- uint8\_t [blength](#)  
*Total size of the descriptor.*
- uint8\_t [bdescriptortype](#)  
*Descriptor type of the descriptor.*
- uint8\_t [bdescriptorsubtype](#)  
*Subtype of the descriptor.*
- uint8\_t [bformattype](#)  
*Constant identifying the Format Type the AudioStreaming interface is using.*
- uint8\_t [bnrchannels](#)  
*Number of channels of device.*
- uint8\_t [bsubframesize](#)  
*Bytes per audio subframe.*
- uint8\_t [bbitresolution](#)  
*Bits per sample.*
- uint8\_t [bsamfreqtype](#)  
*Frequency supported.*
- uint8\_t [tsamfreq](#) [1][3]  
*Sample frequency.*

#### 4.5.6.9 struct audio\_instance\_t

##### Data Fields

- [usb\\_host\\_handle](#) hostHandle  
*This instance's related host handle.*
- [usb\\_device\\_handle](#) deviceHandle  
*This instance's related device handle.*
- [usb\\_host\\_interface\\_handle](#) streamIntfHandle  
*This instance's audio stream interface handle.*
- [usb\\_host\\_interface\\_handle](#) controlIntfHandle  
*This instance's control stream interface handle.*
- [usb\\_audio\\_stream\\_specific\\_as\\_intf\\_desc\\_t](#) \* asIntfDesc  
*Audio class class-specific as interface descriptor pointer.*
- [usb\\_audio\\_stream\\_format\\_type\\_desc\\_t](#) \* formatTypeDesc  
*Audio class class-specific format type descriptor pointer.*
- [usb\\_audio\\_stream\\_specific\\_iso\\_endp\\_desc\\_t](#) \* isoEndpDesc  
*Audio class class-specific ISO audio data endpoint descriptor pointer.*
- [usb\\_host\\_pipe\\_handle](#) isoInPipe  
*Audio class ISO in pipe.*
- [usb\\_host\\_pipe\\_handle](#) isoOutPipe  
*Audio class ISO out pipe.*
- [transfer\\_callback\\_t](#) inCallbackFn  
*Audio class ISO in transfer callback function.*
- void \* [inCallbackParam](#)  
*Audio class ISO in transfer callback parameter.*
- [transfer\\_callback\\_t](#) outCallbackFn  
*Audio class ISO out transfer callback function.*
- void \* [outCallbackParam](#)  
*Audio class ISO out transfer callback function.*
- [usb\\_audio\\_ctrl\\_header\\_desc\\_t](#) \* headerDesc  
*Audio class header descriptor pointer.*
- [usb\\_audio\\_ctrl\\_it\\_desc\\_t](#) \* itDesc  
*Audio class input terminal descriptor pointer.*
- [usb\\_audio\\_ctrl\\_ot\\_desc\\_t](#) \* otDesc  
*Audio class output terminal descriptor pointer.*
- [usb\\_audio\\_ctrl\\_fu\\_desc\\_t](#) \* fuDesc  
*Audio class feature unit descriptor pointer.*
- [usb\\_host\\_pipe\\_handle](#) controlPipe  
*Audio class device control pipe.*
- [transfer\\_callback\\_t](#) controlCallbackFn  
*Audio control transfer callback function.*
- void \* [controlCallbackParam](#)  
*Audio control transfer callback function.*
- [usb\\_host\\_transfer\\_t](#) \* [controlTransfer](#)  
*On-going control transfer.*
- uint16\_t [inPacketSize](#)  
*Audio ISO in maximum packet size.*
- uint16\_t [outPacketSize](#)  
*Audio ISO out maximum packet size.*
- uint8\_t [isSetup](#)

## USB AUDIO Class driver

- *Whether the audio setup transfer is transmitting.*  
uint8\_t [isoEpNum](#)  
*Audio stream ISO endpoint number.*
- uint8\_t [streamIfnum](#)  
*Audio stream ISO interface number.*

### 4.5.7 Function Documentation

#### 4.5.7.1 usb\_status\_t USB\_HostAudioInit ( usb\_device\_handle *deviceHandle*, usb\_host\_class\_handle \* *classHandlePtr* )

This function allocates the resource for the audio instance.

Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandlePtr</i>	Return class handle.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory fail.

#### 4.5.7.2 usb\_status\_t USB\_HostAudioDeinit ( usb\_device\_handle *deviceHandle*, usb\_host\_class\_handle *classHandle* )

This function release the resource for audio instance.

Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandle</i>	The class handle.

Return values

<i>kStatus_USB_Success</i>	The device is deinitialized successfully.
----------------------------	-------------------------------------------

#### 4.5.7.3 usb\_status\_t USB\_HostAudioStreamSetInterface ( usb\_host\_class\_handle *classHandle*, usb\_host\_interface\_handle *interfaceHandle*, uint8\_t *alternateSetting*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )

This function binds the interface with the audio instance.

## Parameters

<i>classHandle</i>	The class handle.
<i>interface-Handle</i>	The interface handle.
<i>alternate-Setting</i>	The alternate setting value.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.
<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

#### 4.5.7.4 **usb\_status\_t USB\_HostAudioControlSetInterface ( usb\_host\_class\_handle *classHandle*, usb\_host\_interface\_handle *interfaceHandle*, uint8\_t *alternateSetting*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function binds the interface with the audio instance.

## Parameters

<i>classHandle</i>	The class handle.
<i>interface-Handle</i>	The interface handle.

## USB AUDIO Class driver

<i>alternate-Setting</i>	The alternate setting value.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.
<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See USB_HostOpenPipe.

### 4.5.7.5 uint16\_t USB\_HostAudioPacketSize ( usb\_host\_class\_handle *classHandle*, uint8\_t *pipeType*, uint8\_t *direction* )

#### Parameters

<i>classHandle</i>	The class handle.
<i>pipeType</i>	Its value is USB_ENDPOINT_CONTROL, USB_ENDPOINT_ISOCHRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTERRUPT. See the usb_spec.h
<i>direction</i>	Pipe direction.

### Return values

<i>0</i>	The classHandle is NULL.
<i>max</i>	Packet size.

### 4.5.7.6 usb\_status\_t USB\_HostAudioStreamRecv ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *buffer*, uint32\_t *bufferLen*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )

This function implements the audio receiving data.



## Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLen</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostRecv.

#### 4.5.7.7 **usb\_status\_t USB\_HostAudioStreamSend ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *buffer*, uint32\_t *bufferLen*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the audio sending data.

## Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLen</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

## USB AUDIO Class driver

<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

### 4.5.7.8 **usb\_status\_t USB\_HostAudioStreamGetCurrentAltsettingDescriptors ( usb\_host\_class\_handle *classHandle*, usb\_audio\_stream\_spepific\_as\_intf\_desc\_t \*\* *asIntfDesc*, usb\_audio\_stream\_format\_type\_desc\_t \*\* *formatTypeDesc*, usb\_audio\_stream\_specific\_iso\_endp\_desc\_t \*\* *isoEndpDesc* )**

This function implements the get audio stream current altsetting descriptor.

Parameters

<i>classHandle</i>	The class handle.
<i>asIntfDesc</i>	The pointer of class specific AS interface descriptor.
<i>formatTypeDesc</i>	The pointer of format type descriptor.
<i>isoEndpDesc</i>	The pointer of specific ISO endp descriptor.

Return values

<i>kStatus_USB_Success</i>	Get the audio stream current altsetting descriptor request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

### 4.5.7.9 **usb\_status\_t USB\_HostAudioFeatureUnitRequest ( usb\_host\_class\_handle *classHandle*, uint8\_t *channelNo*, void \* *buf*, uint32\_t *cmdCode*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the USB audio feature unit request.

Parameters

<i>classHandle</i>	The class handle.
<i>channelNo</i>	The channel number of audio feature unit.
<i>buf</i>	The feature unit request buffer pointer.

<i>cmdCode</i>	The feature unit command code, for example USB_AUDIO_GET_CUR_MUTE, and so on.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

#### Return values

<i>kStatus_USB_Success</i>	Feature unit request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

#### 4.5.7.10 **usb\_status\_t USB\_HostAudioEndpointRequest ( usb\_host\_class\_handle classHandle, void \* buf, uint32\_t cmdCode, transfer\_callback\_t callbackFn, void \* callbackParam )**

This function implements the USB audio endpoint request.

#### Parameters

<i>classHandle</i>	The class handle.
<i>buf</i>	The feature unit buffer pointer.
<i>cmdCode</i>	The feature unit command code, for example USB_AUDIO_GET_CUR_PITCH, and so on.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

#### Return values

<i>kStatus_USB_Success</i>	Endpoint request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.

## USB AUDIO Class driver

<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.

## 4.6 USB PHDC Class driver

### 4.6.1 Overview

The USB Personal Healthcare Device Class (or USB PHDC) defines personal healthcare devices such as weight scales, thermometers, blood pressure meters, glucose meters, and pulse oximeters. This section describes the programming interface of the USB Host PHDC class driver. The USB Host PHDC class driver handles the specific control requests for the PHDC class and transfers data to and from the device through the interrupt and bulk pipes.

### 4.6.2 USB Host PHDC Initialization

When the personal healthcare device is attached, the PHDC initialization flow is as follows:

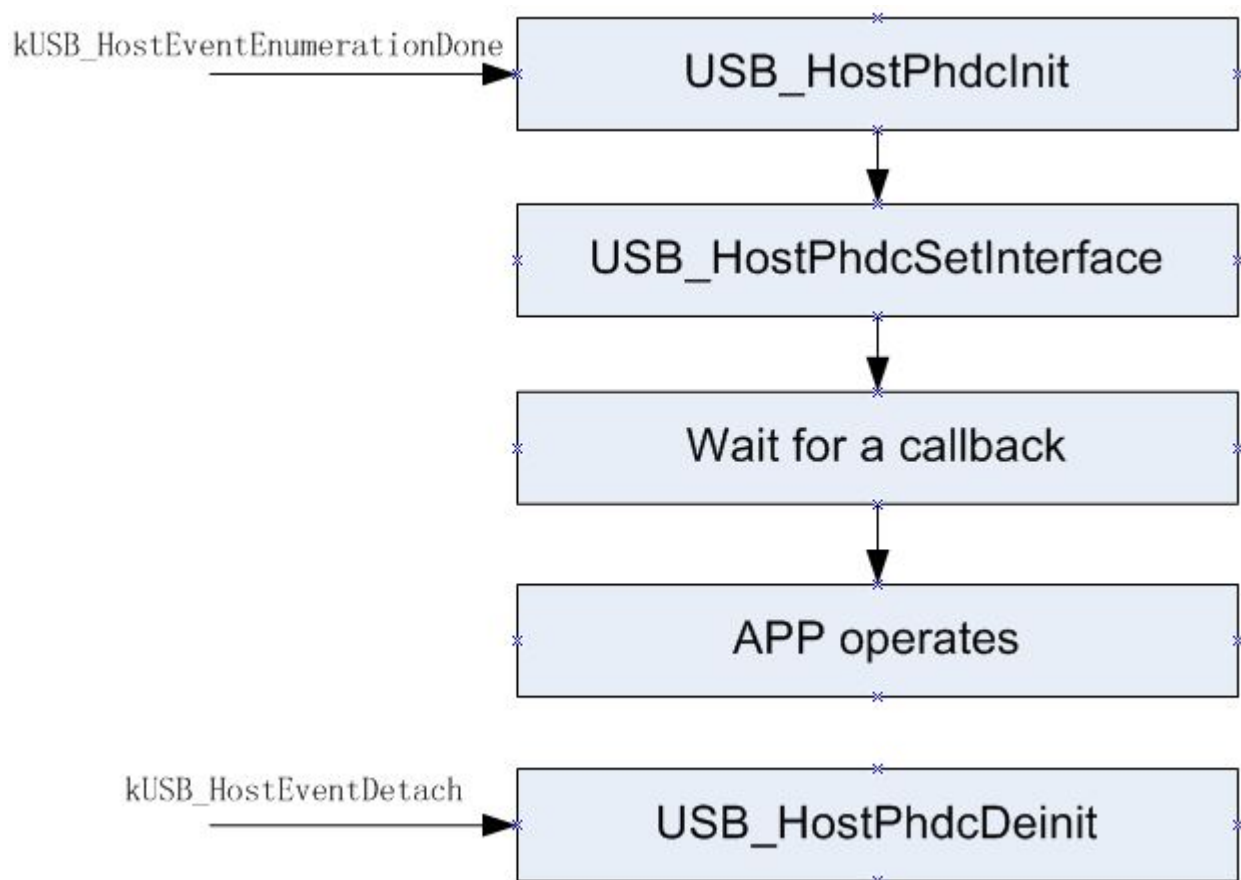


Figure 4.6.1: Host PHDC Initialization

The above figure describes the following steps:

- Call the `USB_HostPhdcInit` to initialize the PHDC class instance `usb_host_phdc_instance_t` and the return class handle pointer to the PHDC class instance. The driver uses an instantiation of

## USB PHDC Class driver

the [usb\\_host\\_phdc\\_instance\\_t](#) structure to maintain the current state of a PHDC instance module driver. This structure holds the USB host handle and the USB device handle and keeps track of transfer information, alternate setting, pipes and interfaces that are enumerated for attached personal healthcare device.

- Call the `USB_HostPhdcSetInterface` to sets the PHDC class interface, which opens the interface's pipes.
- Wait the last step operation callback.
- Call the `USB_HostPhdcRecv` to receive data from device, or call the `USB_HostPhdcSend` to send data to device.
- Wait the last step operation callback.
- Process data and receive or send again.

### 4.6.3 USB Host PHDC Deinitialization

An application can call the `usb_host_phdc_deinit` to deinitialize the PHDC. This function cancels the transfer, closes the pipe, and releases the PHDC class instance.

There are two use cases to call this function:

- A personal healthcare device is detached and this function is called to free the resource.
- An application calls this function and then calls `USBHostPhdcInit` to re-initialize the PHDC class.

### 4.6.4 USB Host PHDC Send data

Provides the buffer pointer, the buffer length, the callback function, the callback parameter and calls the `USB_HostPhdcSend` function to start asynchronous sending. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

### 4.6.5 USB Host PHDC Receive data

Provides the buffer pointer, the buffer length, the callback function, the callback parameter and call the `USB_HostPhdcRecv` function to start asynchronous receiving. Then, the callback function is called with one transfer status parameter when the transfer succeeds or fails.

## Data Structures

- struct [usb\\_host\\_phdc\\_class\\_function\\_descriptor\\_t](#)  
*PHDC class function descriptor structure as defined by the PHDC class specification. [More...](#)*
- struct [usb\\_host\\_phdc\\_function\\_extension\\_descriptor\\_t](#)  
*Function extension descriptor (device specialization) structure as defined by the PHDC class specification. [More...](#)*
- struct [usb\\_host\\_phdc\\_qos\\_descriptor\\_t](#)  
*QoS descriptor structure as defined by the PHDC class specification. [More...](#)*

- struct `usb_host_phdc_metadata_descriptor_t`  
*Metadata descriptor structure as defined by the PHDC class specification. [More...](#)*
- struct `usb_host_phdc_metadata_preamble_t`  
*Metadata message preamble structure as defined by the PHDC class specification. [More...](#)*
- struct `usb_host_phdc_instance_t`  
*PHDC instance structure. [More...](#)*

## Macros

- #define `USB_HOST_PHDC_CLASS_CODE` (0x0FU)  
*PHDC class code.*
- #define `USB_HOST_PHDC_SUBCLASS_CODE` (0x00U)  
*PHDC sub class code.*
- #define `USB_HOST_PHDC_PROTOCOL` (0x00U)  
*PHDC protocol.*
- #define `USB_HOST_PHDC_GET_STATUS_REQUEST` (0x00U)  
*PHDC get status request.*
- #define `USB_HOST_PHDC_SET_FEATURE_REQUEST` (0x03U)  
*PHDC set feature request.*
- #define `USB_HOST_PHDC_CLEAR_FEATURE_REQUEST` (0x01U)  
*PHDC clear feature request.*
- #define `USB_HOST_PHDC_FEATURE_METADATA` (0x01U)  
*PHDC meta-data feature.*
- #define `USB_HOST_PHDC_QOS_ENCODING_VERSION` (0x01U)  
*PHDC QoS information encoding feature.*
- #define `USB_HOST_PHDC_MESSAGE_PREAMBLE_SIGNATURE_SIZE` (0x10U)  
*meta-data message preamble signature size*
- #define `USB_HOST_PHDC_CLASSFUNCTION_DESCRIPTOR` (0x20U)  
*PHDC class function descriptor type.*
- #define `USB_HOST_PHDC_QOS_DESCRIPTOR` (0x21U)  
*PHDC QoS descriptor type.*
- #define `USB_HOST_PHDC_11073PHD_FUNCTION_DESCRIPTOR` (0x30U)  
*PHDC function extension descriptor type.*
- #define `USB_HOST_PHDC_METADATA_DESCRIPTOR` (0x22U)  
*PHDC meta-data descriptor type.*

## USB host PHDC class APIs

- `usb_status_t USB_HostPhdcInit` (`usb_host_handle` deviceHandle, `usb_host_class_handle` \*classHandle)  
*Initializes the PHDC instance.*
- `usb_status_t USB_HostPhdcSetInterface` (`usb_host_class_handle` classHandle, `usb_host_interface_handle` interfaceHandle, `uint8_t` alternateSetting, `transfer_callback_t` callbackFn, void \*callbackParam)  
*Sets an interface.*
- `usb_status_t USB_HostPhdcDeinit` (`usb_host_handle` deviceHandle, `usb_host_class_handle` classHandle)  
*Deinitializes the PHDC instance.*

## USB PHDC Class driver

- `usb_status_t USB_HostPhdcRecv (usb_host_class_handle classHandle, uint8_t qos, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Receives data.*
- `usb_status_t USB_HostPhdcSend (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)`  
*Sends data.*
- `usb_status_t USB_HostPhdcSendControlRequest (usb_host_class_handle classHandle, uint8_t request, transfer_callback_t callbackFn, void *callbackParam)`  
*PHDC sends the control request.*
- `usb_status_t USB_HostPhdcSetClearFeatureEndpointHalt (usb_host_class_handle classHandle, uint8_t request, void *param, transfer_callback_t callbackFn, void *callbackParam)`  
*PHDC set and clear feature endpoint halt request.*
- `usb_host_ep_t * USB_HostPhdcGetEndpointInformation (usb_host_class_handle classHandle, uint8_t pipeType, uint8_t direction)`  
*USB\_HostPhdcGetEndpointInformation.*

## 4.6.6 Data Structure Documentation

### 4.6.6.1 struct usb\_host\_phdc\_class\_function\_descriptor\_t

#### Data Fields

- `uint8_t bLength`  
*Class function descriptor length.*
- `uint8_t bDescriptorType`  
*PHDC\_CLASSFUNCTION\_DESCRIPTOR type.*
- `uint8_t bPhdcDataCode`  
*Data/Messaging format code.*
- `uint8_t bmCapability`  
*If bit 0 is 1 the meta-data message preamble is implemented and 0 if it is not.*

### 4.6.6.2 struct usb\_host\_phdc\_function\_extension\_descriptor\_t

#### Data Fields

- `uint8_t bLength`  
*Function extension descriptor length.*
- `uint8_t bDescriptorType`  
*PHDC\_CLASSFUNCTION\_DESCRIPTOR type.*
- `uint8_t bReserved`  
*Reserved for future use.*
- `uint8_t bNumDevSpecs`  
*Number of wDevSpecializations.*
- `uint16_t * wDevSpecializations`  
*Variable length list that defines the device specialization.*



#### 4.6.6.3 struct usb\_host\_phdc\_qos\_descriptor\_t

##### Data Fields

- uint8\_t [bLength](#)  
*QoS descriptor length.*
- uint8\_t [bDescriptorType](#)  
*PHDC\_QOS\_DESCRIPTOR type.*
- uint8\_t [bQosEncodingVersion](#)  
*Version of QoS information encoding.*
- uint8\_t [bmLatencyReliability](#)  
*Latency/reliability bin for the QoS data.*

#### 4.6.6.4 struct usb\_host\_phdc\_metadata\_descriptor\_t

##### Data Fields

- uint8\_t [bLength](#)  
*Metadata descriptor length.*
- uint8\_t [bDescriptorType](#)  
*Descriptor type.*
- uint8\_t \* [bOpaqueData](#)  
*Opaque metadata.*

#### 4.6.6.5 struct usb\_host\_phdc\_metadata\_preamble\_t

##### Data Fields

- uint8\_t [aSignature](#) [USB\_HOST\_PHDC\_MESSAGE\_PREAMBLE\_SIGNATURE\_SIZE]  
*Constant used to give preamble verifiability.*
- uint8\_t [bNumberTransfers](#)  
*Count of following transfer to which the QoS setting applies.*
- uint8\_t [bQosEncodingVersion](#)  
*Version of QoS information encoding.*
- uint8\_t [bmLatencyReliability](#)  
*See latency/reliability bin for the QoS data.*
- uint8\_t [bOpaqueDataSize](#)  
*Opaque QoS data or meta-data size.*
- uint8\_t \* [bOpaqueData](#)  
*Opaque metadata.*

#### 4.6.6.6 struct usb\_host\_phdc\_instance\_t

##### Data Fields

- [usb\\_host\\_handle](#) hostHandle  
*The host handle.*
- [usb\\_device\\_handle](#) deviceHandle

## USB PHDC Class driver

- *The device handle.*  
`usb_host_interface_handle interfaceHandle`
- *The interface handle.*  
`usb_host_pipe_handle controlPipe`
- *The control pipe.*  
`usb_host_pipe_handle interruptPipe`
- *The interrupt pipe.*  
`usb_host_pipe_handle bulkInPipe`
- *The bulk in pipe.*  
`usb_host_pipe_handle bulkOutPipe`
- *The bulk out pipe.*  
`transfer_callback_t inCallbackFn`
- *The callback function is called when the PHDC receives complete.*  
`void * inCallbackParam`
- *The first parameter of the in callback function.*  
`transfer_callback_t outCallbackFn`
- *The callback function is called when the PHDC sends complete.*  
`void * outCallbackParam`
- *The first parameter of the out callback function.*  
`transfer_callback_t controlCallbackFn`
- *The control callback function.*  
`void * controlCallbackParam`
- *The first parameter of the control callback function.*  
`usb_host_transfer_t * controlTransfer`
- *The control transfer pointer.*  
`usb_host_ep_t interruptInEndpointInformation`
- *The interrupt in information.*  
`usb_host_ep_t bulkInEndpointInformation`
- *The bulk in information.*  
`usb_host_ep_t bulkOutEndpointInformation`
- *The bulk out information.*  
`uint8_t isMessagePreambleEnabled`
- *The flag is used to check the message preamble feature is enabled or not.*  
`uint8_t numberTransferBulkOut`
- *The number of transfer that follow Meta-data Message Preamble.*  
`uint8_t numberTransferBulkIn`
- *The number of transfer that follow Meta-data Message Preamble.*

### 4.6.7 Function Documentation

#### 4.6.7.1 `usb_status_t USB_HostPhdcInit ( usb_host_handle deviceHandle, usb_host_class_handle * classHandle )`

This function allocates the resource for PHDC instance.

## Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandle</i>	Return class handle.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_AllocFail</i>	Allocate memory fail.

**4.6.7.2** `usb_status_t USB_HostPhdcSetInterface ( usb_host_class_handle classHandle,  
usb_host_interface_handle interfaceHandle, uint8_t alternateSetting,  
transfer_callback_t callbackFn, void * callbackParam )`

This function binds the interface with the PHDC instance.

## Parameters

<i>classHandle</i>	The class handle.
<i>interface-Handle</i>	The interface handle.
<i>alternate-Setting</i>	The alternate setting value.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

## Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Send transfer fail. See the USB_HostSendSetup.
<i>kStatus_USB_Busy</i>	Callback return status, there is no idle pipe.

## USB PHDC Class driver

<i>kStatus_USB_Transfer-Stall</i>	Callback return status, the transfer is stalled by the device.
<i>kStatus_USB_Error</i>	Callback return status, open pipe fail. See the USB_HostOpenPipe.

### 4.6.7.3 **usb\_status\_t USB\_HostPhdcDeinit ( usb\_host\_handle *deviceHandle*, usb\_host\_class\_handle *classHandle* )**

This function frees the resource for the PHDC instance.

Parameters

<i>deviceHandle</i>	The device handle.
<i>classHandle</i>	The class handle.

Return values

<i>kStatus_USB_Success</i>	The device is deinitialized successfully.
----------------------------	-------------------------------------------

### 4.6.7.4 **usb\_status\_t USB\_HostPhdcRecv ( usb\_host\_class\_handle *classHandle*, uint8\_t *qos*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the PHDC receiving data.

Parameters

<i>classHandle</i>	The class handle.
<i>qos</i>	QoS of the data being received.
<i>buffer</i>	The buffer pointer.
<i>bufferLength</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	Receive request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostRecv.

#### 4.6.7.5 **usb\_status\_t USB\_HostPhdcSend ( usb\_host\_class\_handle *classHandle*, uint8\_t \* *buffer*, uint32\_t *bufferLength*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

This function implements the PHDC sending data.

Parameters

<i>classHandle</i>	The class handle.
<i>buffer</i>	The buffer pointer.
<i>bufferLength</i>	The buffer length.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

Return values

<i>kStatus_USB_Success</i>	Send request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

#### 4.6.7.6 **usb\_status\_t USB\_HostPhdcSendControlRequest ( usb\_host\_class\_handle *classHandle*, uint8\_t *request*, transfer\_callback\_t *callbackFn*, void \* *callbackParam* )**

Parameters

## USB PHDC Class driver

<i>classHandle</i>	The class handle.
<i>request</i>	Setup packet request.
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Send request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

**4.6.7.7** `usb_status_t USB_HostPhdcSetClearFeatureEndpointHalt ( usb_host_class_handle classHandle, uint8_t request, void * param, transfer_callback_t callbackFn, void * callbackParam )`

### Parameters

<i>classHandle</i>	The class handle.
<i>request</i>	Setup packet request.
<i>param</i>	Request parameter
<i>callbackFn</i>	This callback is called after this function completes.
<i>callbackParam</i>	The first parameter in the callback function.

### Return values

<i>kStatus_USB_Success</i>	Send request successfully.
<i>kStatus_USB_Invalid-Handle</i>	The classHandle is NULL pointer.
<i>kStatus_USB_Busy</i>	There is no idle transfer.
<i>kStatus_USB_Error</i>	Pipe is not initialized. Or, send transfer fail. See the USB_HostSend.

#### 4.6.7.8 `usb_host_ep_t*` **USB\_HostPhdcGetEndpointInformation** ( `usb_host_class_handle` *classHandle*, `uint8_t` *pipeType*, `uint8_t` *direction* )

This function returns the PHDC endpoint information structure, which contains an endpoint descriptor and an endpoint extended descriptor.

## USB PHDC Class driver

### Parameters

<i>classHandle</i>	The class handle.
<i>pipeType</i>	Pipe type.
<i>direction</i>	Pipe direction.

### Return values

<i>endpointReturn</i>	All input parameters are valid.
<i>NULL</i>	One or more input parameters are invalid.



## Chapter 5

# USB OS Adapter

### 5.1 Overview

The OS adapter (OSA) is used to wrap the differences between RTOSes and enable a USB stack with the same code base and behavior.

Note

OSA should not be used in the USB application. Therefore, from the USB application viewpoint, OSA is invisible.

### Macros

- #define **BIG\_ENDIAN** (0U)  
*Define big endian.*
- #define **LITTLE\_ENDIAN** (1U)  
*Define little endian.*
- #define **ENDIANNESS LITTLE\_ENDIAN**  
*Define current endian.*

### Typedefs

- typedef void \* **usb\_osa\_event\_handle**  
*Define USB OSA event handle.*
- typedef void \* **usb\_osa\_sem\_handle**  
*Define USB OSA semaphore handle.*
- typedef void \* **usb\_osa\_mutex\_handle**  
*Define USB OSA mutex handle.*
- typedef void \* **usb\_osa\_msgq\_handle**  
*Define USB OSA message queue handle.*

### Enumerations

- enum **usb\_osa\_status\_t** {  
    **kStatus\_USB\_OSA\_Success** = 0x00U,  
    **kStatus\_USB\_OSA\_Error**,  
    **kStatus\_USB\_OSA\_TimeOut** }  
*USB OSA error code.*
- enum **usb\_osa\_event\_mode\_t** {  
    **kUSB\_OsaEventManualClear** = 0U,  
    **kUSB\_OsaEventAutoClear** = 1U }  
*The event flags are cleared automatically or manually.*

### USB OSA Memory Management

- void \* [USB\\_OsaMemoryAllocate](#) (uint32\_t length)  
*Reserves the requested amount of memory in bytes.*
- void [USB\\_OsaMemoryFree](#) (void \*p)  
*Frees the memory previously reserved.*

### USB OSA Event

- [usb\\_osa\\_status\\_t USB\\_OsaEventCreate](#) ([usb\\_osa\\_event\\_handle](#) \*handle, uint32\_t flag)  
*Creates an event object with all flags cleared.*
- [usb\\_osa\\_status\\_t USB\\_OsaEventDestroy](#) ([usb\\_osa\\_event\\_handle](#) handle)  
*Destroys a created event object.*
- [usb\\_osa\\_status\\_t USB\\_OsaEventSet](#) ([usb\\_osa\\_event\\_handle](#) handle, uint32\_t bitMask)  
*Sets an event flag.*
- [usb\\_osa\\_status\\_t USB\\_OsaEventWait](#) ([usb\\_osa\\_event\\_handle](#) handle, uint32\_t bitMask, uint32\_t flag, uint32\_t timeout, uint32\_t \*bitSet)  
*Waits for an event flag.*
- [usb\\_osa\\_status\\_t USB\\_OsaEventCheck](#) ([usb\\_osa\\_event\\_handle](#) handle, uint32\_t bitMask, uint32\_t \*bitSet)  
*Checks an event flag.*
- [usb\\_osa\\_status\\_t USB\\_OsaEventClear](#) ([usb\\_osa\\_event\\_handle](#) handle, uint32\_t bitMask)  
*Clears an event flag.*

### USB OSA Semaphore

- [usb\\_osa\\_status\\_t USB\\_OsaSemCreate](#) ([usb\\_osa\\_sem\\_handle](#) \*handle, uint32\_t count)  
*Creates a semaphore with a given value.*
- [usb\\_osa\\_status\\_t USB\\_OsaSemDestroy](#) ([usb\\_osa\\_sem\\_handle](#) handle)  
*Destroys a semaphore object.*
- [usb\\_osa\\_status\\_t USB\\_OsaSemPost](#) ([usb\\_osa\\_sem\\_handle](#) handle)  
*Posts a semaphore.*
- [usb\\_osa\\_status\\_t USB\\_OsaSemWait](#) ([usb\\_osa\\_sem\\_handle](#) handle, uint32\_t timeout)  
*Waits on a semaphore.*

### USB OSA Mutex

- [usb\\_osa\\_status\\_t USB\\_OsaMutexCreate](#) ([usb\\_osa\\_mutex\\_handle](#) \*handle)  
*Creates a mutex.*
- [usb\\_osa\\_status\\_t USB\\_OsaMutexDestroy](#) ([usb\\_osa\\_mutex\\_handle](#) handle)  
*Destroys a mutex.*
- [usb\\_osa\\_status\\_t USB\\_OsaMutexLock](#) ([usb\\_osa\\_mutex\\_handle](#) handle)  
*Waits for a mutex and locks it.*
- [usb\\_osa\\_status\\_t USB\\_OsaMutexUnlock](#) ([usb\\_osa\\_mutex\\_handle](#) handle)  
*Unlocks a mutex.*

### USB OSA Message Queue

- [usb\\_osa\\_status\\_t USB\\_OsaMsgqCreate](#) ([usb\\_osa\\_msgq\\_handle](#) \*handle, uint32\_t count, uint32\_t size)

- Creates a message queue.*
- `usb_osa_status_t USB_OsaMsgqDestroy (usb_osa_msgq_handle handle)`
- Destroys a message queue.*
- `usb_osa_status_t USB_OsaMsgqSend (usb_osa_msgq_handle handle, void *msg)`
- Sends a message.*
- `usb_osa_status_t USB_OsaMsgqRecv (usb_osa_msgq_handle handle, void *msg, uint32_t timeout)`
- Receives a message.*
- `usb_osa_status_t USB_OsaMsgqCheck (usb_osa_msgq_handle handle, void *msg)`
- Checks a message queue and receives a message if the queue is not empty.*

## 5.2 Enumeration Type Documentation

### 5.2.1 enum usb\_osa\_status\_t

Enumerator

*kStatus\_USB\_OSA\_Success* Success.  
*kStatus\_USB\_OSA\_Error* Failed.  
*kStatus\_USB\_OSA\_TimeOut* Timeout occurs while waiting.

### 5.2.2 enum usb\_osa\_event\_mode\_t

Enumerator

*kUSB\_OsaEventManualClear* The flags of the event is cleared manually.  
*kUSB\_OsaEventAutoClear* The flags of the event is cleared automatically.

## 5.3 Function Documentation

### 5.3.1 void\* USB\_OsaMemoryAllocate ( uint32\_t length )

The function is used to reserve the requested amount of memory in bytes and initializes it to 0.

Parameters

<i>length</i>	Amount of bytes to reserve.
---------------	-----------------------------

Returns

Pointer to the reserved memory. NULL if memory can't be allocated.

### 5.3.2 void USB\_OsaMemoryFree ( void \* p )

The function is used to free the memory block previously reserved.

## Function Documentation

### Parameters

<i>p</i>	Pointer to the start of the memory block previously reserved.
----------	---------------------------------------------------------------

### 5.3.3 `usb_osa_status_t USB_OsaEventCreate ( usb_osa_event_handle * handle, uint32_t flag )`

This function creates an event object and sets its clear mode. If the clear mode is `kUSB_OsaEvent-AutoClear`, when a task gets the event flags, these flags are cleared automatically. If the clear mode is `kUSB_OsaEventManualClear`, the flags must be cleared manually.

### Parameters

<i>handle</i>	It is an out parameter, which is used to return the pointer of the event object.
<i>flag</i>	The event is auto-clear or manual-clear. See the enumeration <a href="#">usb_osa_event_mode_t</a> .

### Returns

A USB OSA error code or `kStatus_OSA_Success`.

### Example:

```
usb_osa_event_handle eventHandle;  
usb_osa_status_t      usbOsaStatus;  
usbOsaStatus = USB_OsaEventCreate(&eventHandle,  
    kUSB_OsaEventManualClear);
```

### 5.3.4 `usb_osa_status_t USB_OsaEventDestroy ( usb_osa_event_handle handle )`

### Parameters

<i>handle</i>	Pointer to the event object.
---------------	------------------------------

### Returns

A USB OSA error code or `kStatus_OSA_Success`.

### Example:

```
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventDestroy(eventHandle);
```

### 5.3.5 `usb_osa_status_t` **USB\_OsaEventSet** ( `usb_osa_event_handle` *handle*, `uint32_t` *bitMask* )

Sets specified flags for an event object.

## Function Documentation

### Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to be set.

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventSet(eventHandle, 0x01U);
```

### 5.3.6 usb\_osa\_status\_t USB\_OsaEventWait ( usb\_osa\_event\_handle *handle*, uint32\_t *bitMask*, uint32\_t *flag*, uint32\_t *timeout*, uint32\_t \* *bitSet* )

This function waits for a combination of flags to be set in an event object. An applications can wait for any/all bits to be set. This function can get the flags that wake up the waiting task.

### Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to wait.
<i>flag</i>	Wait all flags or any flag to be set. 0U - wait any flag, others, wait all flags.
<i>timeout</i>	The maximum number of milliseconds to wait for the event. If the wait condition is not met, passing 0U waits indefinitely when the environment is an RTOS and returns the kStatus_OSA_Timeout immediately. Pass any value for the bare metal.
<i>bitSet</i>	Flags that wake up the waiting task are obtained by this parameter.

### Returns

An USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventWait(eventHandle, 0x01U, 0U, 0U, &bitSet);
```

### 5.3.7 `usb_osa_status_t` `USB_OsaEventCheck` ( `usb_osa_event_handle` *handle*, `uint32_t` *bitMask*, `uint32_t` \* *bitSet* )

This function checks for a combination of flags to be set in an event object.

## Function Documentation

### Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to check.
<i>bitSet</i>	Flags have been set.

### Returns

An USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventCheck(eventHandle, 0x01U, &bitSet);
```

### 5.3.8 usb\_osa\_status\_t USB\_OsaEventClear ( usb\_osa\_event\_handle *handle*, uint32\_t *bitMask* )

This function clears flags of an event object.

### Parameters

<i>handle</i>	Pointer to the event object
<i>bitMask</i>	Event flags to be cleared.

### Returns

An USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventClear(eventHandle, 0x01U);
```

### 5.3.9 usb\_osa\_status\_t USB\_OsaSemCreate ( usb\_osa\_sem\_handle \* *handle*, uint32\_t *count* )

This function creates a semaphore and sets the default count.



## Parameters

<i>handle</i>	It is an out parameter, which is used to return pointer of the semaphore object.
<i>count</i>	Initializes a value of the semaphore.

## Returns

An USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaSemCreate(&semHandle, 1U);
```

### 5.3.10 usb\_osa\_status\_t USB\_OsaSemDestroy ( usb\_osa\_sem\_handle *handle* )

This function destroys a semaphore object.

## Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

## Returns

An USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemDestroy(semHandle);
```

### 5.3.11 usb\_osa\_status\_t USB\_OsaSemPost ( usb\_osa\_sem\_handle *handle* )

This function wakes up a task waiting on the semaphore. If a task is not pending, increases the semaphore's value.

## Function Documentation

### Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemPost(semHandle);
```

### 5.3.12 usb\_osa\_status\_t USB\_OsaSemWait ( usb\_osa\_sem\_handle *handle*, uint32\_t *timeout* )

This function checks the semaphore's value. If it is positive, it decreases the semaphore's value and return kStatus\_OSA\_Success.

### Parameters

<i>handle</i>	Pointer to the semaphore.
<i>timeout</i>	The maximum number of milliseconds to wait for the semaphore. If the wait condition is not met, pass 0U will wait indefinitely when environment is RTOS. And return kStatus_OSA_Timeout immediately for bare metal no matter what value has been passed.

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemWait(semHandle, 0U);
```

### 5.3.13 usb\_osa\_status\_t USB\_OsaMutexCreate ( usb\_osa\_mutex\_handle \* *handle* )

This function creates a mutex and sets it to an unlocked status.

## Parameters

<i>handle</i>	It is out parameter, which is used to return the pointer of the mutex object.
---------------	-------------------------------------------------------------------------------

## Returns

A USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaMutexCreate(&mutexHandle);
```

### 5.3.14 usb\_osa\_status\_t USB\_OsaMutexDestroy ( usb\_osa\_mutex\_handle *handle* )

This function destroys a mutex and sets it to an unlocked status.

## Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

## Returns

A USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaMutexDestroy(mutexHandle);
```

### 5.3.15 usb\_osa\_status\_t USB\_OsaMutexLock ( usb\_osa\_mutex\_handle *handle* )

This function checks the mutex status. If it is unlocked, it locks it and returns the kStatus\_OSA\_Success. Otherwise, it waits forever to lock in RTOS and returns the kStatus\_OSA\_Success immediately for bare metal.

## Function Documentation

### Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexLock(mutexHandle);
```

### 5.3.16 usb\_osa\_status\_t USB\_OsaMutexUnlock ( usb\_osa\_mutex\_handle *handle* )

This function unlocks a mutex.

### Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexUnlock(mutexHandle);
```

### 5.3.17 usb\_osa\_status\_t USB\_OsaMsgqCreate ( usb\_osa\_msgq\_handle \* *handle*, uint32\_t *count*, uint32\_t *size* )

This function creates a message queue.

## Parameters

<i>handle</i>	It is an out parameter, which is used to return a pointer of the message queue object.
<i>count</i>	The count of elements in the queue.
<i>size</i>	Size of every elements in words.

## Returns

A USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
usbOsaStatus = USB_OsaMsgqCreate(msgqHandle, 8U, 4U);
```

### 5.3.18 usb\_osa\_status\_t USB\_OsaMsgqDestroy ( usb\_osa\_msgq\_handle *handle* )

This function destroys a message queue.

## Parameters

<i>handle</i>	Pointer to a message queue.
---------------	-----------------------------

## Returns

A USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqDestroy(msgqHandle);
```

### 5.3.19 usb\_osa\_status\_t USB\_OsaMsgqSend ( usb\_osa\_msgq\_handle *handle*, void \* *msg* )

This function sends a message to the tail of the message queue.

## Function Documentation

### Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to a message to be put into the queue.

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqSend(msgqHandle, &message);
```

### 5.3.20 usb\_osa\_status\_t USB\_OsaMsgqRecv ( usb\_osa\_msgq\_handle *handle*, void \* *msg*, uint32\_t *timeout* )

This function receives a message from the head of the message queue.

### Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.
<i>timeout</i>	The maximum number of milliseconds to wait for a message. If the wait condition is not met, passing 0U waits indefinitely when an environment is RTOS and returns the kStatus_OSA_Timeout immediately for bare metal.

### Returns

A USB OSA error code or kStatus\_OSA\_Success.

### Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqRecv(msgqHandle, &message, 0U);
```

### 5.3.21 usb\_osa\_status\_t USB\_OsaMsgqCheck ( usb\_osa\_msgq\_handle *handle*, void \* *msg* )

This function checks a message queue and receives a message if the queue is not empty.

## Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.

## Returns

A USB OSA error code or kStatus\_OSA\_Success.

## Example:

```
usb_osa_msgq_handle    msgqHandle;  
message_struct_t       message;  
usb_osa_status_t       usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMsgqCheck(msgqHandle, &message);
```





***How to Reach Us:***

**Home Page:**

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

**Web Support:**

[freescale.com/support](http://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, the Freescale logo, Kinetis, Processor Expert are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. Tower is a trademark of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM, ARM Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2016 Freescale Semiconductor, Inc.

Document Number:  
KSDK20USBHAPIRM  
Rev. 0  
Jan 2016

