USB Stack Host Reference Manual

NXP Semiconductors

Document Number: KSDK20USBHAPIRM
Rev. 0
Sept 2016

Contents

Chapter Overview

1.1 USB Host Initialization flow **3**

1.2 USB Host peripheral attach/detach flow **5**

Chapter Definitions and structures

2.1 Overview **7**

2.2 Data Structure Documentation **8**
2.2.1 struct usb_version_t 8

2.3 Typedef Documentation **9**
2.3.1 usb_device_handle 9

2.4 Enumeration Type Documentation **9**
2.4.1 usb_status_t 9
2.4.2 usb_controller_index_t 9

Chapter USB Host driver

3.1 Overview **11**

3.2 Data Structure Documentation **15**
3.2.1 struct usb_host_ep_t 15
3.2.2 struct usb_host_interface_t 15
3.2.3 struct usb_host_configuration_t 16
3.2.4 struct usb_host_pipe_t 16
3.2.5 struct usb_host_transfer_t 17
3.2.6 struct usb_host_pipe_init_t 17
3.2.7 struct usb_host_cancel_param_t 18
3.2.8 struct usb_host_device_instance_t 18
3.2.9 struct usb_host_process_feature_param_t 19
3.2.10 struct usb_host_process_descriptor_param_t 19
3.2.11 struct usb_host_get_interface_param_t 20
3.2.12 struct usb_host_get_status_param_t 20
3.2.13 struct usb_host_set_interface_param_t 20
3.2.14 struct usb_host_synch_frame_param_t 20

Contents

Section Number	Title	Page Number
3.2.15	struct usb_host_instance_t	21
3.3	**Typedef Documentation**	**21**
3.3.1	host_callback_t	21
3.3.2	transfer_callback_t	22
3.3.3	host_inner_transfer_callback_t	22
3.4	**Enumeration Type Documentation**	**23**
3.4.1	usb_host_event_t	23
3.4.2	usb_host_dev_info_t	23
3.4.3	usb_host_device_enumeration_status_t	23
3.4.4	usb_host_interface_state_t	24
3.4.5	usb_host_device_state_t	24
3.4.6	usb_host_request_type_t	24
3.5	**Function Documentation**	**24**
3.5.1	USB_HostInit	24
3.5.2	USB_HostDeinit	25
3.5.3	USB_HostHelperGetPeripheralInformation	25
3.5.4	USB_HostHelperParseAlternateSetting	26
3.5.5	USB_HostRemoveDevice	27
3.5.6	USB_HostKhciTaskFunction	27
3.5.7	USB_HostEhciTaskFunction	27
3.5.8	USB_HostOhciTaskFunction	28
3.5.9	USB_HostIp3516HsTaskFunction	28
3.5.10	USB_HostKhciIsrFunction	28
3.5.11	USB_HostEhciIsrFunction	28
3.5.12	USB_HostOhciIsrFunction	29
3.5.13	USB_HostIp3516HsIsrFunction	29
3.5.14	USB_HostOpenPipe	29
3.5.15	USB_HostClosePipe	30
3.5.16	USB_HostSend	30
3.5.17	USB_HostSendSetup	31
3.5.18	USB_HostRecv	31
3.5.19	USB_HostCancelTransfer	32
3.5.20	USB_HostMallocTransfer	32
3.5.21	USB_HostFreeTransfer	33
3.5.22	USB_HostRequestControl	33
3.5.23	USB_HostOpenDeviceInterface	34
3.5.24	USB_HostCloseDeviceInterface	34
3.5.25	USB_HostGetVersion	35
3.5.26	USB_HostAttachDevice	35
3.5.27	USB_HostDetachDevice	36
3.5.28	USB_HostDetachDeviceInternal	37
3.5.29	USB_HostGetDeviceAttachState	37

Contents

Section Number	Title	Page Number
3.5.30	USB_HostValidateDevice	37
3.6	**USB Host Controller driver**	**39**
3.6.1	Overview	39
3.6.2	Data Structure Documentation	40
3.6.3	Enumeration Type Documentation	40
3.6.4	USB Host Controller KHCI driver	42
Chapter	**USB Class driver**	
4.1	**Overview**	**49**
4.2	**USB CDC Class driver**	**50**
4.2.1	Overview	50
4.2.2	USB Host CDC Initialization	50
4.2.3	USB Host CDC De-initialization	52
4.2.4	USB Host CDC Send data	52
4.2.5	USB Host CDC Receive data	52
4.2.6	Data Structure Documentation	55
4.2.7	Function Documentation	61
4.3	**USB HID Class driver**	**72**
4.3.1	Overview	72
4.3.2	USB Host HID Initialization	72
4.3.3	USB Host HID Deinitialization	73
4.3.4	USB Host HID Send data	73
4.3.5	USB Host HID Receive data	73
4.3.6	Data Structure Documentation	75
4.3.7	Function Documentation	77
4.4	**USB MSC Class driver**	**87**
4.4.1	Overview	87
4.4.2	USB Host MSC Initialization	87
4.4.3	USB Host MSC Deinitialization	88
4.4.4	USB Host MSC UFI Command	88
4.4.5	Data Structure Documentation	91
4.4.6	Function Documentation	95
4.5	**USB AUDIO Class driver**	**116**
4.5.1	Overview	116
4.5.2	USB Host audio Initialization	116
4.5.3	USB Host audio De-initialization	118
4.5.4	USB Host audio Send data	118
4.5.5	USB Host audio Receive data	118
4.5.6	Data Structure Documentation	121

Contents

Section Number	Title	Page Number
4.5.7	Function Documentation	126
4.6	**USB PHDC Class driver**	**133**
4.6.1	Overview	133
4.6.2	USB Host PHDC Initialization	133
4.6.3	USB Host PHDC Deinitialization	134
4.6.4	USB Host PHDC Send data	134
4.6.5	USB Host PHDC Receive data	134
4.6.6	Data Structure Documentation	136
4.6.7	Function Documentation	138
4.7	**USB PRINTER Class driver**	**145**
4.7.1	Overview	145
4.7.2	Data Structure Documentation	146
4.7.3	Function Documentation	147
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	158
5.3.3	USB_OsaEventCreate	159
5.3.4	USB_OsaEventDestroy	159
5.3.5	USB_OsaEventSet	160
5.3.6	USB_OsaEventWait	161
5.3.7	USB_OsaEventCheck	162
5.3.8	USB_OsaEventClear	163
5.3.9	USB_OsaSemCreate	163
5.3.10	USB_OsaSemDestroy	164
5.3.11	USB_OsaSemPost	164
5.3.12	USB_OsaSemWait	165
5.3.13	USB_OsaMutexCreate	165
5.3.14	USB_OsaMutexDestroy	166
5.3.15	USB_OsaMutexLock	166
5.3.16	USB_OsaMutexUnlock	167
5.3.17	USB_OsaMsgqCreate	167
5.3.18	USB_OsaMsgqDestroy	168
5.3.19	USB_OsaMsgqSend	168
5.3.20	USB_OsaMsgqRecv	169

Contents

Section Number	Title	Page Number
5.3.21	USB_OsaMsgqCheck	169

Chapter Data Structure Documentation

6.0.22	usb_host_ip3516hs_atl_struct_t Struct Reference	171
6.0.23	usb_host_ip3516hs_ptl_struct_t Struct Reference	171
6.0.24	usb_host_ip3516hs_register_struct_t Struct Reference	172
6.0.25	usb_host_ip3516hs_sptl_struct_t Struct Reference	172
6.0.26	usb_host_ip3516hs_state_struct_t Struct Reference	173
6.0.27	usb_host_ohci_endpoint_descritpor_struct_t Struct Reference	173
6.0.28	usb_host_ohci_general_transfer_descritpor_struct_t Struct Reference	174
6.0.29	usb_host_ohci_hcca_struct_t Struct Reference	175
6.0.30	usb_host_ohci_hcor_struct_t Struct Reference	175
6.0.31	usb_host_ohci_isochronous_transfer_descritpor_struct_t Struct Reference	175
6.0.32	usb_host_ohci_state_struct_t Struct Reference	176

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:

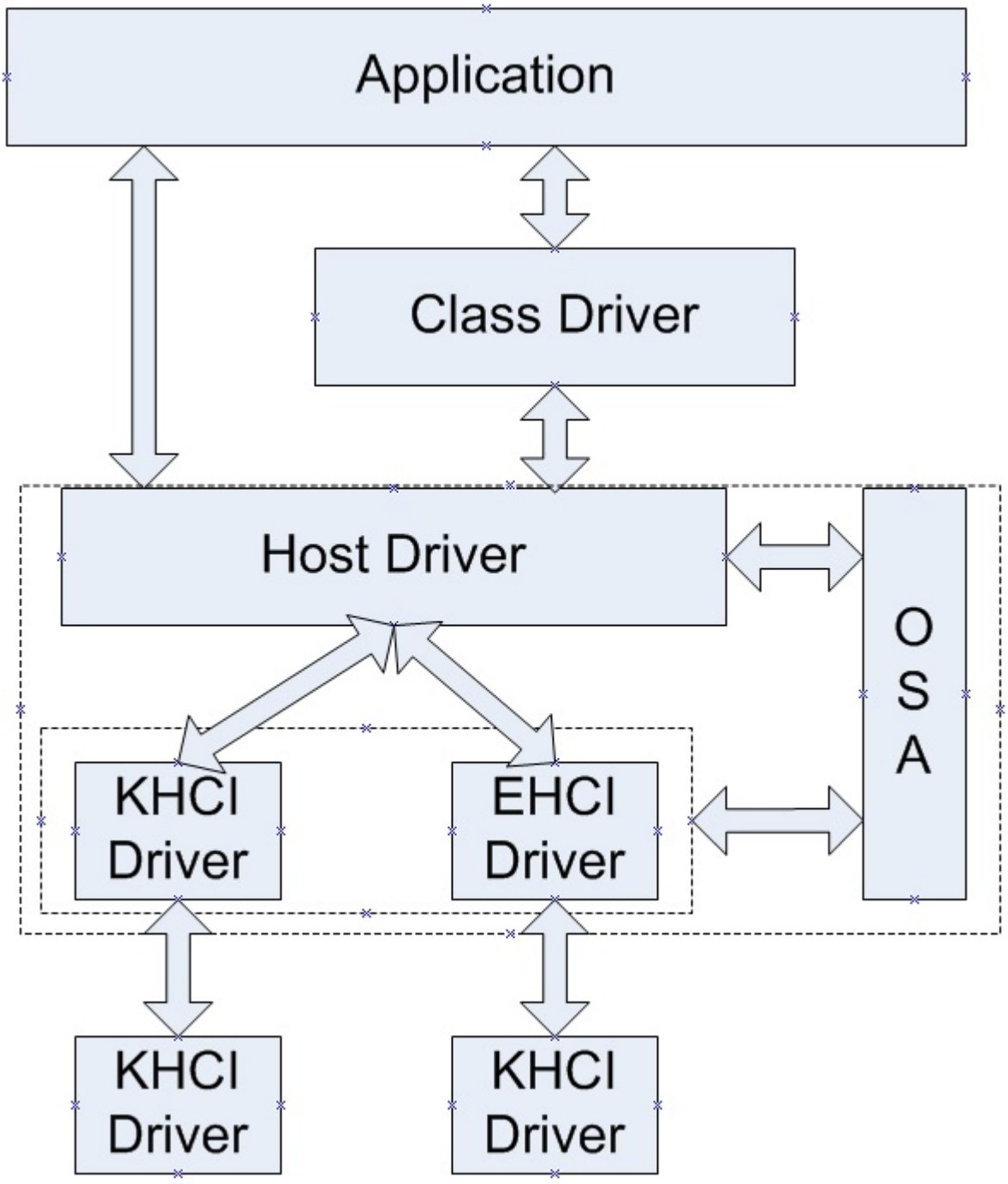

Figure 1: Host stack architecture

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:

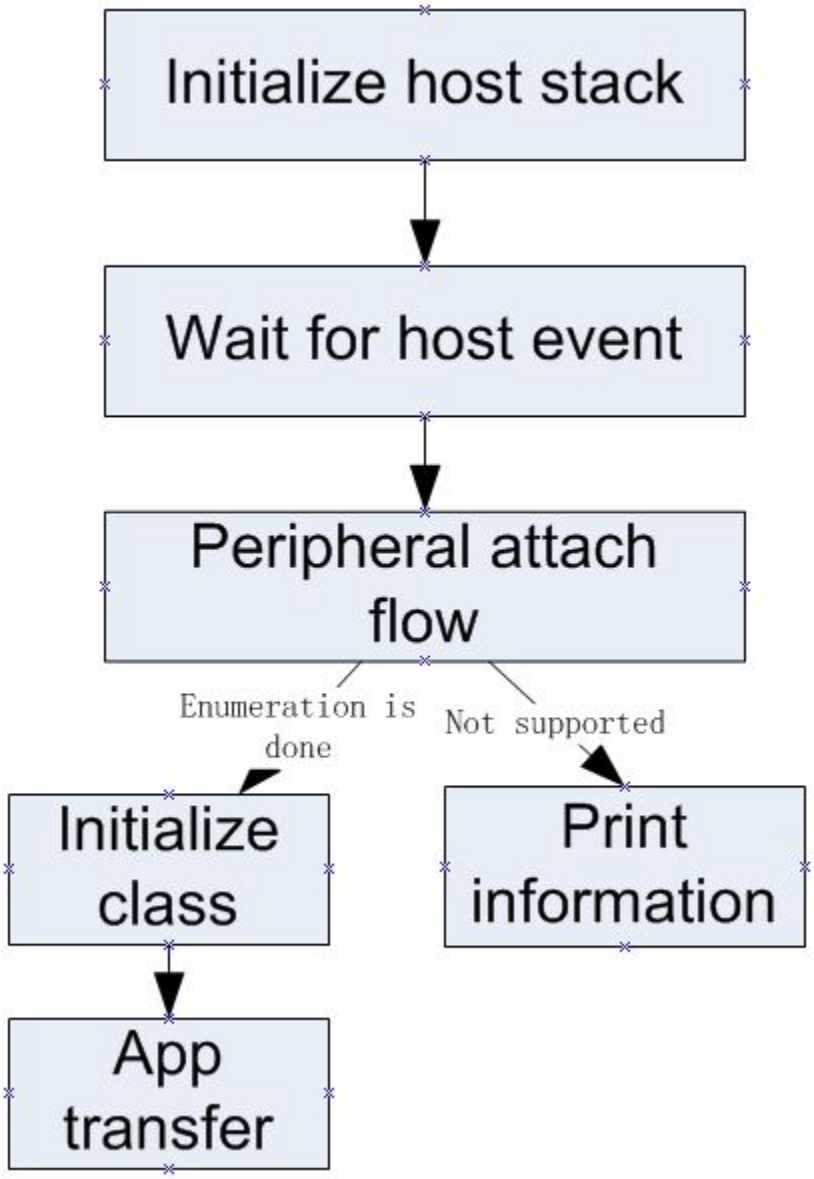

Figure 1.1.1: Host stack work flow

The host stack initialization work flow is as follows:

USB Host Initialization flow

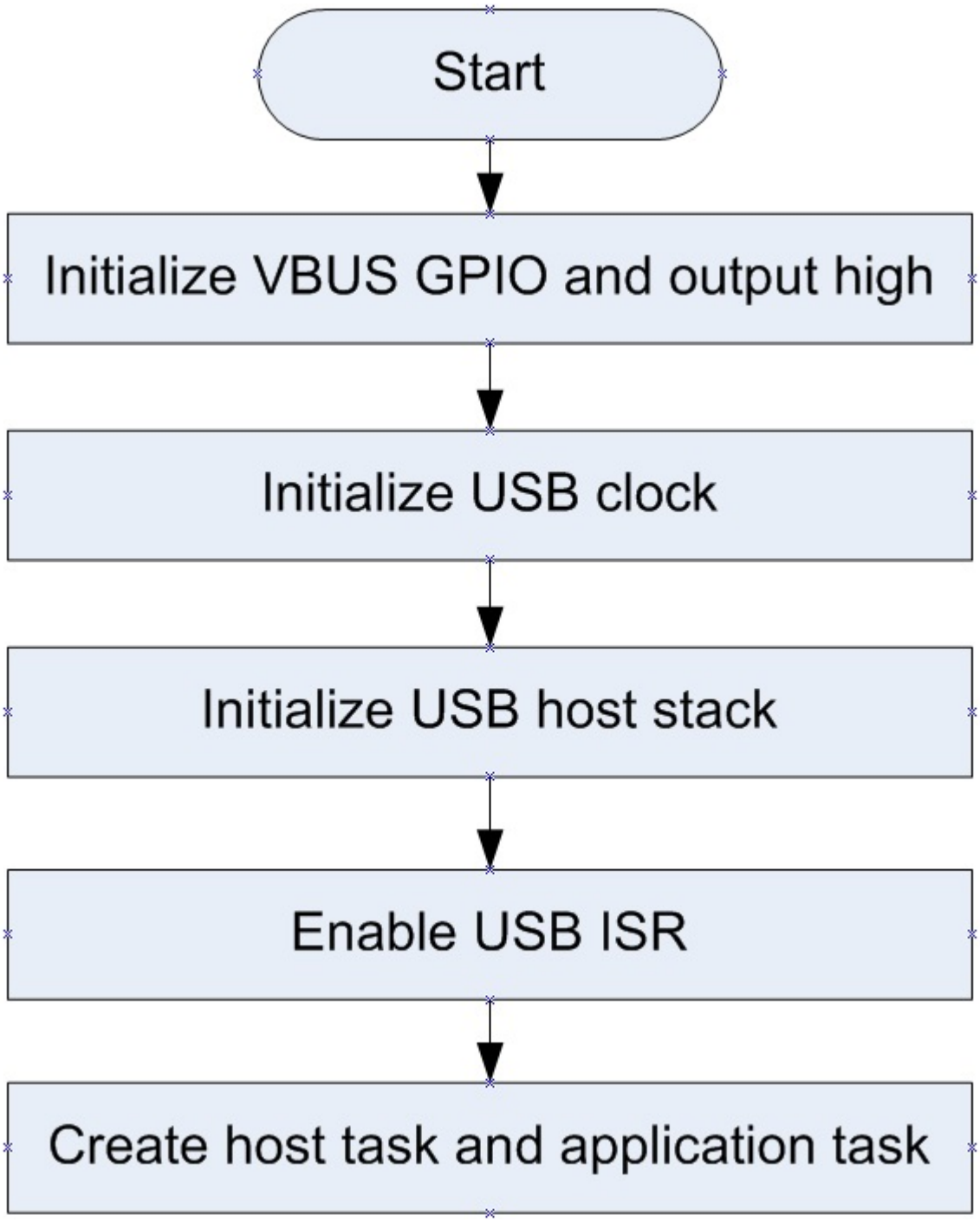

Figure 1.1.2: Host stack initialization flow

- 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:

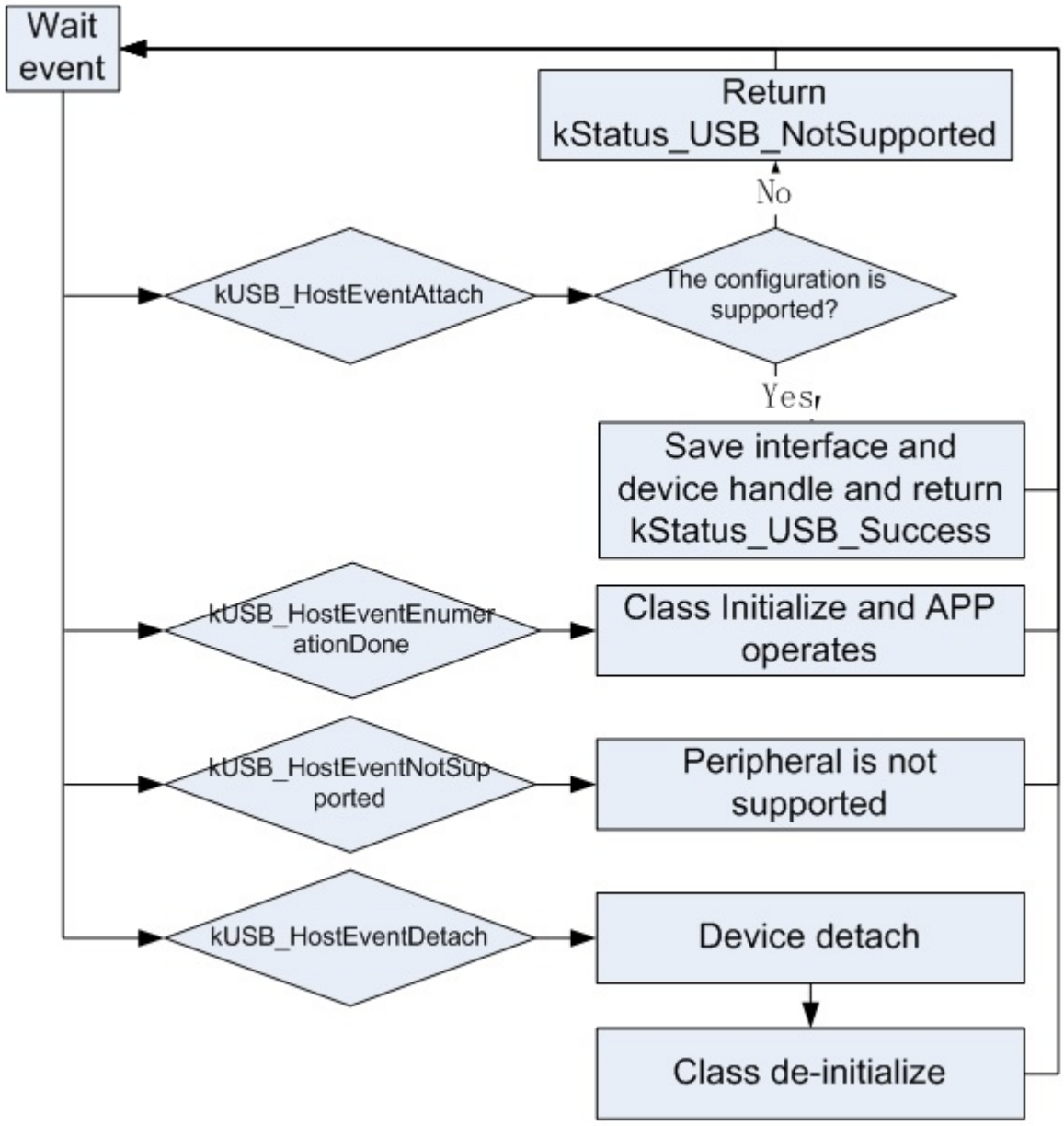

Figure 1.2.1: Host stack attach flow

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 attaching the unsupported 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.

This lists the common definitions and structures for 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 bugfix 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_MSDStatusFail }
  USB error code.
- enum usb_controller_index_t {
  kUSB_ControllerKhci0 = 0U,
  kUSB_ControllerKhci1 = 1U,
  kUSB_ControllerEhci0 = 2U,
  kUSB_ControllerEhci1 = 3U,
  kUSB_ControllerLpcIp3511Fs0 = 4U,
  kUSB_ControllerLpcIp3511Fs1,
  kUSB_ControllerLpcIp3511Hs0 = 6U,
  kUSB_ControllerLpcIp3511Hs1,
  kUSB_ControllerOhci0 = 8U,
  kUSB_ControllerOhci1 = 9U,
  kUSB_ControllerIp3516Hs0 = 10U,
  kUSB_ControllerIp3516Hs1 = 11U }
  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 EHCI IPs, this is reserved to be used in the future.
kUSB_ControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

Enumeration Type Documentation

kUSB_ControllerLpcIp3511Fs1 LPC USB IP3511 FS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kUSB_ControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

kUSB_ControllerLpcIp3511Hs1 LPC USB IP3511 HS controller 1, there are no platforms which have two IP3511 IPs, this is reserved to be used in the future.

kUSB_ControllerOhci0 OHCI 0U.

kUSB_ControllerOhci1 OHCI 1U, Currently, there are no platforms which have two OHCI IPs, this is reserved to be used in the future.

kUSB_ControllerIp3516Hs0 IP3516HS 0U.

kUSB_ControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two I-P3516HS 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_REQUEST_STANDARD_CLEAR_FEATURE and USB_REQUEST_STANDARD_SET_FEATURE. More...
- struct usb_host_process_descriptor_param_t
  For USB_REQUEST_STANDARD_GET_DESCRIPTOR and USB_REQUEST_STANDARD_SET_DESCRIPTOR. More...
- struct usb_host_get_interface_param_t
  For USB_REQUEST_STANDARD_GET_INTERFACE. More...
- struct usb_host_get_status_param_t
  For USB_REQUEST_STANDARD_GET_STATUS. More...
- struct usb_host_set_interface_param_t
  For USB_REQUEST_STANDARD_SET_INTERFACE. More...
- struct usb_host_synch_frame_param_t
  For USB_REQUEST_STANDARD_SYNCH_FRAME. More...
- 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,
  kUSB_HostGetDeviceConfigIndex,
  kUSB_HostGetConfigurationDes,
  kUSB_HostGetConfigurationLength }
  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 }
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.

- 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_HostOhciTaskFunction (void *hostHandle)
  OHCI task function.
- void USB_HostIp3516HsTaskFunction (void *hostHandle)
  IP3516HS task function.
- void USB_HostKhciIsrFunction (void *hostHandle)
  Device KHCI ISR function.
- void USB_HostEhciIsrFunction (void *hostHandle)
  Device EHCI ISR function.
- void USB_HostOhciIsrFunction (void *hostHandle)
  Device OHCI ISR function.
- void USB_HostIp3516HsIsrFunction (void *hostHandle)
  Device IP3516HS 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_INTERFACE]
  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
  Length transferred so far.
- 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.
- uint32_t unitHead
  xTD head for this transfer
- int32_t transferResult
  KHCI transfer result.
- uint32_t unitTail
  xTD tail for this transfer
- uint32_t frame
  KHCI transfer frame number.

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.

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, please reference to 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
  Interface number.
- uint8_t * alternateInterfaceBuffer
  Save the transfer result.

3.2.12 struct usb_host_get_status_param_t

Data Fields

- uint16_t statusSelector
  Interface number, the end pointer number or OTG status selector.
- uint8_t requestType
  See the usb_host_request_type_t.
- 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

deviceHandle	The device handle, which indicates the attached device.

configuration-Handle	The configuration handle contains the attached device's configuration information.
event_code	The callback event code; See the enumeration host_event_t.

Returns

A USB error code or kStatus_USB_Success.

Return values

kStatus_USB_Success	Application handles the attached device successfully.
kStatus_USB_Not-Supported	Application don't support the attached device.
kStatus_USB_Error	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

param	The parameter pointer, which is passed when calling the send/receive APIs.
data	The data buffer pointer.
data_len	The result data length.
status	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

param	The parameter pointer, which is passed when calling the send/receive APIs.
transfer	The transfer information; See the structure usb_host_transfer_t.
status	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.
kUSB_HostGetDeviceConfigIndex Device's running zero-based config index.
kUSB_HostGetConfigurationDes Device's configuration descriptor pointer.
kUSB_HostGetConfigurationLength Device's configuration descriptor pointer.

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.

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.

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	*controllerId*	The controller ID of the USB IP. See the enumeration usb_controller_-index_t.
out	*hostHandle*	Returns the host handle.
in	*callbackFn*	Host callback function notifies device attach/detach.

Return values

kStatus_USB_Success	The host is initialized successfully.
kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer.
kStatus_USB_Controller-NotFound	Cannot find the controller according to the controller ID.
kStatus_USB_AllocFail	Allocation memory fail.
kStatus_USB_Error	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	*hostHandle*	The host handle.

Return values

kStatus_USB_Success	The host is initialized successfully.
kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer.
kStatus_USB_Error	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	*deviceHandle*	Removing device handle.
in	*infoCode*	See the enumeration host_dev_info_t.
out	*infoValue*	Return the information value.

Return values

kStatus_USB_Success	Close successfully.
kStatus_USB_Invalid-Parameter	The deviceHandle or info_value is a NULL pointer.
kStatus_USB_Error	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	*interface-Handle*	The whole interface handle.
in	*alternate-Setting*	Alternate setting value.
out	*interface*	Return interface information.

Return values

kStatus_USB_Success	Close successfully.
kStatus_USB_Invalid-Handle	The interfaceHandle is a NULL pointer.
kStatus_USB_Invalid-Parameter	The alternateSetting is 0.

kStatus_USB_Error	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	*hostHandle*	The host handle.
in	*deviceHandle*	Removing device handle.

Return values

kStatus_USB_Success	Remove successfully.
kStatus_USB_Invalid-Handle	The hostHandle or deviceHandle is a NULL pointer.
kStatus_USB_Invalid-Parameter	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 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	*hostHandle*	The host handle.

3.5.7 void USB_HostEhciTaskFunction (void * *hostHandle*)

The function is used to handle the EHCI controller message. In the 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	*hostHandle*	The host handle.

3.5.8 void USB_HostOhciTaskFunction (void * *hostHandle*)

The function is used to handle the OHCI controller message. In the 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	*hostHandle*	The host handle.

3.5.9 void USB_HostIp3516HsTaskFunction (void * *hostHandle*)

The function is used to handle the IP3516HS controller message. In the 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	*hostHandle*	The host handle.

3.5.10 void USB_HostKhciIsrFunction (void * *hostHandle*)

The function is the KHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

3.5.11 void USB_HostEhciIsrFunction (void * *hostHandle*)

The function is the EHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

3.5.12 void USB_HostOhciIsrFunction (void * *hostHandle*)

The function is the OHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

3.5.13 void USB_HostIp3516HsIsrFunction (void * *hostHandle*)

The function is the IP3516HS interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

3.5.14 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	*hostHandle*	The host handle.
out	*pipeHandle*	The pipe handle pointer used to return the pipe handle.
in	*pipeInit*	Used to initialize the pipe.

Return values

kStatus_USB_Success	The host is initialized successfully.

kStatus_USB_Invalid-Handle	The hostHandle or pipe_handle_ptr is a NULL pointer.
kStatus_USB_Error	There is no idle pipe. Or, there is no idle QH for EHCI. Or, bandwidth allocate fail for EHCI.

3.5.15 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	*hostHandle*	The host handle.
in	*pipeHandle*	The closing pipe handle.

Return values

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

3.5.16 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	*hostHandle*	The host handle.
in	*pipeHandle*	The sending pipe handle.
in	*transfer*	The transfer information.

Return values

kStatus_USB_Success	Send successfully.

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

3.5.17 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.

Parameters

in	*hostHandle*	The host handle.
in	*pipeHandle*	The sending pipe handle.
in	*transfer*	The transfer information.

Return values

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

3.5.18 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	*hostHandle*	The host handle.

in	*pipeHandle*	The receiving pipe handle.
in	*transfer*	The transfer information.

Return values

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

3.5.19 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	*hostHandle*	The host handle.
in	*pipeHandle*	The receiving pipe handle.
in	*transfer*	The transfer information.

Return values

kStatus_USB_Success	Cancel successfully.
kStatus_USB_Invalid-Handle	The hostHandle or pipeHandle is a NULL pointer.

3.5.20 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	*hostHandle*	The host handle.
out	*transfer*	Return the transfer.

Return values

kStatus_USB_Success	Allocate successfully.
kStatus_USB_Invalid-Handle	The hostHandle or transfer is a NULL pointer.
kStatus_USB_Error	There is no idle transfer.

3.5.21 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	*hostHandle*	The host handle.
in	*transfer*	Release the transfer.

Return values

kStatus_USB_Success	Free successfully.
kStatus_USB_Invalid-Handle	The hostHandle or transfer is a NULL pointer.

3.5.22 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	*deviceHandle*	The device handle for control transfer.
in	*usbRequest*	A USB standard request code. Se the usb_spec.h.
in	*transfer*	The used transfer.
in	*param*	The parameter structure is different for different request, see usb_host_framework.h.

Return values

kStatus_USB_Success	Send successfully.
kStatus_USB_InvalidHandle	The deviceHandle is a NULL pointer.
kStatus_USB_LackSwapBuffer	There is no swap buffer for KHCI.
kStatus_USB_Error	There is no idle QTD/ITD/SITD for EHCI, Or, the request is not standard request.

3.5.23 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	*deviceHandle*	Removing device handle.
in	*interfaceHandle*	Opening interface handle.

Return values

kStatus_USB_Success	Open successfully.
kStatus_USB_InvalidHandle	The deviceHandle or interfaceHandle is a NULL pointer.

3.5.24 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	*deviceHandle*	Removing device handle.
in	*interface-Handle*	Opening interface handle.

Return values

kStatus_USB_Success	Close successfully.
kStatus_USB_Invalid-Handle	The deviceHandle is a NULL pointer.

3.5.25 void USB_HostGetVersion (uint32_t * *version*)

The function is used to get the host stack version.

Parameters

out	*version*	The version structure pointer to keep the host stack version.

3.5.26 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

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

Returns

kStatus_USB_Success or error codes.

3.5.27 usb_status_t USB_HostDetachDevice (usb_host_handle *hostHandle,* uint8_t *hubNumber,* uint8_t *portNumber*)

Parameters

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

Returns

kStatus_USB_Success or error codes.

3.5.28 usb_status_t USB_HostDetachDeviceInternal (usb_host_handle *hostHandle*, usb_device_handle *deviceHandle*)

Parameters

hostHandle	Host instance handle.
deviceHandle	Device handle.

Returns

kStatus_USB_Success or error codes.

3.5.29 uint8_t USB_HostGetDeviceAttachState (usb_device_handle *deviceHandle*)

Parameters

deviceHandle	Device handle.

Returns

0x01 - attached; 0x00 - detached.

3.5.30 usb_status_t USB_HostValidateDevice (usb_host_handle *hostHandle*, usb_device_handle *deviceHandle*)

Parameters

hostHandle	Host instance pointer.
deviceHandle	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,
  kUSB_HostPortAttachDisable,
  kUSB_HostPortAttachEnable }
  USB host controller control code.
- enum usb_host_bus_control_t {
  kUSB_HostBusReset = 1U,
  kUSB_HostBusRestart,
  kUSB_HostBusEnableAttach,
  kUSB_HostBusDisableAttach,
  kUSB_HostBusSuspend,
  kUSB_HostBusResume }
  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_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.
kUSB_HostPortAttachDisable Disable the port attach event.
kUSB_HostPortAttachEnable Enable the port attach event.

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.
kUSB_HostBusSuspend Suspend BUS.
kUSB_HostBusResume Resume BUS.

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.

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 bas 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 asyncListAvtive
  KHCI async list is active.
- uint8_t periodicListAvtive
  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 to 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.

Parameters

controllerId	The controller ID of the USB IP. See the enumeration usb_controller_index_t.
hostHandle	The host level handle.
controller-Handle	Returns the controller instance handle.

Return values

kStatus_USB_Success	The host is initialized successfully.
kStatus_USB_AllocFail	Allocates memory failed.
kStatus_USB_Error	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

controller-Handle	The controller handle.

Return values

kStatus_USB_Success	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 * *pipeInitPointer*)

This function opens a pipe according to the pipe_init_ptr parameter.

Parameters

controller-Handle	The controller handle.

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

Return values

kStatus_USB_Success	The host is initialized successfully.
kStatus_USB_Error	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

controller-Handle	The controller handle.
pipeHandle	The closing pipe handle.

Return values

kStatus_USB_Success	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

controller-Handle	The controller handle.
pipeHandle	The sending pipe handle.
transfer	The transfer information.

Return values

kStatus_USB_Success	Send successful.
kStatus_USB_LackSwap-Buffer	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

controller-Handle	The controller handle.
pipeHandle	The receiving pipe handle.
transfer	The transfer information.

Return values

kStatus_USB_Success	Receive successful.
kStatus_USB_LackSwap-Buffer	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

controller-Handle	The controller handle.
ioctlEvent	See the enumeration host_bus_control_t.
ioctlParam	The control parameter.

Return values

kStatus_USB_Success	Cancel successful.
kStatus_USB_Invalid-Handle	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
- USB PRINTER 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:

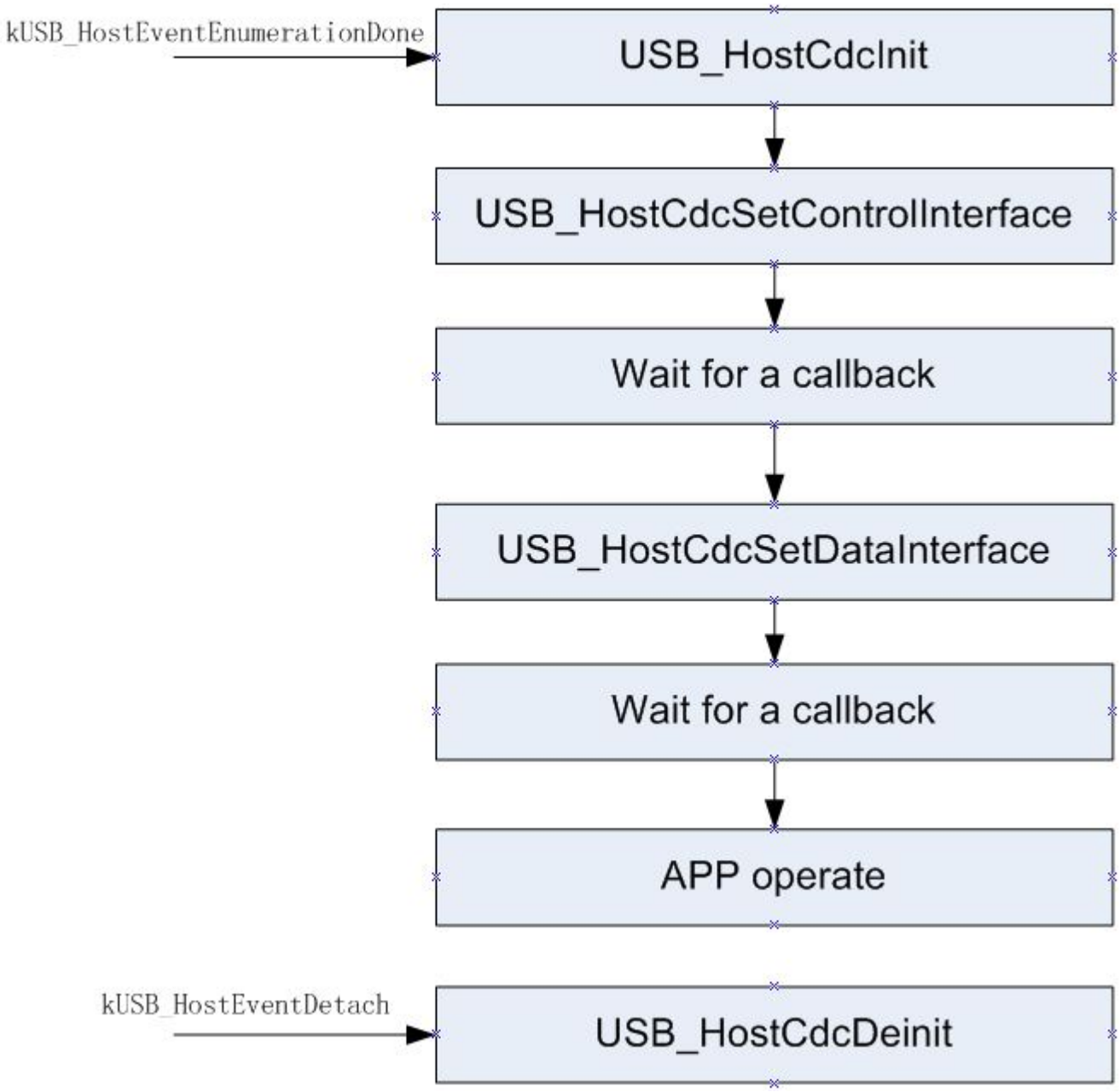

Figure 4.2.1: Host Cdc Initialization

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.

- 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 analyze a class-specific descriptor. More...

Macros

- #define USB_HOST_CDC_SEND_ENCAPSULATED_COMMAND 0x00U
  CDC class-specific request (SEND_ENCAPSULATED_COMMAND)
- #define USB_HOST_CDC_GET_ENCAPSULATED_RESPONSE 0x01U
  CDC class-specific request (GET_ENCAPSULATED_RESPONSE)
- #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.
- #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_HostCdcSendEncapsulatedCommand (usb_host_class_handle classHandle, uint8_t *buffer, uint16_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)
  cdc send encapsulated command.
- usb_status_t USB_HostCdcGetEncapsulatedResponse (usb_host_class_handle classHandle, uint8_t *buffer, uint16_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)
  cdc get encapsulated response.
- 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.

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:.

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

deviceHandle	The device handle.
classHandle	Returns class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	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	*classHandle*	The class handle.
in	*interfaceHandle*	The interface handle.
in	*alternateSetting*	The alternate setting value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_InvalidHandle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSendSetup.
kStatus_USB_Busy	Callback return status, there is no idle pipe.
kStatus_USB_TransferStall	Callback return status, the transfer is stalled by the device.

kStatus_USB_Error	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*)

Parameters

in	*classHandle*	The class handle.
in	*interfaceHandle*	The interface handle.
in	*alternateSetting*	The alternate setting value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_InvalidHandle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSendSetup.
kStatus_USB_Busy	Callback return status, there is no idle pipe.
kStatus_USB_TransferStall	Callback return status, the transfer is stalled by the device.
kStatus_USB_Error	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

deviceHandle	The device handle.
classHandle	The class handle.

Return values

kStatus_USB_Success	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	*classHandle*	The class handle.
in	*pipeType*	Its value is USB_ENDPOINT_CONTROL, USB_ENDPOINT_ISOCHRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTERRUPT. See the usb_spec.h
in	*direction*	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

classHandle	The class handle.
buffer	The buffer pointer.

bufferLength	The buffer length.
callbackFn	This callback is called after this function completes.
callbackParam	The first parameter in the callback function.

Return values

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

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

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

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

classHandle	The class handle.
dtr	The DRS value.
rts	The RTS value.
callbackFn	This callback is called after this function completes.
callbackParam	The first parameter in the callback function.

Return values

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

4.2.7.11 usb_status_t USB_HostCdcSendEncapsulatedCommand (usb_host_class_handle *classHandle*, uint8_t * *buffer*, uint16_t *bufferLength*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements cdc SEND_ENCAPSULATED_COMMAND request.refer to cdc 1.2 spec.

Parameters

classHandle	the class handle.

buffer	the buffer pointer.
bufferLength	the buffer length.
callbackFn	this callback is called after this function completes.
callbackParam	the first parameter in the callback function.

Return values

kStatus_USB_Success	request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	send transfer fail, please reference to USB_HostSendSetup.

4.2.7.12 usb_status_t USB_HostCdcGetEncapsulatedResponse (usb_host_class_handle *classHandle*, uint8_t * *buffer*, uint16_t *bufferLength*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements cdc GET_ENCAPSULATED_RESPONSE request.refer to cdc 1.2 spec.

Parameters

classHandle	the class handle.
buffer	the buffer pointer.
bufferLength	the buffer length.
callbackFn	this callback is called after this function completes.
callbackParam	the first parameter in the callback function.

Return values

kStatus_USB_Success	request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.

kStatus_USB_Error	send transfer fail, please reference to USB_HostSendSetup.

4.2.7.13 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

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

Return values

kStatus_USB_Error	Analyse descriptor error.

4.2.7.14 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

classHandle	The class handle.
request_type	Set up the packet request type.

request	Set up the packet request value.
wvalue_l	Set up the packet wvalue low byte.
wvalue_h	Set up the packet wvalue high byte.
wlength	Set up the packet wlength value.
data	Data buffer pointer
callbackFn	This callback is called after this function completes.
callbackParam	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:

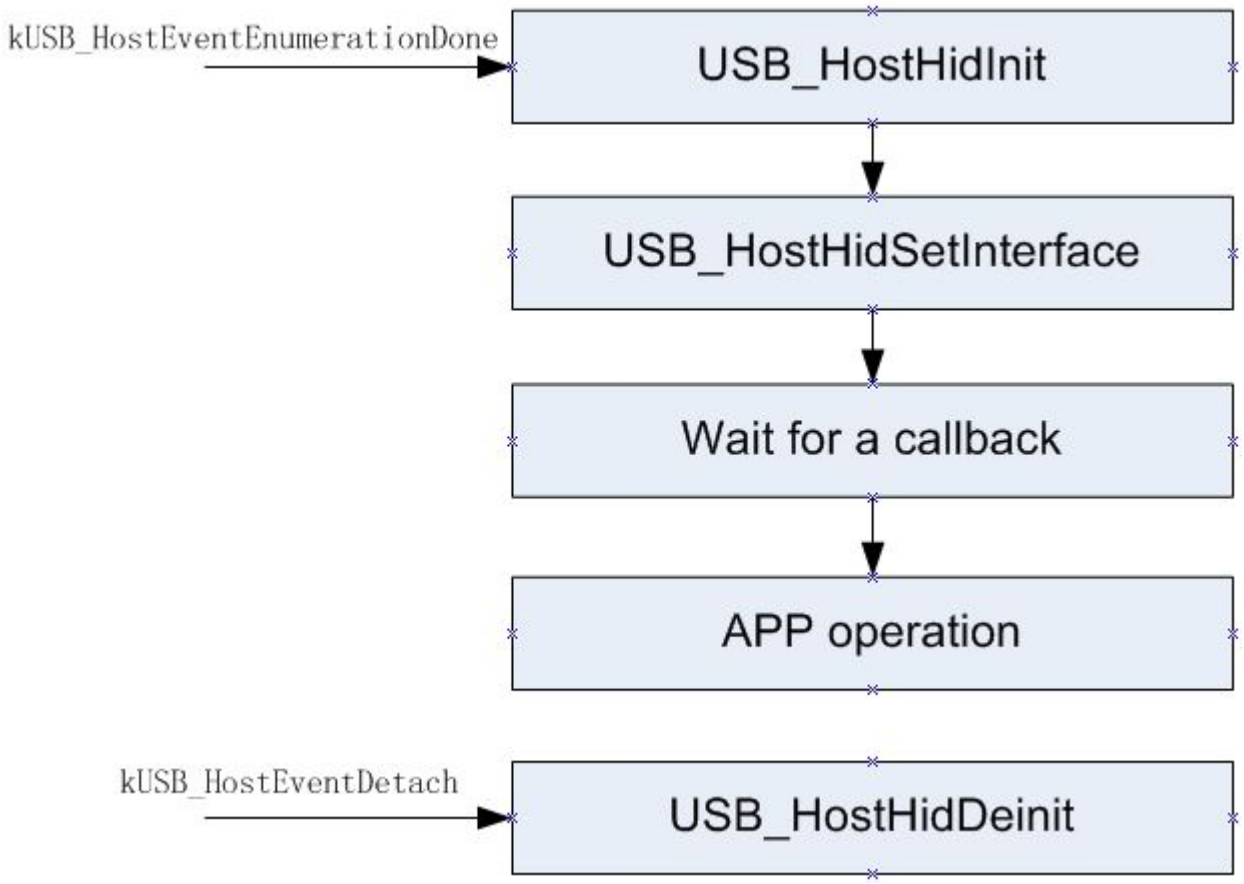

Figure 4.3.1: Host HID Initialization

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

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

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.

Parameters

in	*deviceHandle*	The device handle.
out	*classHandle*	Return class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	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	*classHandle*	The class handle.
in	*interface-Handle*	The interface handle.
in	*alternate-Setting*	The alternate setting value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

kStatus_USB_Transfer-Stall	Callback return status, the transfer is stalled by the device.
kStatus_USB_Error	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	*deviceHandle*	The device handle.
in	*classHandle*	The class handle.

Return values

kStatus_USB_Success	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	*classHandle*	The class handle.
in	*pipeType*	Its value is USB_ENDPOINT_CONTROL, USB_ENDPOINT_ISOCHRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTERRUPT. See the usb_spec.h
in	*direction*	Pipe direction.

Return values

0	The classHandle is NULL.
Maximum	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.

Parameters

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

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
out	*buffer*	The buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
in	*buffer*	The buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	Send request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.

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

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
in	*reportId*	Report ID.
in	*idleRate*	Idle rate value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
out	*protocol*	Return protocol value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
in	*protocol*	Protocol value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

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

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
in	*reportId*	Report ID.
in	*reportType*	Report type.
in	*buffer*	The buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	Request successful.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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:

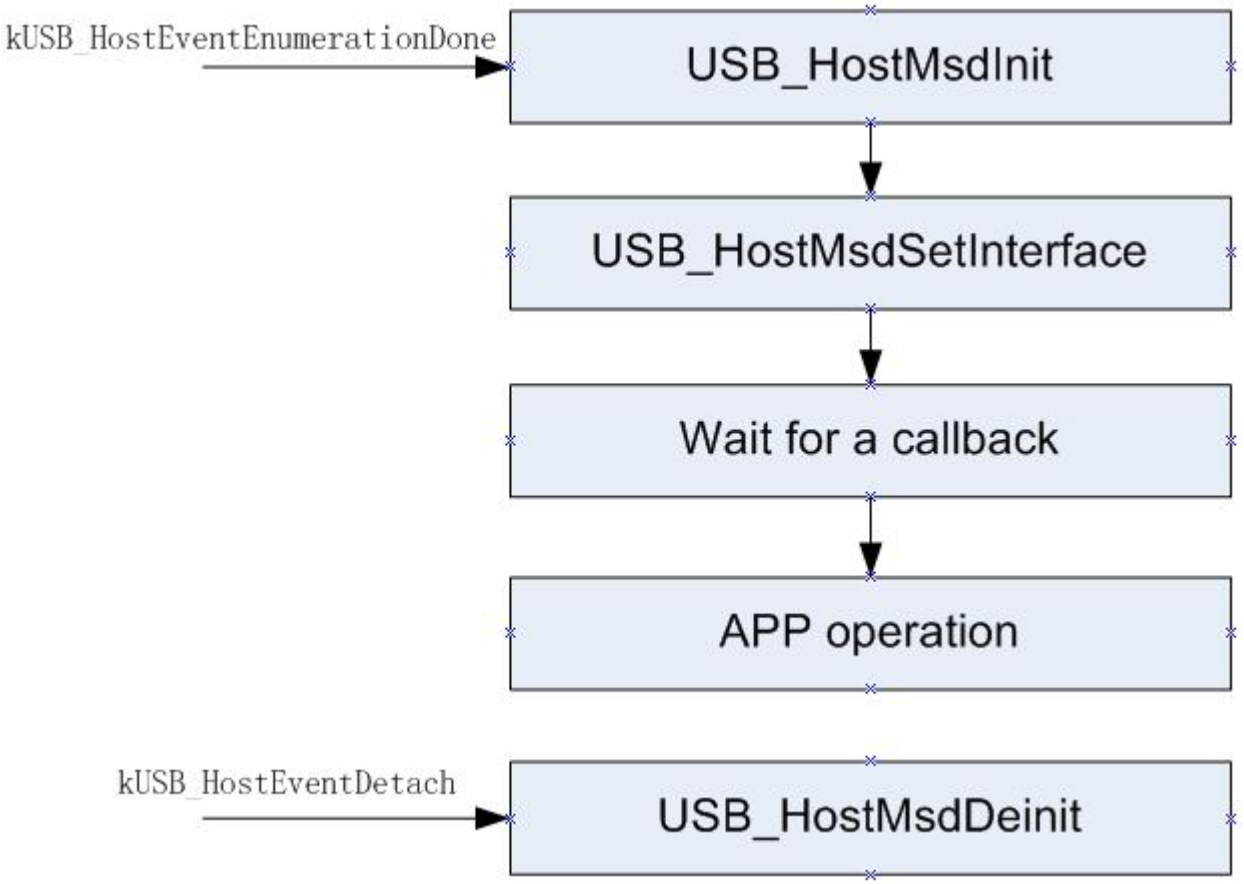

Figure 4.4.1: Host MSD Initialization

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

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_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)
  Mass storage read (12).
- 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)
  Mass storage write (10).
- 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)
  Mass storage write (12).
- usb_status_t USB_HostMsdReadCapacity (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_HostMsdTestUnitReady (usb_host_class_handle classHandle, uint8_t logicalUnit, transfer_callback_t callbackFn, void *callbackParam)
  Mass storage test unit ready.
- 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)
  mass storage request sense.
- 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)
  Mass storage mode select.
- 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)
  Mass storage mode sense.
- 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)
  Mass storage inquiry.
- 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)
  Mass storage read format capacities.
- 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)
  Mass storage format unit.
- usb_status_t USB_HostMsdPreventAllowRemoval (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_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)
  Mass storage write and verify.
- 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)

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

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.

- 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_HostMsdInit (usb_device_handle *deviceHandle,* usb_host_class_handle * *classHandle*)

This function allocates the resources for the MSD instance.

Parameters

in	*deviceHandle*	The device handle.
out	*classHandle*	Return class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	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	*classHandle*	The class handle.
in	*interface-Handle*	The interface handle.
in	*alternate-Setting*	The alternate setting value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

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

4.4.6.3 usb_status_t USB_HostMsdDeinit (usb_device_handle *deviceHandle,* usb_host_class_handle *classHandle*)

This function frees the resource for the MSD instance.

Parameters

in	*deviceHandle*	The device handle.
in	*classHandle*	The class handle.

Return values

kStatus_USB_Success	The device is de-initialized successfully.

4.4.6.4 usb_status_t USB_HostMsdMassStorageReset (usb_host_class_handle *classHandle,* transfer_callback_t *callbackFn,* void * *callbackParam*)

This function implements the mass storage reset request.

Parameters

in	*classHandle*	The class handle.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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	*classHandle*	The class handle.
out	*logicalUnit-Number*	Return logical unit number value.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSendSetup.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*blockAddress*	The start block address.
out	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*blockNumber*	Read block number.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.

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

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*blockAddress*	The start block address.
in	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.

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

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*blockAddress*	The start block address.
in	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*blockNumber*	Write block number.
in	*callbackFn*	This callback is called after this command completes.

in	*callbackParam*	The first parameter in the callback function.

Return values

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

4.4.6.10 usb_status_t USB_HostMsdReadCapacity (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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
out	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

Parameters

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

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.

in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*pageControl*	The page control field specifies the type of mode parameters to return.
in	*pageCode*	Buffer pointer.
out	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
out	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
out	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*trackNumber*	This specifies which track is to be formatted.
in	*interLeave*	This specifies the interleave that shall be used for formatting.
in	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.

in	*prevent*	Prevent or allow • 0: enable (allow) the removal of the medium • 1: disable (prevent) removal of the medium
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*blockAddress*	The start block address.
in	*buffer*	Buffer pointer.
in	*bufferLength*	The buffer length.

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

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*loadEject*	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	*start*	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.

in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*blockAddress*	The start block address.
in	*verification-Length*	The data length that need to be verified.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

kStatus_USB_Success	The device is initialized successfully.

kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

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

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	The previous command is executing or there is no idle transfer.
kStatus_USB_Error	Send transfer fail. See the USB_HostSend/USB_HostRecv.
kStatus_USB_Success	Callback return status, the command succeed.
kStatus_USB_MSD-StatusFail	Callback return status, the CSW status indicate this command fail.
kStatus_USB_Error	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	*classHandle*	The class MSD handle.
in	*logicalUnit*	Logical unit number.
in	*selfTest*	0 = perform special diagnostic test; 1 = perform default self-test.
in	*callbackFn*	This callback is called after this command completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

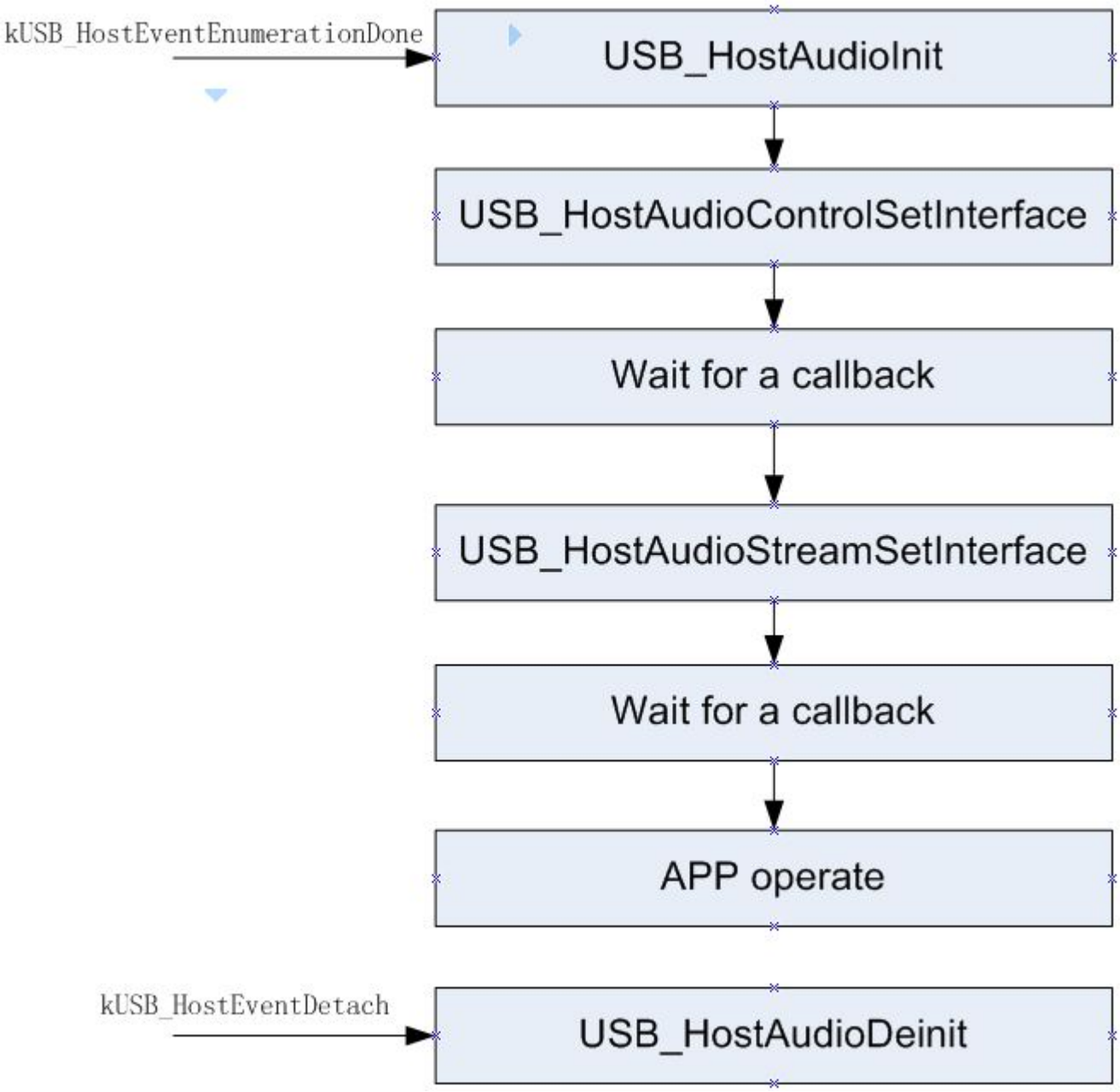

Figure 4.5.1: Host Audio Initialization

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

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.

- #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_spepific_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

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 iterminal
  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 bcontrolsize
  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 blockdlayunits
  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.

- 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_spepific_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

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

deviceHandle	The device handle.
classHandlePtr	Return class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	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

deviceHandle	The device handle.
classHandle	The class handle.

Return values

kStatus_USB_Success	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

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

Return values

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

classHandle	The class handle.
interface-Handle	The interface handle.

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

Return values

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

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

Return values

0	The classHandle is NULL.
max	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

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

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

classHandle	The class handle.
asIntfDesc	The pointer of class-specific AS interface descriptor.
formatTypeDesc	The pointer of format type descriptor.
isoEndpDesc	The pointer of specific ISO endp descriptor.

Return values

kStatus_USB_Success	Get the audio stream current altsetting descriptor request successfully.
kStatus_USB_InvalidHandle	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

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

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

Return values

kStatus_USB_Success	Feature unit request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Endpoint request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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:

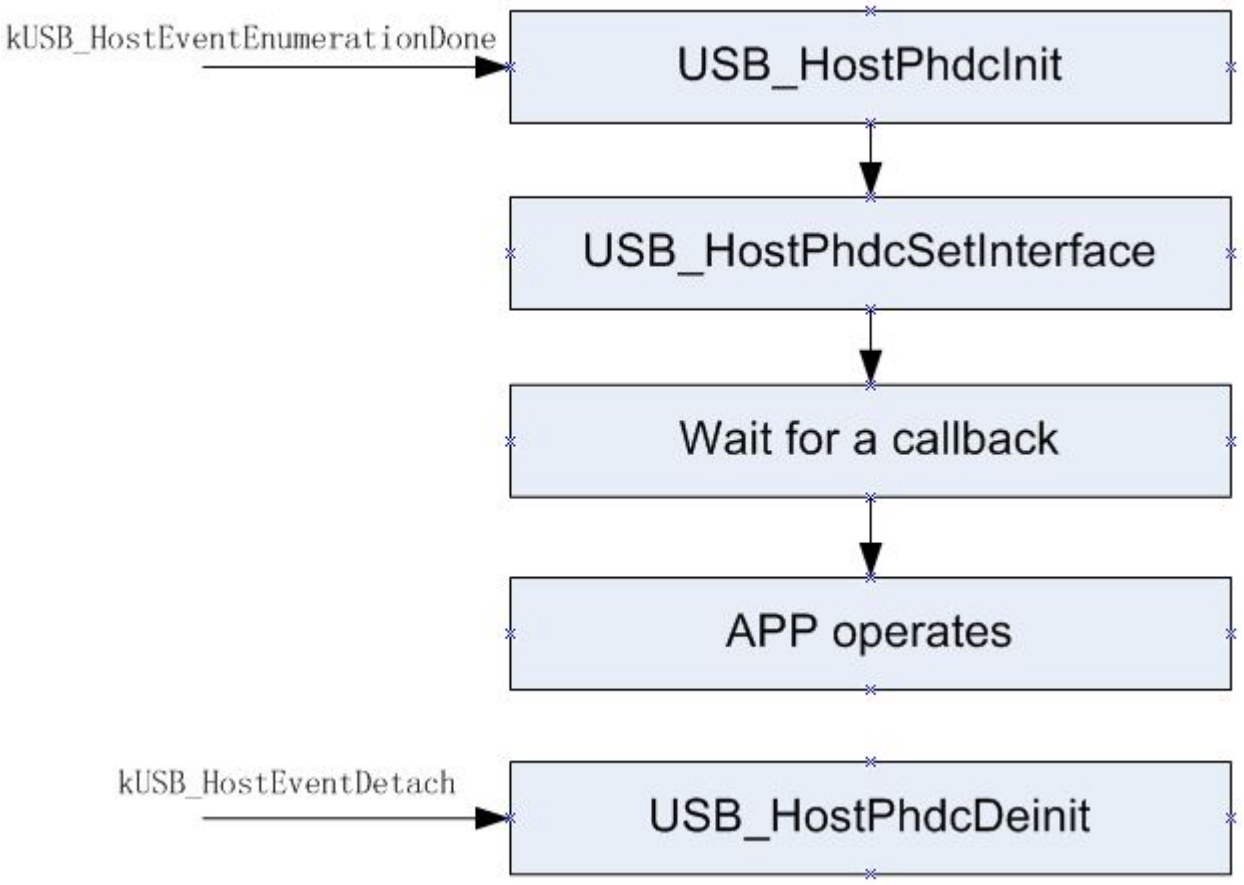

Figure 4.6.1: Host PHDC Initialization

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

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_desciptor_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_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_desciptor_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

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

deviceHandle	The device handle.
classHandle	Return class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	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

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

Return values

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

kStatus_USB_Transfer-Stall	Callback return status, the transfer is stalled by the device.
kStatus_USB_Error	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

deviceHandle	The device handle.
classHandle	The class handle.

Return values

kStatus_USB_Success	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

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

Return values

kStatus_USB_Success	Receive request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Send request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Send request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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

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

Return values

kStatus_USB_Success	Send request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.
kStatus_USB_Busy	There is no idle transfer.
kStatus_USB_Error	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.

Parameters

classHandle	The class handle.
pipeType	Pipe type.
direction	Pipe direction.

Return values

endpointReturn	All input parameters are valid.
NULL	One or more input parameters are invalid.

4.7 USB PRINTER Class driver

4.7.1 Overview

Data Structures

- struct usb_host_printer_instance_t
  Printer instance structure and printer usb_host_class_handle pointer to this structure. More...

Macros

- #define USB_HOST_PRINTER_CLASS_CODE (7U)
  Printer class code.
- #define USB_HOST_PRINTER_SUBCLASS_CODE (1U)
  Printer sub-class code.
- #define USB_HOST_PRINTER_PROTOCOL_UNIDIRECTION (1U)
  Printer class protocol code (Unidirectional interface)
- #define USB_HOST_PRINTER_PROTOCOL_BIDIRECTION (2U)
  Printer class protocol code (Bi-directional interface)
- #define USB_HOST_PRINTER_PROTOCOL_IEEE1284 (3U)
  Printer class protocol code (IEEE® 1284.4 compatible bi-directional interface)
- #define USB_HOST_PRINTER_GET_DEVICE_ID (0)
  Printer class-specific request (GET_DEVICE_ID)
- #define USB_HOST_PRINTER_GET_PORT_STATUS (1)
  Printer class-specific request (GET_PORT_STATUS)
- #define USB_HOST_PRINTER_SOFT_RESET (2)
  Printer class-specific request (SOFT_RESET)
- #define USB_HOST_PRINTER_PORT_STATUS_PAPER_EMPTRY_MASK (0x20U)
  Paper empty bit mask for GET_PORT_STATUS.
- #define USB_HOST_PRINTER_PORT_STATUS_SELECT_MASK (0x10U)
  Select bit mask for GET_PORT_STATUS.
- #define USB_HOST_PRINTER_PORT_STATUS_NOT_ERROR_MASK (0x08U)
  Error bit mask for GET_PORT_STATUS.

USB host printer class APIs

- usb_status_t USB_HostPrinterInit (usb_device_handle deviceHandle, usb_host_class_handle *classHandle)
  Initializes the printer instance.
- usb_status_t USB_HostPrinterSetInterface (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_HostPrinterDeinit (usb_device_handle deviceHandle, usb_host_class_handle classHandle)
  De-initializes the the printer instance.

- uint16_t USB_HostPrinterGetPacketsize (usb_host_class_handle classHandle, uint8_t pipeType, uint8_t direction)
  Gets the pipe maximum packet size.
- usb_status_t USB_HostPrinterRecv (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)
  Receives data.
- usb_status_t USB_HostPrinterSend (usb_host_class_handle classHandle, uint8_t *buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void *callbackParam)
  Sends data.
- usb_status_t USB_HostPrinterGetDeviceId (usb_host_class_handle classHandle, uint8_t interfaceIndex, uint8_t alternateSetting, uint8_t *buffer, uint32_t length, transfer_callback_t callbackFn, void *callbackParam)
  Printer get device ID.
- usb_status_t USB_HostPrinterGetPortStatus (usb_host_class_handle classHandle, uint8_t *portStatus, transfer_callback_t callbackFn, void *callbackParam)
  Printer get port status.
- usb_status_t USB_HostPrinterSoftReset (usb_host_class_handle classHandle, transfer_callback_t callbackFn, void *callbackParam)
  Printer soft reset.

4.7.2 Data Structure Documentation

4.7.2.1 struct usb_host_printer_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
  Printer bulk in pipe.
- usb_host_pipe_handle outPipe
  Printer bulk out pipe.
- transfer_callback_t inCallbackFn
  Printer bulk in transfer callback function pointer.
- void * inCallbackParam
  Printer bulk in transfer callback parameter.
- transfer_callback_t outCallbackFn
  Printer bulk out transfer callback function pointer.
- void * outCallbackParam
  Printer bulk out transfer callback parameter.
- transfer_callback_t controlCallbackFn
  Printer control transfer callback function pointer.

- void * controlCallbackParam
  Printer control transfer callback parameter.
- usb_host_transfer_t * controlTransfer
  Ongoing control transfer.
- uint16_t inPacketSize
  Printer bulk in maximum packet size.
- uint16_t outPacketSize
  Printer bulk out maximum packet size.

4.7.3 Function Documentation

4.7.3.1 usb_status_t USB_HostPrinterInit (usb_device_handle *deviceHandle*, usb_host_class_handle * *classHandle*)

This function allocate the resource for the printer instance.

Parameters

in	*deviceHandle*	The device handle.
out	*classHandle*	Return class handle.

Return values

kStatus_USB_Success	The device is initialized successfully.
kStatus_USB_AllocFail	Allocate memory fail.

4.7.3.2 usb_status_t USB_HostPrinterSetInterface (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 printer instance.

Parameters

in	*classHandle*	The class handle.
in	*interface-Handle*	The interface handle.

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

Return values

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

4.7.3.3 usb_status_t USB_HostPrinterDeinit (usb_device_handle *deviceHandle,* usb_host_class_handle *classHandle*)

This function frees the resources for the printer instance.

Parameters

in	*deviceHandle*	The device handle.
in	*classHandle*	The class handle.

Return values

kStatus_USB_Success	The device is de-initialized successfully.

4.7.3.4 uint16_t USB_HostPrinterGetPacketsize (usb_host_class_handle *classHandle,* uint8_t *pipeType,* uint8_t *direction*)

Parameters

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

Return values

0	The classHandle is NULL.
Maximum	Packet size.

4.7.3.5 usb_status_t USB_HostPrinterRecv (usb_host_class_handle *classHandle*, uint8_t * *buffer*, uint32_t *bufferLength*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements the printer receiving data.

Parameters

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

Return values

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

4.7.3.6 usb_status_t USB_HostPrinterSend (usb_host_class_handle *classHandle,* uint8_t * *buffer,* uint32_t *bufferLength,* transfer_callback_t *callbackFn,* void * *callbackParam*)

This function implements the printer sending data.

Parameters

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

Return values

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

4.7.3.7 usb_status_t USB_HostPrinterGetDeviceId (usb_host_class_handle *classHandle*, uint8_t *interfaceIndex*, uint8_t *alternateSetting*, uint8_t * *buffer*, uint32_t *length*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements the printer class-specific request (GET_DEVICE_ID).

Parameters

in	*classHandle*	The class handle.
in	*interfaceIndex*	Interface index.
in	*alternate-Setting*	Get the alternateSetting's device ID.
out	*buffer*	The buffer pointer.
in	*length*	The buffer length.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

4.7.3.8 usb_status_t USB_HostPrinterGetPortStatus (usb_host_class_handle *classHandle*, uint8_t * *portStatus*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements the printer class-specific request (GET_PORT_STATUS).

Parameters

in	*classHandle*	The class handle.
in	*portStatus*	Port status buffer.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

4.7.3.9 usb_status_t USB_HostPrinterSoftReset (usb_host_class_handle *classHandle*, transfer_callback_t *callbackFn*, void * *callbackParam*)

This function implements the printer class-specific request (SOFT_RESET).

Parameters

in	*classHandle*	The class handle.

in	*portStatus*	Port status buffer.
in	*callbackFn*	This callback is called after this function completes.
in	*callbackParam*	The first parameter in the callback function.

Return values

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

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.

The OS adapter (OSA) is used to wrap the differences between RTOSes and make the 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

length	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.

Parameters

p	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

handle	It is an out parameter, which is used to return the pointer of the event object.
flag	The event is auto-clear or manual-clear. See the enumeration usb_osa_event_mode_t.

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

handle	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.

Parameters

handle	Pointer to the event object.
bitMask	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

handle	Pointer to the event object.
bitMask	Event flags to wait.
flag	Wait all flags or any flag to be set. 0U - wait any flag, others, wait all flags.
timeout	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.
bitSet	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.

Parameters

handle	Pointer to the event object.
bitMask	Event flags to check.
bitSet	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

handle	Pointer to the event object
bitMask	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.

Function Documentation

Parameters

handle	It is an out parameter, which is used to return pointer of the semaphore object.
count	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

handle	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.

Parameters

handle	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

handle	Pointer to the semaphore.
timeout	The maximum number of milliseconds to wait for the semaphore. If the wait condition is not met, passing 0U waits 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

handle	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

handle	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.

Parameters

handle	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

handle	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

handle	It is an out parameter, which is used to return a pointer of the message queue object.
count	The count of elements in the queue.
size	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

handle	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.

Parameters

handle	Pointer to a message queue.
msg	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

handle	Pointer to a message queue.
msg	The pointer to save a received message.
timeout	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.

Function Documentation

Parameters

handle	Pointer to a message queue.
msg	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);
```

Chapter 6
Data Structure Documentation

6.0.22 usb_host_ip3516hs_atl_struct_t Struct Reference

IP3516HS Transaction Descriptor.

`#include <usb_host_ip3516hs.h>`

6.0.22.1 Field Documentation

6.0.22.1.1 uint32_t usb_host_ip3516hs_atl_struct_t::R1

6.0.22.1.2 volatile uint32_t usb_host_ip3516hs_atl_struct_t::J

1: enable the next PTD branching.

6.0.22.1.3 volatile uint32_t usb_host_ip3516hs_atl_struct_t::uFrame

6.0.22.1.4 uint32_t usb_host_ip3516hs_atl_struct_t::R2

6.0.22.1.5 volatile uint32_t usb_host_ip3516hs_atl_struct_t::S

6.0.22.1.6 volatile uint32_t usb_host_ip3516hs_atl_struct_t::RL

RL and NakCnt are set to the same value before a transaction.

6.0.22.1.7 volatile uint32_t usb_host_ip3516hs_atl_struct_t::Token

6.0.23 usb_host_ip3516hs_ptl_struct_t Struct Reference

IP3516HS Transaction Descriptor.

`#include <usb_host_ip3516hs.h>`

6.0.23.1 Field Documentation

6.0.23.1.1 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::NextPTDPointer

6.0.23.1.2 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::J

1: enable the next PTD branching.

6.0.23.1.3 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::uFrame

6.0.23.1.4 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::S

6.0.23.1.5 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::RL

RL and NakCnt are set to the same value before a transaction.

6.0.23.1.6 volatile uint32_t usb_host_ip3516hs_ptl_struct_t::Token

6.0.24 usb_host_ip3516hs_register_struct_t Struct Reference

IP3516HS Host Controller Operational Registers.

`#include <usb_host_ip3516hs.h>`

6.0.25 usb_host_ip3516hs_sptl_struct_t Struct Reference

IP3516HS Transaction Descriptor.

`#include <usb_host_ip3516hs.h>`

6.0.25.1 Field Documentation

6.0.25.1.1 uint32_t usb_host_ip3516hs_sptl_struct_t::R1

6.0.25.1.2 volatile uint32_t usb_host_ip3516hs_sptl_struct_t::uFrame

6.0.25.1.3 volatile uint32_t usb_host_ip3516hs_sptl_struct_t::MaxPacketLength

6.0.25.1.4 uint32_t usb_host_ip3516hs_sptl_struct_t::R2

6.0.25.1.5 volatile uint32_t usb_host_ip3516hs_sptl_struct_t::S

6.0.25.1.6 volatile uint32_t usb_host_ip3516hs_sptl_struct_t::RL

6.0.25.1.7 volatile uint32_t usb_host_ip3516hs_sptl_struct_t::Token

6.0.26 usb_host_ip3516hs_state_struct_t Struct Reference

IP3516HS controller driver instance structure.

`#include <usb_host_ip3516hs.h>`

Data Fields

- volatile usb_host_ip3516hs_register_struct_t * usbRegBase
  The base address of the register.
- void * hostHandle
  Related host handle.
- uint32_t bufferArrayBitMap [4]
  Bit map for USB dedicated RAM (Uint is 64bytes)
- usb_osa_event_handle ip3516HsEvent
  IP3516HS event.
- usb_osa_mutex_handle mutex
  Ip3516Hs layer mutex.
- uint8_t controllerId
  Controller id.
- uint8_t portNumber
  Port count.
- uint8_t isrNumber
  ISR Number.

6.0.27 usb_host_ohci_endpoint_descritpor_struct_t Struct Reference

OHCI Endpoint Descriptor.

`#include <usb_host_ohci.h>`

Data Fields

- volatile uint32_t TailP
  TDQueueTailPointer.
- volatile uint32_t HeadP
  TDQueueHeadPointer.
- volatile uint32_t NextED
  NextED.
- struct _usb_host_ohci_pipe_struct * pipe
  Pipe handle for the ED.
- uint32_t FA: 7U
  FunctionAddress.
- uint32_t EN: 4U
  EndpointNumber.
- uint32_t D: 2U
  Direction: 00,11 - Get dir from TD, 01 - OUT, 10 - IN.
- uint32_t S: 1U
  Speed: 0 - full speed, 1 - low speed.
- uint32_t K: 1U
  Skip.
- uint32_t F: 1U
  Format: 0 - Control, Bulk, or Interrupt Endpoint, 1 - ISO Endpoint.
- volatile uint32_t MPS: 11U
  MaximumPacketSize.
- uint32_t reserved1: 5U
  Reserved.

6.0.27.1 Field Documentation

6.0.27.1.1 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::D

6.0.27.1.2 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::S

6.0.27.1.3 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::F

6.0.28 usb_host_ohci_general_transfer_descritpor_struct_t Struct Reference

OHCI General Transfer Descriptor.

`#include <usb_host_ohci.h>`

Data Fields

- uint32_t CBP
  CurrentBufferPointer.
- volatile uint32_t NextTD
  NextTD.
- uint32_t BE

BufferEnd.
- struct _usb_host_ohci_pipe_struct * pipe
  Pipe handle for the GTD.
- uint32_t reserved1: 18U
  Reserved.
- volatile uint32_t R: 1U
  bufferRounding
- volatile uint32_t DP: 2U
  Direction/PID.
- volatile uint32_t DI: 3U
  DelayInterrupt.
- volatile uint32_t T: 2U
  DataToggle.
- volatile uint32_t EC: 2U
  ErrorCount.
- volatile uint32_t CC: 4U
  ConditionCode.

6.0.29 usb_host_ohci_hcca_struct_t Struct Reference

OHCI Host Controller Communications Area.

```
#include <usb_host_ohci.h>
```

Data Fields

- volatile uint32_t HccaInterrruptTable [USB_HOST_OHCI_HCCA_SIZE]
  These 32 Dwords are pointers to interrupt EDs.
- volatile uint16_t HccaFrameNumber
  Contains the current frame number.
- uint16_t HccaPad1
  When the HC updates HccaFrameNumber, it sets this word to 0.
- volatile uint32_t HccaDoneHead
  Hcca done head.
- uint32_t reserved [30]
  Reserved for use by HC.

6.0.30 usb_host_ohci_hcor_struct_t Struct Reference

OHCI Host Controller Operational Registers.

```
#include <usb_host_ohci.h>
```

6.0.31 usb_host_ohci_isochronous_transfer_descritpor_struct_t Struct Reference

OHCI Isochronous Transfer Descriptor.

```
#include <usb_host_ohci.h>
```

Data Fields

- uint32_t BP0
  BufferPage0.
- volatile uint32_t NextTD
  NextTD.
- uint32_t BE
  BufferEnd.
- volatile uint16_t OffsetPSW [8]
  Offest or PacketStatusWord.
- struct _usb_host_ohci_pipe_struct * pipe
  Pipe handle for the ITD.
- volatile uint32_t SF: 16U
  StartingFrame.
- uint32_t reserved1: 5U
  Reserved.
- volatile uint32_t DI: 3U
  DelayInterrupt.
- volatile uint32_t FC: 3U
  FrameCount.
- uint32_t reserved2: 1U
  Reserved.
- volatile uint32_t CC: 4U
  ConditionCode.

6.0.32 usb_host_ohci_state_struct_t Struct Reference

OHCI controller driver instance structure.

```
#include <usb_host_ohci.h>
```

Data Fields

- volatile
  usb_host_ohci_hcor_struct_t * usbRegBase
  The base address of the register.
- void * hostHandle
  Related host handle.
- usb_osa_event_handle ohciEvent
  OHCI event.
- usb_osa_mutex_handle mutex
  OHCI layer mutex.
- uint8_t controllerId
  Controller id.
- uint8_t portNumber
  Port count.

- uint8_t isrNumber
  ISR Number.
- volatile uint8_t gtdCount
  Gtd count.
- volatile uint8_t itdCount
  Itd count.

How to Reach Us:

Home Page:
nxp.com

Web Support:
nxp.com/support

Document Number: KSDK20USBHAPIRM
Rev. 0
Sept 2016