MCUXpresso SDK USB Stack Host Reference Manual

NXP Semiconductors

Contents

Contents

Contents

Contents

Contents

Chapter 1
Overview

1.1 Introduction

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 1.1.1: Host stack architecture

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

1.2   USB Host Initialization flow

The host stack works as follows:
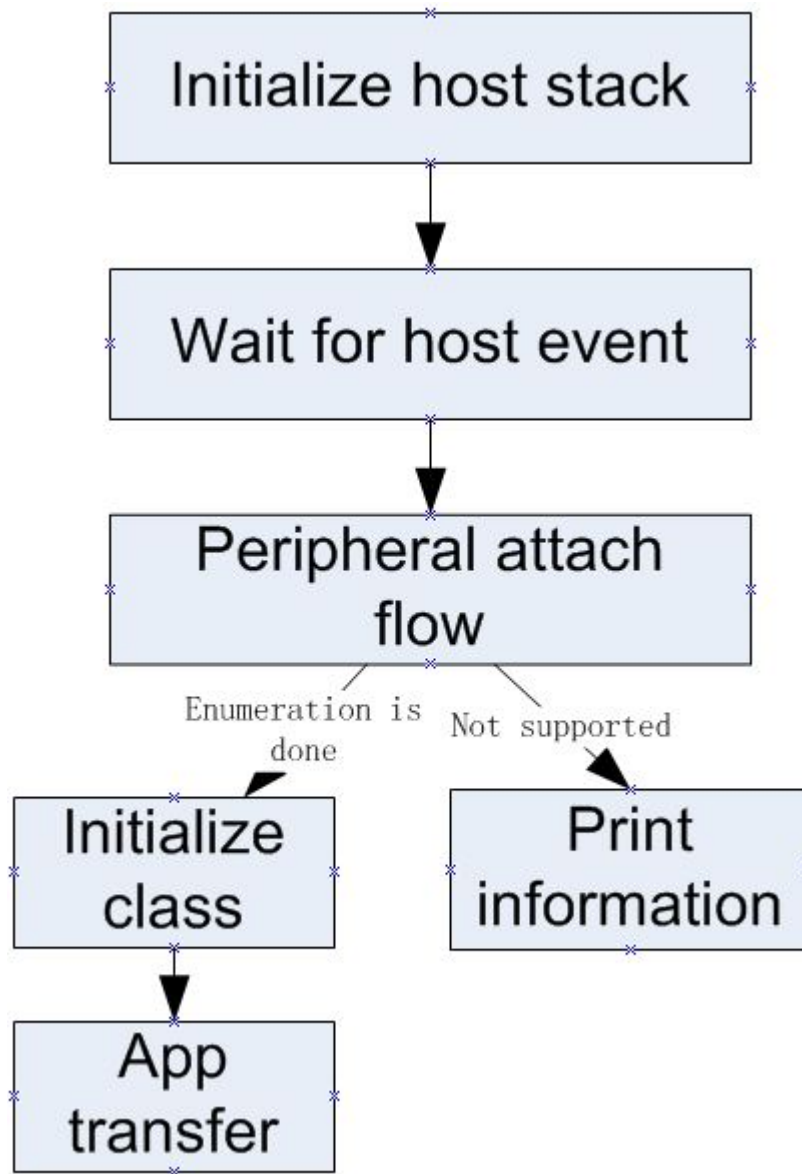
Figure 1.2.1: Host stack work flow

The host stack initialization work flow is as follows:
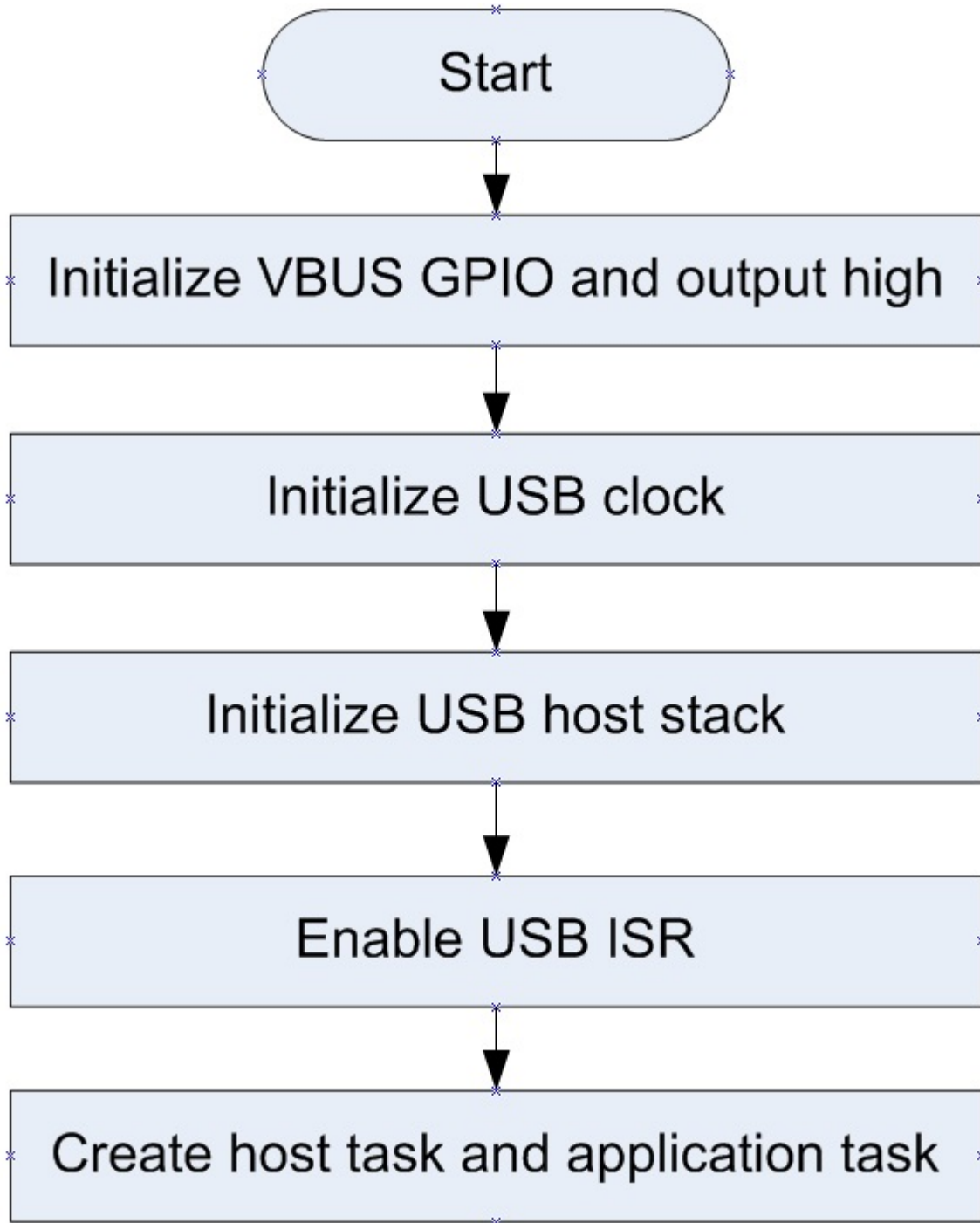
Figure 1.2.2: Host stack initialization flow

- 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.

1.3 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:
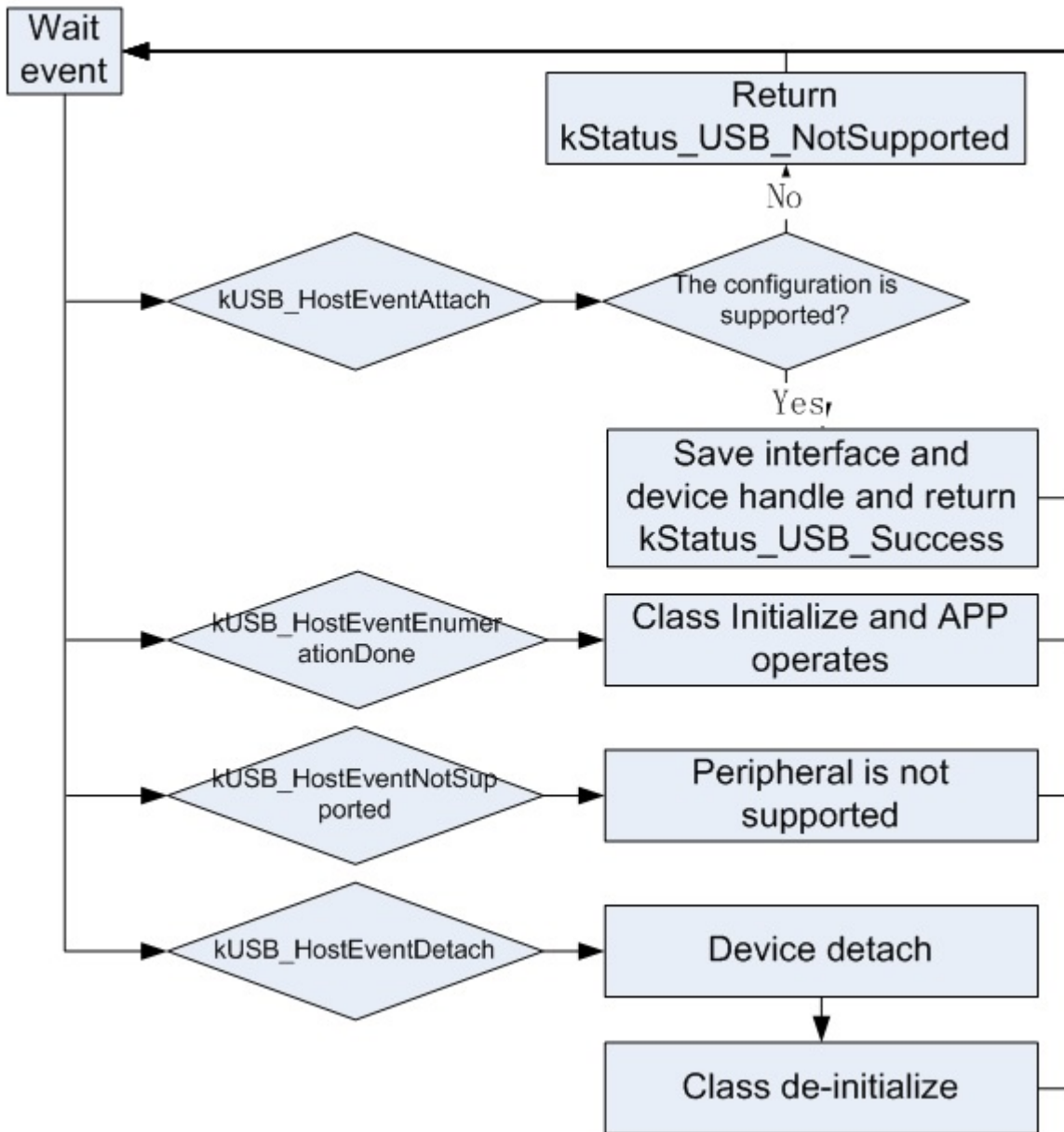
Figure 1.3.1: Host stack attach flow

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
Deprecated List

Global USB_HostAudioEndpointRequest (usb_host_class_handle classHandle, void ∗buf, uint32_t cmdCode, transfer_callback_t callbackFn, void ∗callbackParam)

Do not use this function. It has been superceded by USB_HostAudioGetSetEndpointRequest.

Global USB_HostAudioFeatureUnitRequest (usb_host_class_handle classHandle, uint8_t channel-No, void ∗buf, uint32_t cmdCode, transfer_callback_t callbackFn, void ∗callbackParam)

Do not use this function. It has been superceded by USB_HostAudioGetSetFeatureUnitRequest.

Global USB_HostAudioStreamGetCurrentAltsettingDescriptors (usb_host_class_handle class-Handle, usb_audio_stream_spepific_as_intf_desc_t ∗∗asIntfDesc, usb_audio_stream_format_-type_desc_t ∗∗formatTypeDesc, usb_descriptor_endpoint_t ∗∗isoEndpDesc)

Do not use this function. It has been superceded by USB_HostAudioStreamGetCurrentAltsetting-SpecificDescriptors.

MCUXpresso SDK USB Stack Host Reference Manual NXP Semiconductors

Chapter 3
Definitions and structures

3.1   Overview

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

Data Structures

- struct usb_version_t
  USB stack version fields. More...

Macros

- #define USB_STACK_VERSION_MAJOR (0x01UL)
  Defines USB stack major version.
- #define USB_STACK_VERSION_MINOR (0x00UL)
  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.
- #define USB_STACK_COMPONENT_VERSION MAKE_VERSION(USB_STACK_VERSION_MAJOR, USB_STACK_VERSION_MINOR, USB_STACK_VERSION_BUGFIX)
  USB stack component version definition, changed with component in yaml together.

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 ,
  kStatus_USB_DataOverRun }
  	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 = 5U,
  kUSB_ControllerLpcIp3511Hs0 = 6U,
  kUSB_ControllerLpcIp3511Hs1 = 7U,
  kUSB_ControllerOhci0 = 8U,
  kUSB_ControllerOhci1 = 9U,
  kUSB_ControllerIp3516Hs0 = 10U,
  kUSB_ControllerIp3516Hs1 = 11U,
  kUSB_ControllerDwc30 = 12U,
  kUSB_ControllerDwc31 = 13U }
  	USB controller ID.

3.2 Data Structure Documentation

3.2.1 struct usb_version_t

Data Fields

- uint8_t major

Major.
- uint8_t minor
    Minor.
- uint8_t bugfix
    Bug fix.

3.3 Typedef Documentation

3.3.1 typedef void∗ usb_device_handle

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

3.4 Enumeration Type Documentation

3.4.1 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.
kStatus_USB_DataOverRun The amount of data returned by the endpoint exceeded either the size of the maximum data packet allowed from the endpoint or the remaining buffer size.

3.4.2 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.

kUSB_ControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

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.

kUSB_ControllerDwc30 DWC3 0U.

kUSB_ControllerDwc31 DWC3 1U Currently, there are no platforms which have two Dwc IPs, this is reserved to be used in the future.

Chapter 4
USB Host driver

4.1   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_process_descriptor_param_t

    For USB_REQUEST_STANDARD_GET_DESCRIPTOR and USB_REQUEST_STANDARD_SET_DESCRIPTOR. More...
- struct usb_host_process_feature_param_t

    For USB_REQUEST_STANDARD_CLEAR_FEATURE and USB_REQUEST_STANDARD_SET_FEATURE. 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_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_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,
  kUSB_HostEventEnumerationFail,
  kUSB_HostEventNotSuspended,
  kUSB_HostEventSuspended,
  kUSB_HostEventNotResumed,
  kUSB_HostEventDetectResume,
  kUSB_HostEventResumed,
  kUSB_HostEventL1Sleeped,
  kUSB_HostEventL1SleepNYET,
  kUSB_HostEventL1SleepNotSupport,
  kUSB_HostEventL1SleepError,
  kUSB_HostEventL1NotResumed,
  kUSB_HostEventL1DetectResume,
  kUSB_HostEventL1Resumed }

  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_request_type_t {

kRequestDevice = 1U,

kRequestInterface,

kRequestEndpoint }

 Request type.
- 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.

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 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.
- usb_status_t USB_HostSuspendDeviceResquest (usb_host_handle hostHandle, usb_device_handle deviceHandle)
    Send a bus or device suspend request.
- usb_status_t USB_HostResumeDeviceResquest (usb_host_handle hostHandle, usb_device_handle deviceHandle)
    Send a bus or device resume request.
- usb_status_t USB_HostL1SleepDeviceResquest (usb_host_handle hostHandle, usb_device_handle deviceHandle, uint8_t sleepType)
    Send a bus or device suspend request.
- usb_status_t USB_HostL1ResumeDeviceResquest (usb_host_handle hostHandle, usb_device_handle deviceHandle, uint8_t sleepType)

Send a bus or device resume request.
- usb_status_t USB_HostL1SleepDeviceResquestConfig (usb_host_handle hostHandle, uint8_t
  ∗lpmParam)
    Update the lpm param.
- usb_status_t USB_HostUpdateHwTick (usb_host_handle hostHandle, uint64_t tick)
    Update the hardware tick.

4.2   Data Structure Documentation

4.2.1   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.
- uint16_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.

4.2.2   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.

4.2.3   struct usb_host_get_interface_param_t

Data Fields

- uint8_t interface
    Interface number.
- uint8_t ∗ alternateInterfaceBuffer
    Save the transfer result.

4.2.4   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.

4.2.5   struct usb_host_set_interface_param_t

Data Fields

- uint8_t alternateSetting
    Alternate setting value.
- uint8_t interface
    Interface number.

4.2.6   struct usb_host_synch_frame_param_t

Data Fields

- uint8_t endpoint
    Endpoint number.
- uint8_t ∗ frameNumberBuffer
    Frame number data buffer.

4.2.7   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.

4.2.8   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.

4.2.9   struct usb_host_configuration_t

Data Fields

- usb_host_interface_t interfaceList [USB_HOST_CONFIG_CONFIGURATION_MAX_INTERF-ACE]
  Interface array.
- usb_descriptor_configuration_t ∗ configurationDesc
  Configuration descriptor pointer.
- uint8_t ∗ configurationExtension
  Configuration extended descriptor pointer.
- uint16_t configurationExtensionLength
  Extended descriptor length.
- uint8_t interfaceCount
  The configuration's interface number.

4.2.10   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.

4.2.11   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.
- uint16_t nakTimeout

    KHCI transfer NAK timeout.
- uint16_t retry

    KHCI transfer retry.
- 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.

4.2.12 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_ENDP-
    OINT_ISOCHRONOUS, USB_ENDPOINT_BULK.
- uint8_t numberPerUframe
    Transaction number for each micro-frame.

4.2.12.0.0.1 Field Documentation

4.2.12.0.0.1.1 uint16_t usb_host_pipe_init_t::nakCount

MUST be zero for interrupt

4.2.13 struct usb_host_cancel_param_t

Data Fields

- usb_host_pipe_handle pipeHandle
    Canceling pipe handle.
- usb_host_transfer_t ∗ transfer
    Canceling transfer.

4.2.14 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.
- uint8_t * enumBuffer
    Buffer for enumeration.
- uint16_t configurationLen
    Configuration descriptor length.
- uint8_t interfaceStatus [USB_HOST_CONFIG_CONFIGURATION_MAX_INTERFACE]
    Interfaces' status, please reference to usb_host_interface_state_t.
- uint8_t configurationValue
    Configuration index.
- 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.
- uint8_t hubNumber
    Device's first connected hub address (root hub = 0)
- uint8_t portNumber
    Device's first connected hub's port no (1 - 8)
- uint8_t hsHubNumber
    Device's first connected high-speed hub's address (1 - 8)
- uint8_t hsHubPort
    Device's first connected high-speed hub's port no (1 - 8)
- uint8_t level
    Device's level (root device = 0)

4.2.14.0.0.2   Field Documentation

4.2.14.0.0.2.1   uint8_t usb_host_device_instance_t::allocatedAddress

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

4.2.15   struct usb_host_instance_t

Data Fields

- void ∗ controllerHandle
    The low level controller handle.
- host_callback_t deviceCallback
    Device attach/detach callback.
- osa_mutex_handle_t hostMutex
    Host layer mutex.
- uint32_t mutexBuffer [(OSA_MUTEX_HANDLE_SIZE+3)/4]
    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.
- void ∗ suspendedDevice
    Suspended device handle.
- volatile uint64_t hwTick
    Current hw tick(ms)
- uint8_t sleepType
    L1 LPM device handle.
- 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.

4.2.15.0.0.3   Field Documentation

4.2.15.0.0.3.1   uint8_t usb_host_instance_t::addressBitMap[16]

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

4.3 Typedef Documentation

4.3.1 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.

Typedef Documentation

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.

4.3.2 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.

4.3.3 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.

4.4 Enumeration Type Documentation

4.4.1 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.

kUSB_HostEventEnumerationFail Device's enumeration failed due to errors fail reason is put in the high 2 bytes of callback event code. kStatus_USB_TransferFailed - the transfer failed. kStatus_USB_TransferCancel - transfer is canceled by application. kStatus_USB_Error - parsing descriptor failed, the power cannot satisfy device's requirement, device addresss allocation failed, transfer is not enough or the transfer API failed. kStatus_USB_AllocFail - malloc failed.

kUSB_HostEventNotSuspended Suspend failed.

kUSB_HostEventSuspended Suspend successful.

kUSB_HostEventNotResumed Resume failed.

kUSB_HostEventDetectResume Detect resume signal.

kUSB_HostEventResumed Resume successful.

kUSB_HostEventL1Sleeped L1 Sleep successful,state transition was successful (ACK)

kUSB_HostEventL1SleepNYET Device was unable to enter the L1 state at this time (NYET)

kUSB_HostEventL1SleepNotSupport Device does not support the L1 state (STALL)

kUSB_HostEventL1SleepError Device failed to respond or an error occurred.

kUSB_HostEventL1NotResumed Resume failed.

kUSB_HostEventL1DetectResume Detect resume signal.

kUSB_HostEventL1Resumed Resume successful.

4.4.2 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.

Enumeration Type Documentation

    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.

4.4.3 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.

4.4.4 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.

4.4.5 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.

4.4.6 enum usb_host_device_state_t

Enumerator

> ***kStatus_device_Detached*** Device is used by application.
> ***kStatus_device_Attached*** Device's default status.

4.5 Function Documentation

4.5.1 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.

4.5.2 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.

4.5.3 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.

4.5.4 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.

4.5.5 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.

4.5.6 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.

4.5.7 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.

4.5.8 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.

4.5.9 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.

4.5.10   void USB_HostKhciIsrFunction (void ∗ *hostHandle*)

The function is the KHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

4.5.11 void USB_HostEhciIsrFunction (void ∗ *hostHandle*)

The function is the EHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

4.5.12 void USB_HostOhciIsrFunction (void ∗ *hostHandle*)

The function is the OHCI interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

4.5.13 void USB_HostIp3516HsIsrFunction (void ∗ *hostHandle*)

The function is the IP3516HS interrupt service routine.

Parameters

in	*hostHandle*	The host handle.

4.5.14 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.

4.5.15 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.

4.5.16 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.

4.5.17   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.

4.5.18   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.

Function Documentation

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.

4.5.19   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.

4.5.20   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.

4.5.21 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.

4.5.22 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

Function Documentation

in	*deviceHandle*	The device handle for control transfer.
in	*usbRequest*	A USB standard request code. See 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_Invalid-Handle	The deviceHandle 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, Or, the request is not standard request.

4.5.23 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	*interface-Handle*	Opening interface handle.

Return values

kStatus_USB_Success	Open successfully.
kStatus_USB_Invalid-Handle	The deviceHandle or interfaceHandle is a NULL pointer.

4.5.24 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.

MCUXpresso SDK USB Stack Host Reference Manual NXP Semiconductors

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.

4.5.25 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.

4.5.26 usb_status_t USB_HostSuspendDeviceResquest (usb_host_handle *hostHandle,* usb_device_handle *deviceHandle*)

This function is used to send a bus or device suspend request.

Parameters

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

Return values

kStatus_USB_Success	Request successfully.
kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer. Or the controller handle is invalid.

Function Documentation

kStatus_USB_Error	There is no idle transfer. Or, the deviceHandle is invalid. Or, the request is invalid.

4.5.27   usb_status_t USB_HostResumeDeviceResquest (usb_host_handle *hostHandle,* usb_device_handle *deviceHandle*)

This function is used to send a bus or device resume request.

Parameters

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

Return values

kStatus_USB_Success	Request successfully.
kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer. Or the controller handle is invalid.
kStatus_USB_Error	There is no idle transfer. Or, the deviceHandle is invalid. Or, the request is invalid.

4.5.28   usb_status_t USB_HostL1SleepDeviceResquest (usb_host_handle *hostHandle,* usb_device_handle *deviceHandle,* uint8_t *sleepType*)

This function is used to send a bus or device suspend request.

Parameters

in	*hostHandle*	The host handle.
in	*deviceHandle*	The device handle.
in	*sleeptype*	Bus suspend or single device suspend.

Return values

kStatus_USB_Success	Request successfully.

kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer. Or the controller handle is invalid.
kStatus_USB_Error	There is no idle transfer. Or, the deviceHandle is invalid. Or, the request is invalid.

4.5.29   usb_status_t USB_HostL1ResumeDeviceResquest (usb_host_handle *hostHandle,* usb_device_handle *deviceHandle,* uint8_t *sleepType*)

This function is used to send a bus or device resume request.

Parameters

in	*hostHandle*	The host handle.
in	*deviceHandle*	The device handle. ∗
in	*sleeptype*	Bus suspend or single device suspend.

Return values

kStatus_USB_Success	Request successfully.
kStatus_USB_Invalid-Handle	The hostHandle is a NULL pointer. Or the controller handle is invalid.
kStatus_USB_Error	There is no idle transfer. Or, the deviceHandle is invalid. Or, the request is invalid.

4.5.30   usb_status_t USB_HostL1SleepDeviceResquestConfig (usb_host_handle *hostHandle,* uint8_t ∗ *lpmParam*)

The function is used to configure the lpm token.

Parameters

in	*hostHandle*	The host handle.
in	*lpmParam*	HIRD value and whether enable remotewakeup.

4.5.31   usb_status_t USB_HostUpdateHwTick (usb_host_handle *hostHandle,* uint64_t *tick*)

The function is used to update the hardware tick.

Function Documentation

Parameters

in	*hostHandle*	The host handle.
in	*tick*	Current hardware tick(uint is ms).

4.5.32 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.

4.5.33 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.

4.5.34 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.

4.5.35 uint8_t USB_HostGetDeviceAttachState (usb_device_handle *deviceHandle*)

Parameters

deviceHandle	Device handle.

Returns

0x01 - attached; 0x00 - detached.

4.5.36 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.

4.6 USB Host Controller driver

4.6.1 Overview

The USB Host controller driver implements the real send/receive function. Implementations are different for different controllers. The supported controller drivers are KHCI, EHCI,IP3516HS and so on.

Modules

- USB Host Controller EHCI driver
- USB Host Controller IP3516HS driver
- USB Host Controller KHCI driver
- USB Host Controller OHCI 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,
  kUSB_HostL1Config,
  kUSB_HostSetChargerType }
    USB host controller control code.
- enum usb_host_bus_control_t {
  kUSB_HostBusReset = 1U,
  kUSB_HostBusRestart,
  kUSB_HostBusEnableAttach,
  kUSB_HostBusDisableAttach,
  kUSB_HostBusSuspend,
  kUSB_HostBusResume,
  kUSB_HostBusL1SuspendInit,
  kUSB_HostBusL1Sleep,
  kUSB_HostBusL1Resume }
    USB host controller bus control code.

4.6.2 Data Structure Documentation

4.6.2.1 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.

4.6.3 Enumeration Type Documentation

4.6.3.1 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.
    kUSB_HostL1Config   L1 suspend Bus control code.
    kUSB_HostSetChargerType   set charger type

4.6.3.2   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.
> ***kUSB_HostBusL1SuspendInit***   L1 Suspend BUS.
> ***kUSB_HostBusL1Sleep***   L1 Suspend BUS.
> ***kUSB_HostBusL1Resume***   L1 Resume BUS.

4.6.4 USB Host Controller KHCI driver

4.6.4.1 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 0x65U
  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.

4.6.4.2 Data Structure Documentation

4.6.4.2.1 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.
- uint32_t taskEventHandleBuffer [(OSA_EVENT_HANDLE_SIZE+3)/4]
    KHCI task event handle buffer.
- osa_mutex_handle_t khciMutex
    KHCI mutex.
- uint32_t mutexBuffer [(OSA_MUTEX_HANDLE_SIZE+3)/4]
    The mutex buffer.
- 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.
- bus_suspend_request_state_t busSuspendStatus
    Bus Suspend Status.
- int8_t deviceAttached
    Device attach/detach state.

4.6.4.2.1.1 Field Documentation

4.6.4.2.1.1.1 uint32_t ptr_usb_host_khci_state_struct_t::mutexBuffer[(OSA_MUTEX_HANDLE_SI-ZE+3)/4]

4.6.4.3 Macro Definition Documentation

4.6.4.3.1 #define KHCICFG_THSLD_DELAY 0x65U

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.

4.6.4.4 Function Documentation

4.6.4.4.1 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.

4.6.4.4.2 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.

4.6.4.4.3 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.

4.6.4.4.4 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.

4.6.4.4.5 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.

4.6.4.4.6 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.

4.6.4.4.7 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.

4.6.5   USB Host Controller EHCI driver

4.6.5.1   Overview

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

Data Structures

- struct usb_host_ehci_pipe_t

  EHCI pipe structure. More...
- struct usb_host_ehci_qh_t

  EHCI QH structure. More...
- struct usb_host_ehci_qtd_t

  EHCI QTD structure. More...
- struct usb_host_ehci_itd_t

  EHCI ITD structure. More...
- struct usb_host_ehci_sitd_t

  EHCI SITD structure. More...
- struct usb_host_ehci_iso_t

  EHCI ISO structure; An ISO pipe has an instance of this structure to keep the ISO pipe-specific information. More...
- struct usb_host_ehci_instance_t

  EHCI instance structure. More...
- struct usb_host_ehci_data_t

  EHCI data structure. More...

Macros

- #define USB_HOST_EHCI_ISO_NUMBER USB_HOST_CONFIG_EHCI_MAX_ITD

  The maximum supported ISO pipe number.
- #define USB_HOST_EHCI_PORT_CONNECT_DEBOUNCE_DELAY (101U)

  Check the port connect state delay if the state is unstable.
- #define USB_HOST_EHCI_PORT_RESET_DELAY (11U)

  Delay for port reset.
- #define USB_HOST_EHCI_ISO_MAX_CONTINUOUS_TRANSFER (8U)

  The MAX continuous transfers that application can send.
- #define USB_HOST_EHCI_ISO_BOUNCE_FRAME_NUMBER (2U)

  The SITD inserts a frame interval for putting more SITD continuously.
- #define USB_HOST_EHCI_ISO_BOUNCE_UFRAME_NUMBER (16U)

  The ITD inserts a micro-frame interval for putting more ITD continuously.
- #define USB_HOST_EHCI_CONTROL_BULK_TIME_OUT_VALUE (50U)

  Control or bulk transaction timeout value (unit: 100 ms)

Enumerations

- enum host_ehci_device_state_t {
  kEHCIDevicePhyAttached = 1,
  kEHCIDeviceAttached,

kEHCIDeviceDetached }
> *EHCI state for device attachment/detachment.*

USB host EHCI APIs

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

4.6.5.2 Data Structure Documentation

4.6.5.2.1 struct usb_host_ehci_pipe_t

Data Fields

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

- uint8_t uframeSmask
    Start micro-frame.
- uint8_t uframeCmask
    Complete micro-frame.

4.6.5.2.1.1 Field Documentation

4.6.5.2.1.1.1 uint16_t usb_host_ehci_pipe_t::dataTime

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

4.6.5.2.1.1.2 uint16_t usb_host_ehci_pipe_t::startSplitTime

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

4.6.5.2.1.1.3 uint16_t usb_host_ehci_pipe_t::completeSplitTime

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

4.6.5.2.1.1.4 uint8_t usb_host_ehci_pipe_t::uframeSmask

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

4.6.5.2.1.1.5 uint8_t usb_host_ehci_pipe_t::uframeCmask

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

4.6.5.2.2 struct usb_host_ehci_qh_t

See the USB EHCI specification

Data Fields

- uint32_t horizontalLinkPointer
    QH specification filed, queue head a horizontal link pointer.
- uint32_t staticEndpointStates [2]
    QH specification filed, static endpoint state and configuration information.
- uint32_t currentQtdPointer
    QH specification filed, current qTD pointer.
- uint32_t nextQtdPointer
    QH specification filed, next qTD pointer.

- uint32_t alternateNextQtdPointer
    QH specification filed, alternate next qTD pointer.
- uint32_t transferOverlayResults [6]
    QH specification filed, transfer overlay configuration and transfer results.
- usb_host_ehci_pipe_t ∗ ehciPipePointer
    EHCI pipe pointer.
- usb_host_transfer_t ∗ ehciTransferHead
    Transfer list head on this QH.
- usb_host_transfer_t ∗ ehciTransferTail
    Transfer list tail on this QH.
- uint16_t timeOutValue
    Its maximum value is USB_HOST_EHCI_CONTROL_BULK_TIME_OUT_VALUE.
- uint16_t timeOutLabel
    It's used to judge the transfer timeout.

4.6.5.2.2.1 Field Documentation

4.6.5.2.2.1.1 uint16_t usb_host_ehci_qh_t::timeOutValue

When the value is zero, the transfer times out.

4.6.5.2.2.1.2 uint16_t usb_host_ehci_qh_t::timeOutLabel

The EHCI driver maintain the value

4.6.5.2.3 struct usb_host_ehci_qtd_t

See the USB EHCI specification.

Data Fields

- uint32_t nextQtdPointer
    QTD specification filed, the next QTD pointer.
- uint32_t alternateNextQtdPointer
    QTD specification filed, alternate next QTD pointer.
- uint32_t transferResults [2]
    QTD specification filed, transfer results fields.
- uint32_t bufferPointers [4]
    QTD specification filed, transfer buffer fields.

4.6.5.2.4 struct usb_host_ehci_itd_t

See the USB EHCI specification.

Data Fields

- uint32_t nextLinkPointer

> *ITD specification filed, the next linker pointer.*
- uint32_t transactions [8]
  > *ITD specification filed, transactions information.*
- uint32_t bufferPointers [7]
  > *ITD specification filed, transfer buffer fields.*
- struct _usb_host_ehci_itd ∗ nextItdPointer
  > *Next ITD pointer.*
- uint32_t frameEntryIndex
  > *The ITD inserted frame value.*
- uint32_t reserved [6]
  > *Reserved fields for 32 bytes align.*

4.6.5.2.5 struct usb_host_ehci_sitd_t

See the USB EHCI specification.

Data Fields

- uint32_t nextLinkPointer
  > *SITD specification filed, the next linker pointer.*
- uint32_t endpointStates [2]
  > *SITD specification filed, endpoint configuration information.*
- uint32_t transferResults [3]
  > *SITD specification filed, transfer result fields.*
- uint32_t backPointer
  > *SITD specification filed, back pointer.*
- uint16_t frameEntryIndex
  > *The SITD inserted frame value.*
- uint8_t nextSitdIndex
  > *The next SITD index; Get the next SITD pointer through adding base address with the index.*
- uint8_t reserved
  > *Reserved fields for 32 bytes align.*

4.6.5.2.5.1 Field Documentation

4.6.5.2.5.1.1 uint8_t usb_host_ehci_sitd_t::nextSitdIndex

0xFF means invalid.

4.6.5.2.6 struct usb_host_ehci_iso_t

Data Fields

- struct _usb_host_ehci_iso ∗ next
  > *Next instance pointer.*
- usb_host_pipe_t ∗ ehciPipePointer
  > *This ISO's EHCI pipe pointer.*
- usb_host_transfer_t ∗ ehciTransferHead

> *Transfer list head on this ISO pipe.*
- usb_host_transfer_t ∗ ehciTransferTail
  > *Transfer list head on this ISO pipe.*
- uint16_t lastLinkFrame
  > *It means that the inserted frame for ISO ITD/SITD.*

4.6.5.2.6.1 Field Documentation

4.6.5.2.6.1.1 uint16_t usb_host_ehci_iso_t::lastLinkFrame

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

4.6.5.2.7 struct usb_host_ehci_instance_t

Data Fields

- usb_host_handle hostHandle
  > *Related host handle.*
- uint32_t ∗ ehciUnitBase
  > *Keep the QH/QTD/ITD/SITD buffer pointer for release.*
- uint8_t ∗ ehciFrameList
  > *The frame list of the current ehci instance.*
- usb_host_ehci_qh_t ∗ ehciQhList
  > *Idle QH list pointer.*
- usb_host_ehci_qtd_t ∗ ehciQtdHead
  > *Idle QTD list pointer head.*
- usb_host_ehci_qtd_t ∗ ehciQtdTail
  > *Idle QTD list pointer tail (recently used qTD will be used)*
- usb_host_ehci_itd_t ∗ ehciItdList
  > *Idle ITD list pointer.*
- usb_host_ehci_sitd_t ∗ ehciSitdIndexBase
  > *SITD buffer's start pointer.*
- usb_host_ehci_sitd_t ∗ ehciSitdList
  > *Idle SITD list pointer.*
- usb_host_ehci_iso_t ∗ ehciIsoList
  > *Idle ISO list pointer.*
- USBHS_Type ∗ ehciIpBase
  > *EHCI IP base address.*
- usb_host_ehci_qh_t ∗ shedFirstQh
  > *First async QH.*
- usb_host_ehci_pipe_t ∗ ehciPipeIndexBase
  > *Pipe buffer's start pointer.*
- usb_host_ehci_pipe_t ∗ ehciPipeList
  > *Idle pipe list pointer.*
- usb_host_ehci_pipe_t ∗ ehciRunningPipeList
  > *Running pipe list pointer.*
- osa_mutex_handle_t ehciMutex
  > *EHCI mutex.*
- uint32_t mutexBuffer [(OSA_MUTEX_HANDLE_SIZE+3)/4]
  > *The mutex buffer.*

- osa_event_handle_t taskEventHandle

    EHCI task event.
- uint32_t taskEventHandleBuffer [(OSA_EVENT_HANDLE_SIZE+3)/4]

    EHCI task event handle buffer.
- uint8_t controllerId

    EHCI controller ID.
- uint8_t deviceAttached

    Device attach/detach state, see host_ehci_device_state_t.
- uint8_t firstDeviceSpeed

    The first device's speed, the controller's work speed.
- uint8_t ehciItdNumber

    Idle ITD number.
- uint8_t ehciSitdNumber

    Idle SITD number.
- uint8_t ehciQtdNumber

    Idle QTD number.
- bus_ehci_suspend_request_state_t busSuspendStatus

    Bus Suspend Status.

4.6.5.2.7.1   Field Documentation

4.6.5.2.7.1.1   uint32_t usb_host_ehci_instance_t::mutexBuffer[(OSA_MUTEX_HANDLE_SIZE+3)/4]

4.6.5.2.8   struct usb_host_ehci_data_t

4.6.5.3   Macro Definition Documentation

4.6.5.3.1   #define USB_HOST_EHCI_ISO_MAX_CONTINUOUS_TRANSFER (8U)

4.6.5.3.2   #define USB_HOST_EHCI_ISO_BOUNCE_FRAME_NUMBER (2U)

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

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

4.6.5.3.3   #define USB_HOST_EHCI_ISO_BOUNCE_UFRAME_NUMBER (16U)

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

- Use case 1: when inserting ITD first, the inserted micro-frame = the current micro-frame value +

this MACRO value.

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

4.6.5.4 Enumeration Type Documentation

4.6.5.4.1 enum host_ehci_device_state_t

Enumerator

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

4.6.5.5 Function Documentation

4.6.5.5.1 usb_status_t USB_HostEhciCreate (uint8_t *controllerId,* usb_host_handle *upperLayerHandle,* usb_host_controller_handle ∗ *controllerHandle*)

This function initializes the USB host EHCI controller driver.

Parameters

in	*controllerId*	The controller ID of the USB IP. Please refer to the enumeration usb_-controller_index_t.
in	*upperLayer-Handle*	The host level handle.
out	*controller-Handle*	return the controller instance handle.

Return values

kStatus_USB_Success	The host is initialized successfully.
kStatus_USB_AllocFail	Allocating memory failed.
kStatus_USB_Error	Host mutex create fail, KHCI/EHCI mutex or KHCI/EHCI event create fail. Or, KHCI/EHCI IP initialize fail.

4.6.5.5.2 usb_status_t USB_HostEhciDestory (usb_host_controller_handle *controllerHandle*)

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

Parameters

in	*controller-Handle*	The controller handle.

Return values

kStatus_USB_Success	The host is initialized successfully.

4.6.5.5.3 usb_status_t USB_HostEhciOpenPipe (usb_host_controller_handle *controllerHandle,* 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	*controller-Handle*	The controller handle.
out	*pipeHandle*	The pipe handle pointer, it is used to return the pipe handle.
in	*pipeInit*	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. Or, there is no idle QH for EHCI. Or, bandwidth allocate fail for EHCI.

4.6.5.5.4 usb_status_t USB_HostEhciClosePipe (usb_host_controller_handle *controllerHandle,* usb_host_pipe_handle *pipeHandle*)

This function closes a pipe and releases related resources.

Parameters

in	*controller-Handle*	The controller handle.

in	*pipeHandle*	The closing pipe handle.

Return values

kStatus_USB_Success	The host is initialized successfully.

4.6.5.5.5 usb_status_t USB_HostEhciWritePipe (usb_host_controller_handle *controllerHandle,* usb_host_pipe_handle *pipeHandle,* usb_host_transfer_t ∗ *transfer*)

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

Parameters

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

Return values

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

4.6.5.5.6 usb_status_t USB_HostEhciReadpipe (usb_host_controller_handle *controllerHandle,* usb_host_pipe_handle *pipeHandle,* usb_host_transfer_t ∗ *transfer*)

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

Parameters

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

Return values

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

4.6.5.5.7 usb_status_t USB_HostEhciIoctl (usb_host_controller_handle *controllerHandle,* uint32_t *ioctlEvent,* void ∗ *ioctlParam*)

This function controls the EHCI.

Parameters

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

Return values

kStatus_USB_Success	Cancel successfully.
kStatus_USB_Invalid-Handle	The controllerHandle is a NULL pointer.

4.6.6   USB Host Controller OHCI driver

4.6.6.1   Overview

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

Data Structures

- struct usb_host_ohci_endpoint_descritpor_struct_t
    OHCI Endpoint Descriptor. More...
- struct usb_host_ohci_general_transfer_descritpor_struct_t
    OHCI General Transfer Descriptor. More...
- struct usb_host_ohci_isochronous_transfer_descritpor_struct_t
    OHCI Isochronous Transfer Descriptor. More...
- struct usb_host_ohci_hcca_struct_t
    OHCI Host Controller Communications Area. More...
- struct usb_host_ohci_hcor_struct_t
    OHCI Host Controller Operational Registers. More...
- struct usb_host_ohci_state_struct_t
    OHCI controller driver instance structure. More...

Macros

- #define USB_HOST_OHCI_EVENT_ATTACH (0x01U)
    task event
- #define USB_HOST_OHCI_PORT_CONNECT_DEBOUNCE_DELAY (101U)
    Check the port connect state delay if the state is unstable.
- #define USB_HOST_OHCI_TRANSFER_SCAN_INTERVAL (10U)
    Transfer scan interval (ms)
- #define USB_HOST_OHCI_TRANSFER_TIMEOUT_GAP (5000U / USB_HOST_OHCI_TRA-NSFER_SCAN_INTERVAL)
    Time out gap for each transfer (USB_HOST_OHCI_TRANSFER_SCAN_INTERVAL ∗ 1ms)
- #define USB_HostOhciLock() (void)OSA_MutexLock(usbHostState->mutex, USB_OSA_WAIT-_TIMEOUT)
    USB host OHCI lock.
- #define USB_HostOhciUnlock() (void)OSA_MutexUnlock(usbHostState->mutex)
    USB host OHCI unlock.

Enumerations

- enum usb_host_ohci_port_state_t {
    kUSB_DeviceOhciPortDetached = 0U,
    kUSB_DeviceOhciPortPhyDetached,
    kUSB_DeviceOhciPortPhyAttached,
    kUSB_DeviceOhciPortAttached }
    Port state for device attachment/detachment.

USB host OHCI APIs

- usb_status_t USB_HostOhciCreate (uint8_t controllerId, usb_host_handle hostHandle, usb_host_controller_handle ∗controllerHandle)

    Creates the USB host OHCI instance.
- usb_status_t USB_HostOhciDestory (usb_host_controller_handle controllerHandle)

    Destroys the USB host OHCI instance.
- usb_status_t USB_HostOhciOpenPipe (usb_host_controller_handle controllerHandle, usb_host_pipe_handle ∗pipeHandle, usb_host_pipe_init_t ∗pipeInit)

    Opens the USB host pipe.
- usb_status_t USB_HostOhciClosePipe (usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle)

    Closes the USB host pipe.
- usb_status_t USB_HostOhciWritePipe (usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle, usb_host_transfer_t ∗transfer)

    Sends data to the pipe.
- usb_status_t USB_HostOhciReadPipe (usb_host_controller_handle controllerHandle, usb_host_pipe_handle pipeHandle, usb_host_transfer_t ∗transfer)

    Receives data from the pipe.
- usb_status_t USB_HostOhciIoctl (usb_host_controller_handle controllerHandle, uint32_t ioctl-Event, void ∗ioctlParam)

    Controls the OHCI.

4.6.6.2 Data Structure Documentation

4.6.6.2.1 struct usb_host_ohci_endpoint_descritpor_struct_t

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.

4.6.6.2.1.1 Field Documentation

4.6.6.2.1.1.1 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::D

4.6.6.2.1.1.2 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::S

4.6.6.2.1.1.3 uint32_t usb_host_ohci_endpoint_descritpor_struct_t::F

4.6.6.2.2 struct usb_host_ohci_general_transfer_descritpor_struct_t

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.

4.6.6.2.3 struct usb_host_ohci_isochronous_transfer_descritpor_struct_t

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.*

4.6.6.2.4   struct usb_host_ohci_hcca_struct_t

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.*

4.6.6.2.5   struct usb_host_ohci_hcor_struct_t

4.6.6.2.6   struct usb_host_ohci_state_struct_t

Data Fields

- volatile
  usb_host_ohci_hcor_struct_t ∗ usbRegBase
    - *The base address of the register.*
- void ∗ hostHandle
    - *Related host handle.*
- osa_event_handle_t ohciEvent
    - *OHCI event.*
- uint32_t taskEventHandleBuffer [(OSA_EVENT_HANDLE_SIZE+3)/4]
    - *task event handle buffer*
- osa_mutex_handle_t 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.
- volatile uint8_t controlIsBusy

    Is handling control transfer.
- volatile uint8_t isrLevel

    The disable deep of the ISR.
- volatile
  bus_ohci_suspend_request_state_t busSuspendStatus

    Bus Suspend Status.

4.6.6.3   Enumeration Type Documentation

4.6.6.3.1   enum usb_host_ohci_port_state_t

Enumerator

kUSB_DeviceOhciPortDetached Port is detached and de-initialized.
kUSB_DeviceOhciPortPhyDetached Port is physically detached.
kUSB_DeviceOhciPortPhyAttached Port is physically attached.
kUSB_DeviceOhciPortAttached Port is attached and initialized.

4.6.6.4   Function Documentation

4.6.6.4.1   usb_status_t USB_HostOhciCreate (uint8_t *controllerId,* usb_host_handle *hostHandle,* usb_host_controller_handle ∗ *controllerHandle*)

This function initializes the USB host OHCI 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, OHCI mutex or OHCI event create failed. Or, OHCI IP initialize failed.

4.6.6.4.2   usb_status_t USB_HostOhciDestory (usb_host_controller_handle *controllerHandle*)

This function deinitializes the USB host OHCI controller driver.

Parameters

controller-Handle	The controller handle.

Return values

kStatus_USB_Success	The host is initialized successfully.

4.6.6.4.3   usb_status_t USB_HostOhciOpenPipe (usb_host_controller_handle *controllerHandle,* usb_host_pipe_handle ∗ *pipeHandle,* usb_host_pipe_init_t ∗ *pipeInit*)

This function opens a pipe according to the pipe_init_ptr parameter.

Parameters

controller-Handle	The controller handle.
pipeHandle	The pipe handle pointer used to return the pipe handle.
pipeInit	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.

4.6.6.4.4   usb_status_t USB_HostOhciClosePipe (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.

4.6.6.4.5   usb_status_t USB_HostOhciWritePipe (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 OHCI.

4.6.6.4.6   usb_status_t USB_HostOhciReadPipe (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 OHCI.

4.6.6.4.7 usb_status_t USB_HostOhciIoctl (usb_host_controller_handle *controllerHandle,* uint32_t *ioctlEvent,* void ∗ *ioctlParam*)

This function controls the OHCI.

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.

4.6.7   USB Host Controller IP3516HS driver

4.6.7.1   Overview

The IP3516HS host controller driver implements send/receive data through the IP3516HS IP.

Data Structures

- struct usb_host_ip3516hs_atl_struct_t
    IP3516HS Transaction Descriptor. More...
- struct usb_host_ip3516hs_ptl_struct_t
    IP3516HS Transaction Descriptor. More...
- struct usb_host_ip3516hs_sptl_struct_t
    IP3516HS Transaction Descriptor. More...
- struct usb_host_ip3516hs_register_struct_t
    IP3516HS Host Controller Operational Registers. More...
- struct usb_host_ip3516hs_state_struct_t
    IP3516HS controller driver instance structure. More...

Macros

- #define USB_HOST_IP3516HS_PORT_CONNECT_DEBOUNCE_DELAY (101U)
    Check the port connect state delay if the state is unstable.
- #define USB_HOST_IP3516HS_EVENT_ATTACH (0x01U)
    task event
- #define USB_HOST_IP3516HS_TRANSFER_SCAN_INTERVAL (200U)
    Transfer scan interval (ms)
- #define USB_HOST_IP3516HS_TRANSFER_TIMEOUT_GAP ((5000U ∗ 8U) / USB_HOST_I-P3516HS_TRANSFER_SCAN_INTERVAL)
    Time out gap for each transfer (USB_HOST_OHCI_TRANSFER_SCAN_INTERVAL ∗ 1ms)
- #define USB_HostIp3516HsLock() (void)OSA_MutexLock(usbHostState->mutex, USB_OSA_-WAIT_TIMEOUT)
    USB host Ip3516Hs lock.
- #define USB_HostIp3516HsUnlock() (void)OSA_MutexUnlock(usbHostState->mutex)
    USB host Ip3516Hs unlock.

Enumerations

- enum usb_host_ip3516hs_port_state_t {
  kUSB_DeviceIp3516HsPortDetached = 0U,
  kUSB_DeviceIp3516HsPortPhyDetached,
  kUSB_DeviceIp3516HsPortPhyAttached,
  kUSB_DeviceIp3516HsPortAttached }
    Port state for device attachment/detachment.

USB host IP3516HS APIs

- usb_status_t USB_HostIp3516HsCreate (uint8_t controllerId, usb_host_handle hostHandle, usb_-host_controller_handle *controllerHandle)

  Creates the USB host IP3516HS instance.
- usb_status_t USB_HostIp3516HsDestory (usb_host_controller_handle controllerHandle)

  Destroys the USB host IP3516HS instance.
- usb_status_t USB_HostIp3516HsOpenPipe (usb_host_controller_handle controllerHandle, usb_-host_pipe_handle *pipeHandle, usb_host_pipe_init_t *pipeInit)

  Opens the USB host pipe.
- usb_status_t USB_HostIp3516HsClosePipe (usb_host_controller_handle controllerHandle, usb_-host_pipe_handle pipeHandle)

  Closes the USB host pipe.
- usb_status_t USB_HostIp3516HsWritePipe (usb_host_controller_handle controllerHandle, usb_-host_pipe_handle pipeHandle, usb_host_transfer_t *transfer)

  Sends data to the pipe.
- usb_status_t USB_HostIp3516HsReadPipe (usb_host_controller_handle controllerHandle, usb_-host_pipe_handle pipeHandle, usb_host_transfer_t *transfer)

  Receives data from the pipe.
- usb_status_t USB_HostIp3516HsIoctl (usb_host_controller_handle controllerHandle, uint32_t ioctlEvent, void *ioctlParam)

  Controls the IP3516HS.

4.6.7.2 Data Structure Documentation

4.6.7.2.1 struct usb_host_ip3516hs_atl_struct_t

4.6.7.2.1.1 Field Documentation

4.6.7.2.1.1.1 volatile uint32_t usb_host_ip3516hs_atl_struct_t::J

1: enable the next PTD branching.

4.6.7.2.1.1.2 volatile uint32_t usb_host_ip3516hs_atl_struct_t::uFrame

4.6.7.2.1.1.3 volatile uint32_t usb_host_ip3516hs_atl_struct_t::S

4.6.7.2.1.1.4 volatile uint32_t usb_host_ip3516hs_atl_struct_t::RL

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

4.6.7.2.1.1.5 volatile uint32_t usb_host_ip3516hs_atl_struct_t::NrBytesToTransfer

Number of Bytes Transferred.

4.6.7.2.1.1.6   volatile uint32_t usb_host_ip3516hs_atl_struct_t::Token

4.6.7.2.2   struct usb_host_ip3516hs_ptl_struct_t

4.6.7.2.2.1   Field Documentation

4.6.7.2.2.1.1   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::NextPTDPointer

4.6.7.2.2.1.2   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::J

1: enable the next PTD branching.

4.6.7.2.2.1.3   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::uFrame

4.6.7.2.2.1.4   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::S

4.6.7.2.2.1.5   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::RL

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

4.6.7.2.2.1.6   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::NrBytesToTransfer

Number of Bytes Transferred.

4.6.7.2.2.1.7   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::I

4.6.7.2.2.1.8   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::Token

4.6.7.2.2.1.9   volatile uint32_t usb_host_ip3516hs_ptl_struct_t::uSA

4.6.7.2.3   struct usb_host_ip3516hs_sptl_struct_t

4.6.7.2.3.1   Field Documentation

4.6.7.2.3.1.1   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::uFrame

4.6.7.2.3.1.2   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::MaxPacketLength

4.6.7.2.3.1.3   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::S

4.6.7.2.3.1.4   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::RL

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

4.6.7.2.3.1.5   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::NrBytesToTransfer

Number of Bytes Transferred.

4.6.7.2.3.1.6   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::l

4.6.7.2.3.1.7   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::Token

4.6.7.2.3.1.8   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::uSA

4.6.7.2.3.1.9   volatile uint32_t usb_host_ip3516hs_sptl_struct_t::uSCS

It specifies which uSOF the complete split needs to be sent.

4.6.7.2.4   struct usb_host_ip3516hs_register_struct_t

4.6.7.2.5   struct usb_host_ip3516hs_state_struct_t

Data Fields

- volatile
  usb_host_ip3516hs_register_struct_t ∗ usbRegBase
    The base address of the register.
- void ∗ hostHandle
    Related host handle.
- volatile uint32_t bufferArrayBitMap [4]
    Bit map for USB dedicated RAM (Uint is 64bytes)
- osa_event_handle_t ip3516HsEvent
    IP3516HS event.
- uint32_t taskEventHandleBuffer [(OSA_EVENT_HANDLE_SIZE+3)/4]
    task event handle buffer
- osa_mutex_handle_t mutex
    Ip3516Hs layer mutex.
- uint8_t controllerId
    Controller id.
- uint8_t portNumber
    Port count.
- uint8_t isrNumber
    ISR Number.
- volatile uint8_t isrLevel
    The disable deep of the ISR.
- uint8_t busSuspendStatus
    Bus Suspend Status.

4.6.7.3   Enumeration Type Documentation

4.6.7.3.1   enum usb_host_ip3516hs_port_state_t

Enumerator

  kUSB_DeviceIp3516HsPortDetached   Port is detached and de-initialized.
  kUSB_DeviceIp3516HsPortPhyDetached   Port is physically detached.
  kUSB_DeviceIp3516HsPortPhyAttached   Port is physically attached.

kUSB_DeviceIp3516HsPortAttached   Port is attached and initialized.

4.6.7.4   Function Documentation

4.6.7.4.1   usb_status_t USB_HostIp3516HsCreate (uint8_t *controllerId,* usb_host_handle *hostHandle,* usb_host_controller_handle ∗ *controllerHandle*)

This function initializes the USB host IP3516HS 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, IP3516HS mutex or IP3516HS event create failed. Or, IP3516HS IP initialize failed.

4.6.7.4.2   usb_status_t USB_HostIp3516HsDestory (usb_host_controller_handle *controllerHandle*)

This function deinitializes the USB host IP3516HS controller driver.

Parameters

controller-Handle	The controller handle.

Return values

kStatus_USB_Success	The host is initialized successfully.

4.6.7.4.3   usb_status_t USB_HostIp3516HsOpenPipe (usb_host_controller_handle *controllerHandle,* usb_host_pipe_handle ∗ *pipeHandle,* usb_host_pipe_init_t ∗ *pipeInit*)

This function opens a pipe according to the pipe_init_ptr parameter.

Parameters

controller-Handle	The controller handle.
pipeHandle	The pipe handle pointer used to return the pipe handle.
pipeInit	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.

4.6.7.4.4 usb_status_t USB_HostIp3516HsClosePipe (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.

4.6.7.4.5 usb_status_t USB_HostIp3516HsWritePipe (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 IP3516HS.

4.6.7.4.6 usb_status_t USB_HostIp3516HsReadPipe (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 IP3516HS.

4.6.7.4.7 usb_status_t USB_HostIp3516HsIoctl (usb_host_controller_handle controllerHandle, uint32_t ioctlEvent, void ∗ ioctlParam)

This function controls the IP3516HS.

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 5
USB Class driver

5.1   Overview

Modules

- USB AUDIO Class driver
- USB CDC Class driver
- USB CDC-ECM Class driver
- USB HID Class driver
- USB MSC Class driver
- USB PHDC Class driver
- USB PRINTER Class driver

5.2　USB CDC Class driver

5.2.1　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.

5.2.2　USB Host CDC Initialization

When the CDC device is attached, the CDC initialization flow is as follows:
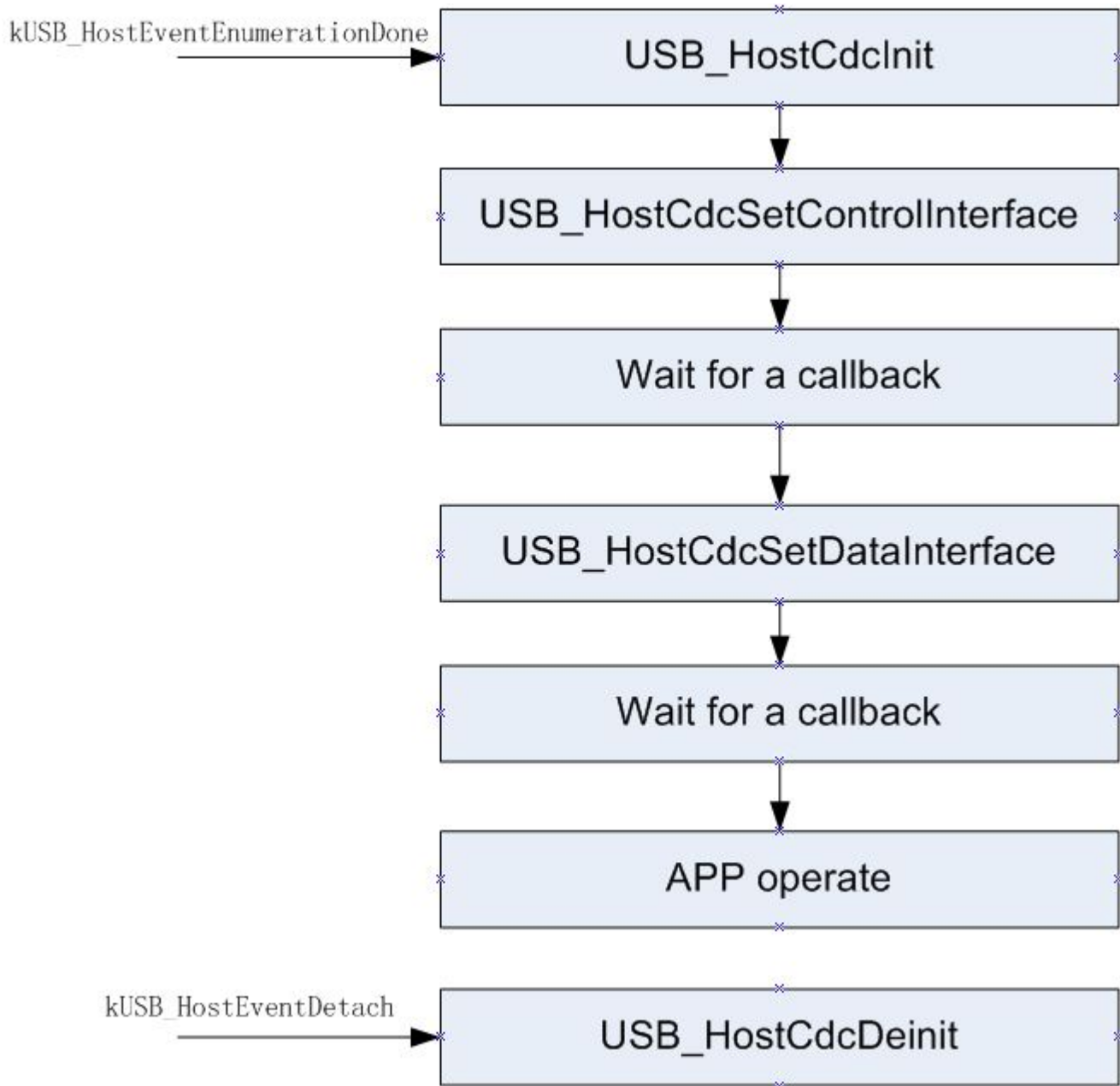
Figure 5.2.1: Host CDC Initialization

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.

5.2.3 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.

5.2.4 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.

5.2.5 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_CDC_NOTIFICATION_NETWORK_CONNECTION (0x00U)

  CDC Class-specific Management Element Notifications.
- #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_DESC_SUBTYPE_ECM 0x0FU
  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 ∗class-Handle)
  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 class-Handle)
  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_HostCdcSetAcmLineCoding (usb_host_class_handle classHandle, usb_host_-cdc_line_coding_struct_t ∗uartLineCoding, transfer_callback_t callbackFn, void ∗callbackParam)
    cdc set 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 ∗∗call-ManageDesc, 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_HostCdcGetEcmDescriptor (usb_host_class_handle classHandle, usb_host_cdc-_head_function_desc_struct_t ∗∗headDesc, usb_host_cdc_union_interface_desc_struct_t ∗∗union-InterfaceDesc, usb_host_cdc_ethernet_networking_desc_struct_t ∗∗ethernetNetworkingDesc)
    CDC gets the ECM 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.

5.2.6 Data Structure Documentation

5.2.6.1 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).

5.2.6.1.0.1 Field Documentation

5.2.6.1.0.1.1 uint8_t usb_host_cdc_line_coding_struct_t::bDataBits

5.2.6.2 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.

5.2.6.2.0.2 Field Documentation

5.2.6.2.0.2.1 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

5.2.6.3 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.

5.2.6.4 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.

5.2.6.4.0.3   Field Documentation

5.2.6.4.0.3.1   uint8_t usb_host_cdc_head_function_desc_struct_t::bFunctionLength

5.2.6.4.0.3.2   uint8_t usb_host_cdc_head_function_desc_struct_t::bDescriptorType

5.2.6.4.0.3.3   uint8_t usb_host_cdc_head_function_desc_struct_t::bDescriptorSubtype

5.2.6.4.0.3.4   uint8_t usb_host_cdc_head_function_desc_struct_t::bcdCDC[2]

5.2.6.5   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.*

5.2.6.5.0.4   Field Documentation

5.2.6.5.0.4.1   uint8_t usb_host_cdc_call_manage_desc_struct_t::bFunctionLength

5.2.6.5.0.4.2   uint8_t usb_host_cdc_call_manage_desc_struct_t::bDescriptorType

5.2.6.5.0.4.3   uint8_t usb_host_cdc_call_manage_desc_struct_t::bDescriptorSubtype

5.2.6.5.0.4.4   uint8_t usb_host_cdc_call_manage_desc_struct_t::bmCapabilities

5.2.6.5.0.4.5   uint8_t usb_host_cdc_call_manage_desc_struct_t::bDataInterface

5.2.6.6   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.*

5.2.6.6.0.5   Field Documentation

5.2.6.6.0.5.1   uint8_t usb_host_cdc_abstract_control_desc_struct_t::bFunctionLength

5.2.6.6.0.5.2   uint8_t usb_host_cdc_abstract_control_desc_struct_t::bDescriptorType

5.2.6.6.0.5.3   uint8_t usb_host_cdc_abstract_control_desc_struct_t::bDescriptorSubtype

5.2.6.6.0.5.4   uint8_t usb_host_cdc_abstract_control_desc_struct_t::bmCapabilities

5.2.6.7   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.

5.2.6.7.0.6   Field Documentation

5.2.6.7.0.6.1   uint8_t usb_host_cdc_direct_line_desc_struct_t::bFunctionLength

5.2.6.7.0.6.2   uint8_t usb_host_cdc_direct_line_desc_struct_t::bDescriptorType

5.2.6.7.0.6.3   uint8_t usb_host_cdc_direct_line_desc_struct_t::bDescriptorSubtype

5.2.6.7.0.6.4   uint8_t usb_host_cdc_direct_line_desc_struct_t::bmCapabilities

5.2.6.8   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.

5.2.6.8.0.7   Field Documentation

5.2.6.8.0.7.1   uint8_t usb_host_cdc_telephone_ringer_desc_struct_t::bFunctionLength

5.2.6.8.0.7.2   uint8_t usb_host_cdc_telephone_ringer_desc_struct_t::bDescriptorType

5.2.6.8.0.7.3   uint8_t usb_host_cdc_telephone_ringer_desc_struct_t::bRingerVolSteps

5.2.6.8.0.7.4   uint8_t usb_host_cdc_telephone_ringer_desc_struct_t::bNumRingerPatterns

5.2.6.9   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.

5.2.6.9.0.8   Field Documentation

5.2.6.9.0.8.1   uint8_t usb_host_cdc_tcLsr_desc_struct_t::bFunctionLength

5.2.6.9.0.8.2   uint8_t usb_host_cdc_tcLsr_desc_struct_t::bDescriptorType

5.2.6.9.0.8.3   uint8_t usb_host_cdc_tcLsr_desc_struct_t::bDescriptorSubtype

5.2.6.9.0.8.4   uint8_t usb_host_cdc_tcLsr_desc_struct_t::bmCapabilities[4]

5.2.6.10   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.

5.2.6.10.0.9   Field Documentation

5.2.6.10.0.9.1   uint8_t usb_host_cdc_union_interface_desc_struct_t::bFunctionLength

5.2.6.10.0.9.2   uint8_t usb_host_cdc_union_interface_desc_struct_t::bDescriptorType

5.2.6.10.0.9.3   uint8_t usb_host_cdc_union_interface_desc_struct_t::bDescriptorSubtype

5.2.6.10.0.9.4   uint8_t usb_host_cdc_union_interface_desc_struct_t::bControlInterface

5.2.6.11   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:.*

5.2.6.11.0.10   Field Documentation

5.2.6.11.0.10.1   uint8_t usb_host_cdc_tom_desc_struct_t::bFunctionLength

5.2.6.11.0.10.2   uint8_t usb_host_cdc_tom_desc_struct_t::bDescriptorType

5.2.6.11.0.10.3   uint8_t usb_host_cdc_tom_desc_struct_t::bDescriptorSubtype

5.2.6.11.0.10.4   uint8_t usb_host_cdc_tom_desc_struct_t::bmCapabilities

5.2.6.12   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.*

5.2.6.12.0.11   Field Documentation

5.2.6.12.0.11.1   uint8_t usb_host_cdc_common_desc_struct_t::bFunctionLength

5.2.6.12.0.11.2   uint8_t usb_host_cdc_common_desc_struct_t::bDescriptorType

5.2.6.12.0.11.3   uint8_t usb_host_cdc_common_desc_struct_t::bDescriptorSubtype

5.2.6.13   union usb_cdc_func_desc_struct_t

5.2.7   Function Documentation

5.2.7.1   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.

5.2.7.2 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	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.

5.2.7.3   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	*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.

5.2.7.4   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.

5.2.7.5 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_ISOC-HRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTER-RUPT. See the usb_spec.h
in	*direction*	Pipe direction.

Return values

0	The classHandle is NULL.
max	Packet size.

5.2.7.6 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.

5.2.7.7 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.

5.2.7.8 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.

5.2.7.9 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.

5.2.7.10 usb_status_t USB_HostCdcSetAcmLineCoding (usb_host_class_handle *classHandle,* usb_host_cdc_line_coding_struct_t ∗ *uartLineCoding,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

This function implements cdc SetLineCoding request.refer to pstn 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.

5.2.7.11 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.

5.2.7.12 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.

5.2.7.13 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.

5.2.7.14 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.

5.2.7.15 usb_status_t USB_HostCdcGetEcmDescriptor (usb_host_class_handle *classHandle,* usb_host_cdc_head_function_desc_struct_t ∗∗ *headDesc,* usb_host_cdc_union_interface_desc_struct_t ∗∗ *unionInterfaceDesc,* usb_host_cdc_ethernet_networking_desc_struct_t ∗∗ *ethernetNetworkingDesc*)

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

Parameters

classHandle	The class handle.
headDesc	The head function descriptor pointer.
unionInterface-Desc	The union functional descriptor pointer.
ethernet-Networking-Desc	The ethernet networking functional descriptor pointer.

Returns

An error code or kStatus_USB_Success.

5.2.7.16 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.

5.3   USB CDC-ECM Class driver

5.3.1   Overview

The USB CDC-ECM defines mechanisms for a device and host to forward Ethernet frames, and it is used for Ethernet-related functional devices. It is an subclass of USB Communication Device Class, and the CDC-ECM driver handles the specific control requests for CDC-ECM class and transfers data to and from the device through the bulk & interrupt pipes. This section describes the programming interface of the USB HOST CDC-ECM class driver.

5.3.2   USB Host CDC-ECM Initialization

USB Host CDC-ECM Initialization is the same as USB Host CDC Initialization. When the CDC-ECM device is attached, the CDC-ECM initialization flow is as follows:
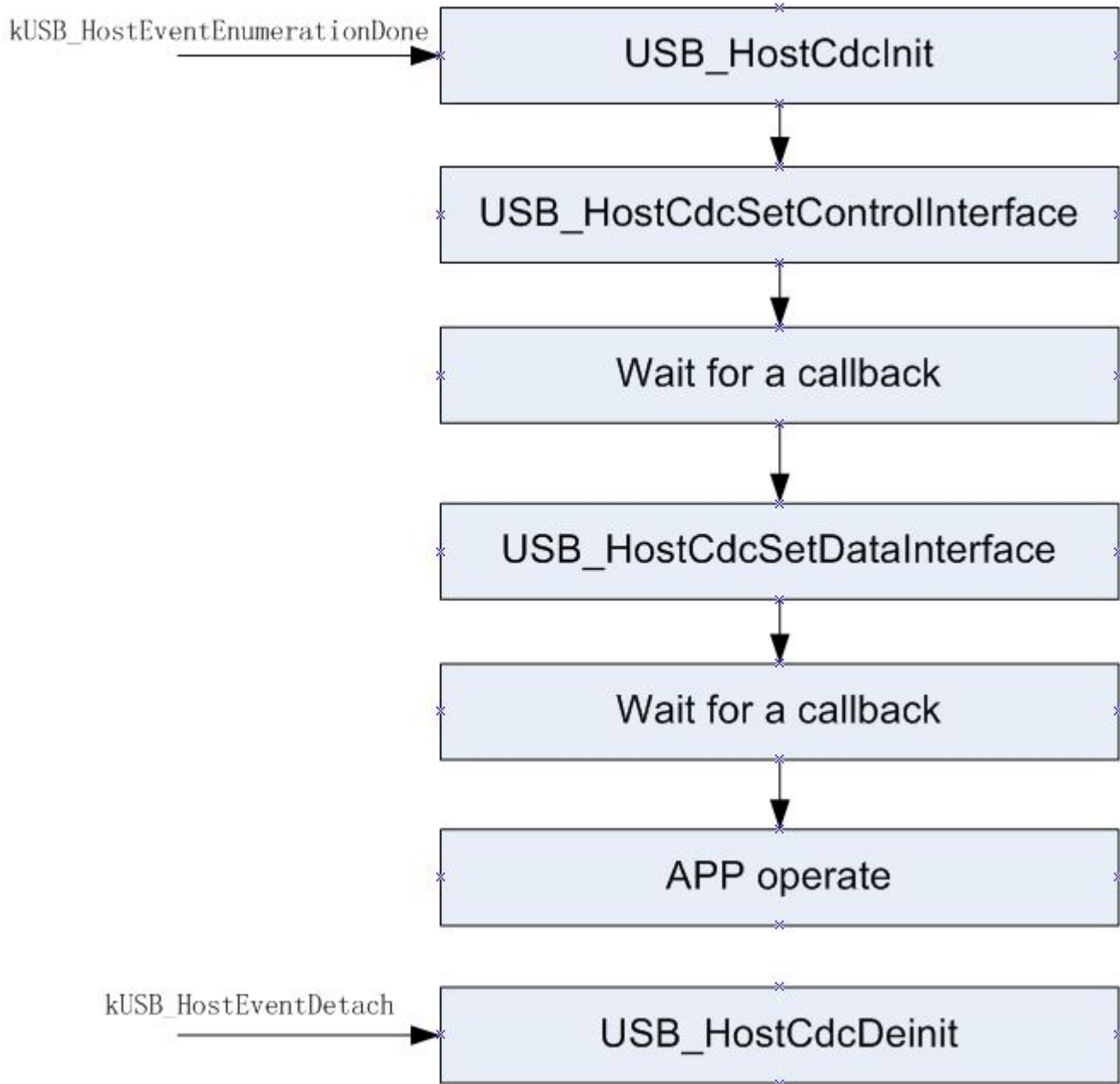
Figure 5.3.1: Host CDC Initialization

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.

5.3.3 USB Host CDC De-initialization

USB Host CDC-ECM Initialization is the same as USB Host CDC Initialization. An application can call the USB_HostCdcDeinit to deinitialize the CDC. This function cancels the transfer, closes the pipe, and frees the CDC 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.

5.3.4 USB Host CDC-ECM Send data

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

5.3.5 USB Host CDC-ECM Receive data

Provides the buffer pointer, the buffer length, the callback function, and the callback parameter and calls USB_HostCdcEcmDataRecv 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_HostCdcEcmPowerManagementPatternFilter_t
    Power Management Pattern Filter Structure. More...
- struct USB_HostCdcEcmSetEthernetMulticastFiltersParam_t
    Parameters of Request SetEthernetMulticastFilters. More...
- struct USB_HostCdcEcmSetEthernetPowerManagementPatternFilterParam_t
    Parameters of Request SetEthernetPowerManagementPatternFilter. More...
- struct USB_HostCdcEcmGetEthernetPowerManagementPatternFilterParam_t

Parameters of Request GetEthernetPowerManagementPatternFilter. More...
- struct USB_HostCdcEcmGetEthernetStatisticParam_t

    Parameters of Request GetEthernetStatistic. More...
- union USB_HostCdcEcmRequestParam_t

    Parameters of CDC-ECM Class-Specific Request. More...

Macros

- #define USB_HOST_CDC_ECM_CLASS_CODE (0x02U)

    Class-Specific Codes.
- #define CDC_ECM_SET_ETHERNET_MULTICAST_FILTER (0x40U)

    Class-Specific Request Codes for Ethernet subclass.
- #define CDC_ECM_ETH_PACKET_FILTER_PROMISCUOUS (0x01U)

    Ethernet Packet Filter Bitmap.
- #define CDC_ECM_ETH_STATS_BITMAP(type) ((uint32_t)(1 << type))

    Ethernet Statistics Capabilities Bitmap.
- #define CDC_ECM_ETH_STATS_FEATURE_SEL_CODE(type) ((uint8_t)(type + 1))

    Ethernet Statistics Feature Selector Codes.
- #define CDC_ECM_FRAME_MAX_FRAMELEN (1514U)

    Default maximum Ethernet frame size.

Typedefs

- typedef uint16_t USB_HostCdcEcmSetEthernetPacketFilterParam_t

    Parameters of Request SetEthernetPacketFilter.

Enumerations

- enum USB_HostCdcEcmEthStatsCap_t {
  XMIT_OK = 0,
  RVC_OK,
  XMIT_ERROR,
  RCV_ERROR,
  RCV_NO_BUFFER,
  DIRECTED_BYTES_XMIT,
  DIRECTED_FRAMES_XMIT,
  MULTICAST_BYTES_XMIT,
  MULTICAST_FRAMES_XMIT,
  BROADCAST_BYTES_XMIT,
  BROADCAST_FRAMES_XMIT,
  DIRECTED_BYTES_RCV,
  DIRECTED_FRAMES_RCV,
  MULTICAST_BYTES_RCV,
  MULTICAST_FRAMES_RCV,
  BROADCAST_BYTES_RCV,
  BROADCAST_FRAMES_RCV,
  RCV_CRC_ERROR,
  TRANSMIT_QUEUE_LENGTH,
  RCV_ERROR_ALIGNMENT,
  XMIT_ONE_COLLISION,
  XMIT_MORE_COLLISIONS,
  XMIT_DEFERRED,
  XMIT_MAX_COLLISIONS,
  RCV_OVERRUN,
  XMIT_UNDERRUN,
  XMIT_HEARTBEAT_FAILURE,
  XMIT_TIMES_CRS_LOST,
  XMIT_LATE_COLLISIONS,
  ETH_STATS_CAP_COUNT }

  Ethernet Statistics Capabilities.

USB CDC-ECM host class driver

- usb_status_t USB_HostCdcEcmSetEthernetMulticastFilters (usb_host_class_handle classHandle, uint16_t filterNumSum, uint8_t(∗filterData)[6], transfer_callback_t callbackFn, void ∗callbackParam)

  Set the Ethernet device multicast filters as specified in the sequential list of 48 bit Ethernet multicast addresses.
- usb_status_t USB_HostCdcEcmSetEthernetPowerManagementPatternFilter (usb_host_class_handle classHandle, uint16_t filterNum, uint8_t ∗buffer, uint16_t bufferLength, transfer_callback_t

callbackFn, void ∗callbackParam)

    Set up the specified Ethernet power management pattern filter as described in the data structure.

- usb_status_t   USB_HostCdcEcmGetEthernetPowerManagementPatternFilter (usb_host_class_-handle classHandle, uint16_t filterNum, uint8_t(∗buffer)[2], transfer_callback_t callbackFn, void ∗callbackParam)

    Retrieves the status of the specified Ethernet power management pattern filter from the device.

- usb_status_t   USB_HostCdcEcmSetEthernetPacketFilter (usb_host_class_handle classHandle, uint16_t packetFilterBitmap, transfer_callback_t callbackFn, void ∗callbackParam)

    Configure device Ethernet packet filter settings.

- usb_status_t USB_HostCdcEcmGetEthernetStatistic (usb_host_class_handle classHandle, uint16_t featureSelector, uint8_t(∗buffer)[4], transfer_callback_t callbackFn, void ∗callbackParam)

    Retrieve a statistic based on the feature selector.

- usb_status_t USB_HostCdcEcmDataRecv (usb_host_class_handle classHandle, uint8_t ∗buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void ∗callbackParam)

    Receive data.

- usb_status_t USB_HostCdcEcmDataSend (usb_host_class_handle classHandle, uint8_t ∗buffer, uint32_t bufferLength, uint16_t maxSegmentSize, transfer_callback_t callbackFn, void ∗callback-Param)

    Send data.

- usb_status_t   USB_HostCdcEcmGetMacStringDescriptor (usb_host_class_handle classHandle, uint16_t index, uint16_t langID, uint8_t ∗buffer, uint16_t bufferLength, transfer_callback_t callbackFn, void ∗callbackParam)

    Get CDC-ECM MAC string descriptor.

- uint32_t   USB_HostCdcEcmPowerManagementPatternFilterData (USB_HostCdcEcmPower-ManagementPatternFilter_t ∗param, uint8_t ∗dataBuf)

    Encapsulating a Power Management Pattern Filter.

5.3.6 Data Structure Documentation

5.3.6.1 struct USB_HostCdcEcmPowerManagementPatternFilter_t

5.3.6.2 struct USB_HostCdcEcmSetEthernetMulticastFiltersParam_t

5.3.6.3 struct USB_HostCdcEcmSetEthernetPowerManagementPatternFilterParam_t

5.3.6.4 struct USB_HostCdcEcmGetEthernetPowerManagementPatternFilterParam_t

5.3.6.5 struct USB_HostCdcEcmGetEthernetStatisticParam_t

5.3.6.6 union USB_HostCdcEcmRequestParam_t

5.3.7 Enumeration Type Documentation

5.3.7.1 enum USB_HostCdcEcmEthStatsCap_t

Enumerator

XMIT_OK Frames transmitted without errors.
RVC_OK Frames received without errors.
XMIT_ERROR Frames not transmitted, or transmitted with errors.
RCV_ERROR Frames received with errors that are not delivered to the USB host.
RCV_NO_BUFFER Frame missed, no buffers.
DIRECTED_BYTES_XMIT Directed bytes transmitted without errors.
DIRECTED_FRAMES_XMIT Directed frames transmitted without errors.
MULTICAST_BYTES_XMIT Multicast bytes transmitted without errors.
MULTICAST_FRAMES_XMIT Multicast frames transmitted without errors.
BROADCAST_BYTES_XMIT Broadcast bytes transmitted without errors.
BROADCAST_FRAMES_XMIT Broadcast frames transmitted without errors.
DIRECTED_BYTES_RCV Directed bytes received without errors.
DIRECTED_FRAMES_RCV Directed frames received without errors.
MULTICAST_BYTES_RCV Multicast bytes received without errors.
MULTICAST_FRAMES_RCV Multicast frames received without errors.
BROADCAST_BYTES_RCV Broadcast bytes received without errors.
BROADCAST_FRAMES_RCV Broadcast frames received without errors.
RCV_CRC_ERROR Frames received with circular redundancy check (CRC) or frame check sequence (FCS) error.
TRANSMIT_QUEUE_LENGTH Length of transmit queue.
RCV_ERROR_ALIGNMENT Frames received with alignment error.
XMIT_ONE_COLLISION Frames transmitted with one collision.
XMIT_MORE_COLLISIONS Frames transmitted with more than one collision.
XMIT_DEFERRED Frames transmitted after deferral.
XMIT_MAX_COLLISIONS Frames not transmitted due to collisions.

RCV_OVERRUN   Frames not received due to overrun.

XMIT_UNDERRUN   Frames not transmitted due to underrun.

XMIT_HEARTBEAT_FAILURE   Frames transmitted with heartbeat failure.

XMIT_TIMES_CRS_LOST   Times carrier sense signal lost during transmission.

XMIT_LATE_COLLISIONS   Late collisions detected.

ETH_STATS_CAP_COUNT   The count of Ethernet Statistics Capabilities.

5.3.8   Function Documentation

5.3.8.1   usb_status_t USB_HostCdcEcmSetEthernetMulticastFilters (usb_host_-class_handle *classHandle,* uint16_t *filterNumSum,* uint8_t(∗) *filterData[6],* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
filterNumSum	The sum of filters to be set.
filterData	The data of filters.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.2   usb_status_t USB_HostCdcEcmSetEthernetPowerManagementPatternFilter (usb_host_class_handle *classHandle,* uint16_t *filterNum,* uint8_t ∗ *buffer,* uint16_t *bufferLength,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
filterNum	The number of the specific pattern filter to be set.
buffer	The data of pattern filters.

bufferLength	The length of pattern filters.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.3 usb_status_t USB_HostCdcEcmGetEthernetPowerManagementPatternFilter (usb_host_class_handle *classHandle,* uint16_t *filterNum,* uint8_t(∗) *buffer[2],* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
filterNum	The number of the specific pattern filter to be set.
buffer	The pattern filter active.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.4 usb_status_t USB_HostCdcEcmSetEthernetPacketFilter (usb_host_class_handle *classHandle,* uint16_t *packetFilterBitmap,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
packetFilter-Bitmap	The config of the packet filter to be set.

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

Returns

An error code or kStatus_USB_Success.

5.3.8.5 usb_status_t USB_HostCdcEcmGetEthernetStatistic (usb_host_class_handle *classHandle,* uint16_t *featureSelector,* uint8_t(∗) *buffer[4],* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
featureSelector	The ethernet statistics feature selector to be set.
buffer	The receiving address for Ethernet statistics.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.6 usb_status_t USB_HostCdcEcmDataRecv (usb_host_class_handle *classHandle,* uint8_t ∗ *buffer,* uint32_t *bufferLength,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
buffer	Received data.
bufferLength	The length of data.

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

Returns

An error code or kStatus_USB_Success.

5.3.8.7 usb_status_t USB_HostCdcEcmDataSend (usb_host_class_handle *classHandle,* uint8_t ∗ *buffer,* uint32_t *bufferLength,* uint16_t *maxSegmentSize,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
buffer	Data to be sent.
bufferLength	The length of data.
maxSegment-Size	The max segment size of the data to be sent.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.8 usb_status_t USB_HostCdcEcmGetMacStringDescriptor (usb_host_class_handle *classHandle,* uint16_t *index,* uint16_t *langID,* uint8_t ∗ *buffer,* uint16_t *bufferLength,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Parameters

classHandle	The class handle.
index	The index of string descriptor where MAC is in it.

langID	Language ID of string descriptor.
buffer	The string to be received.
bufferLength	The length of string.
callbackFn	This callback is called after this function completes.
callbackParam	The parameter in the callback function.

Returns

An error code or kStatus_USB_Success.

5.3.8.9 uint32_t USB_HostCdcEcmPowerManagementPatternFilterData (USB_HostCdcEcmPowerManagementPatternFilter_t ∗ *param,* uint8_t ∗ *dataBuf*)

Parameters

param	Information of Power Management Pattern Filter.
dataBuf	The buffer to store Encapsulated Power Management Pattern Filter.

Returns

The length of Power Management Pattern Filter.

5.4 USB HID Class driver

5.4.1 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.

5.4.2 USB Host HID Initialization

When the HID device is attached, the HID initialization flow is as follows:
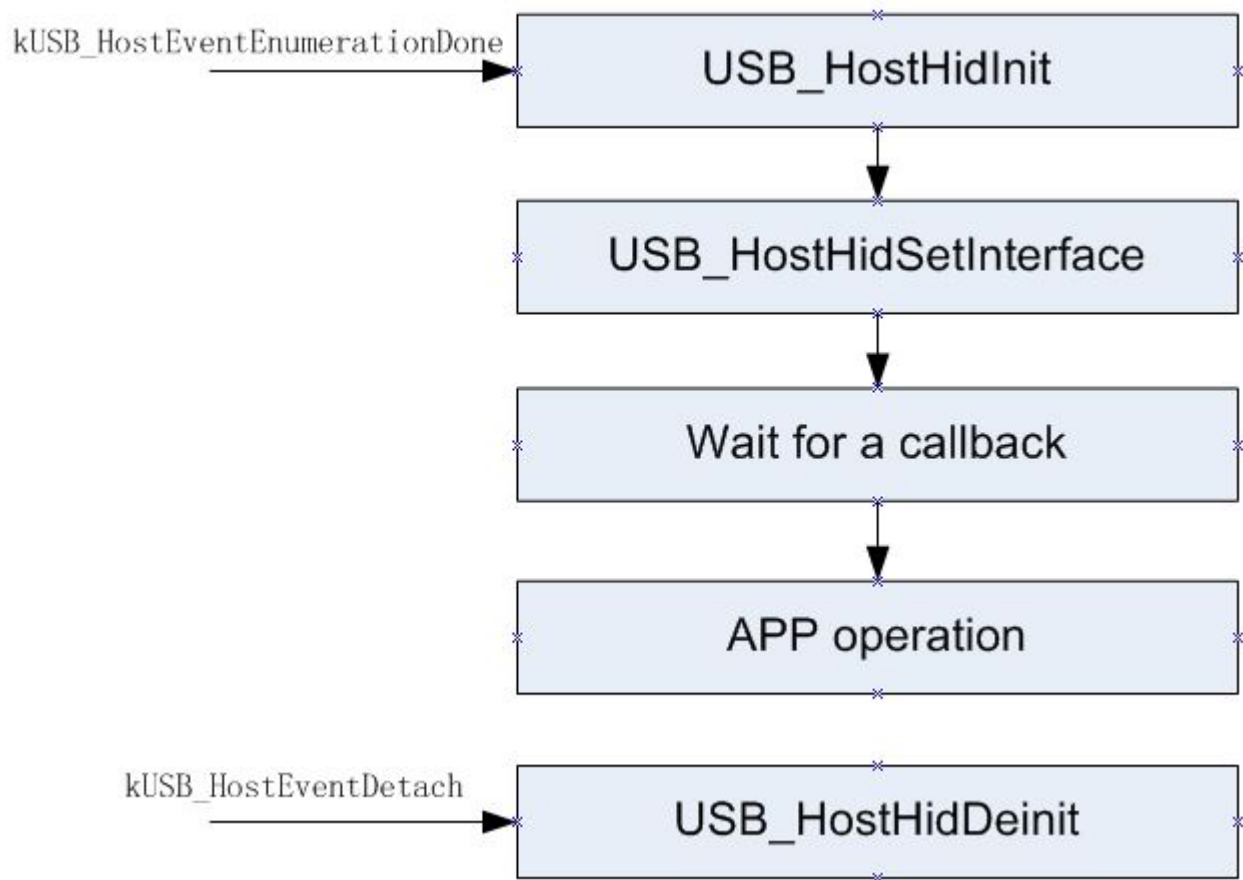
Figure 5.4.1: Host HID Initialization

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.

5.4.3 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.

5.4.4 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.

5.4.5 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 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.

5.4.6 Data Structure Documentation

5.4.6.1 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.

5.4.6.2 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.

5.4.6.3 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.

5.4.7 Function Documentation

5.4.7.1 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.

5.4.7.2 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.

5.4.7.3   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.

5.4.7.4   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_ISOC-HRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTER-RUPT. See the usb_spec.h
in	direction	Pipe direction.

Return values

0	The classHandle is NULL.
Maximum	packet size.

5.4.7.5   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.

5.4.7.6 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.

5.4.7.7 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.

5.4.7.8 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.

5.4.7.9 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.

5.4.7.10 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.

5.4.7.11 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.

5.4.7.12   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.

5.4.7.13   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.

5.5 USB MSC Class driver

5.5.1 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.

5.5.2 USB Host MSC Initialization

When the MSD device is attached, the MSD initialization flow is as follows:
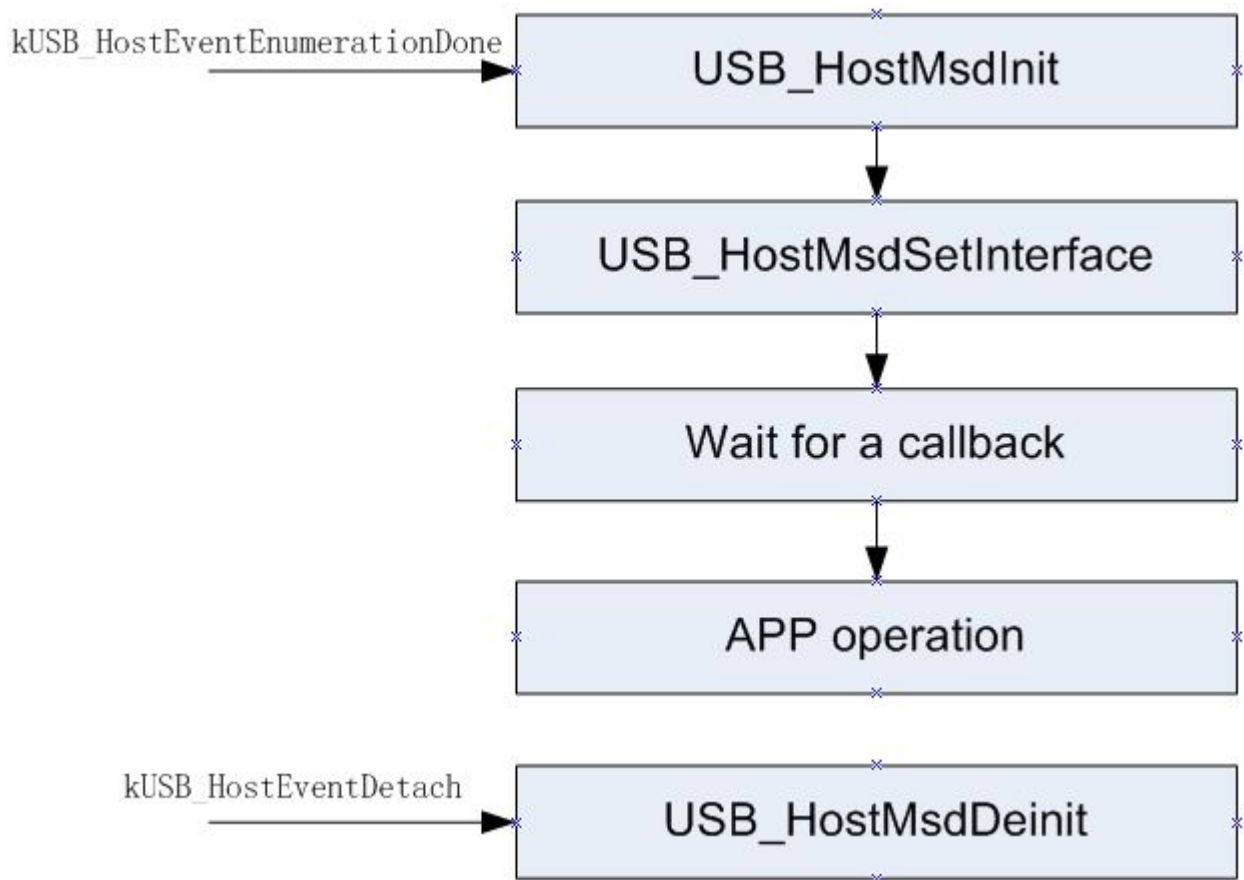
Figure 5.5.1: Host MSD Initialization

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.

5.5.3 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.

5.5.4 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 ∗callback-Param)

  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 logical-Unit, uint8_t selfTest, transfer_callback_t callbackFn, void ∗callbackParam)

  Mass storage send diagnostic.
- usb_status_t USB_HostMsdCommand (usb_host_class_handle classHandle, uint8_t ∗buffer, uint32_t bufferLength, transfer_callback_t callbackFn, void ∗callbackParam, uint8_t direction, uint8_t byteValues[10])

  all ufi function calls this api.

5.5.5   Data Structure Documentation

5.5.5.1   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.

5.5.5.1.0.12   Field Documentation

5.5.5.1.0.12.1   uint32_t usb_host_cbw_t::CBWSignature

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

5.5.5.1.0.12.2 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

5.5.5.1.0.12.3 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.

5.5.5.1.0.12.4 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).

5.5.5.2 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.

5.5.5.2.0.13 Field Documentation

5.5.5.2.0.13.1 uint32_t usb_host_csw_t::CSWSignature

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

5.5.5.2.0.13.2 uint32_t usb_host_csw_t::CSWDataResidue

5.5.5.2.0.13.3 uint8_t usb_host_csw_t::CSWStatus

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

5.5.5.3 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.

5.5.5.4 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_HostMsdMass-StorageReset to reset or there is no reset

5.5.5.5 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.

5.5.5.6 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.

5.5.5.7 struct usb_host_ufi_read_capacity_t

Data Fields

- uint8_t lastLogicalBlockAddress [4]
    - *The logical block number.*
- uint8_t blockLengthInBytes [4]
    - *Block size.*

5.5.6 Function Documentation

5.5.6.1 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.

5.5.6.2 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.

5.5.6.3 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.

5.5.6.4 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.

5.5.6.5 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.

5.5.6.6 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.

5.5.6.7 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.

5.5.6.8 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.

5.5.6.9 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.

5.5.6.10 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.

5.5.6.11 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.

5.5.6.12 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.

5.5.6.13 **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.

5.5.6.14 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.

5.5.6.15 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.

5.5.6.16 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.

5.5.6.17 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.

5.5.6.18 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.

5.5.6.19   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.

5.5.6.20 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.

5.5.6.21 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.

5.5.6.22 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.

5.5.6.23 **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.

5.5.6.24 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.

5.5.6.25 usb_status_t USB_HostMsdCommand (usb_host_class_handle *classHandle,* uint8_t ∗ *buffer,* uint32_t *bufferLength,* transfer_callback_t *callbackFn,* void ∗ *callbackParam,* uint8_t *direction,* uint8_t *byteValues[10]*)

This function implements the common ufi commands.

Parameters

classHandle	the class msd handle.
buffer	buffer pointer.
bufferLength	buffer length.
callbackFn	callback function.
callbackParam	callback parameter.
direction	command direction.
byteValues	ufi command fields value.

Returns

An error code or kStatus_USB_Success.

5.6   USB AUDIO Class driver

5.6.1   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. The MCUXpresso SDK USB stack provides support for USB Audio Class 1.0 and USB Audio Class 2.0.

5.6.2   USB Host audio Initialization

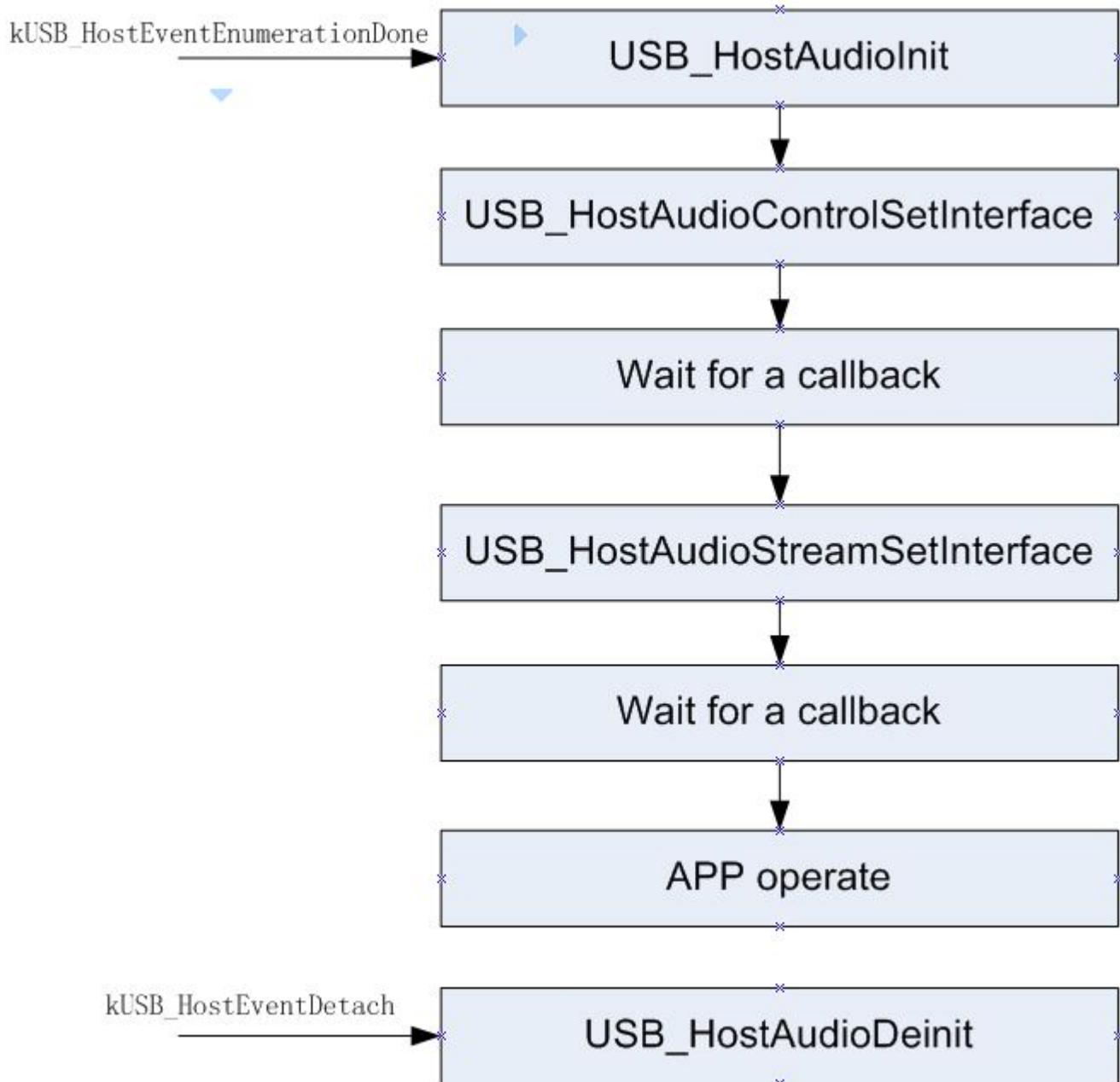When audio device is attached, audio initialization occurs as follows:

Figure 5.6.1: 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.

5.6.3 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.

5.6.4 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.

5.6.5 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_2_0_ctrl_header_desc_t

  Audio control interface header descriptor structure. More...
- struct usb_audio_2_0_ctrl_clock_source_desc_t

  Audio control interface clock source descriptor structure. More...
- struct usb_audio_2_0_ctrl_it_desc_t

  Audio control interface input terminal descriptor structure. More...
- struct usb_audio_2_0_ctrl_ot_desc_t

  Audio control interface output terminal descriptor structure. More...
- struct usb_audio_2_0_ctrl_fu_desc_t

Audio control interface feature unit descriptor structure. [More...](#)
- struct usb_audio_2_0_ctrl_mu_desc_t

  Audio control interface feature unit descriptor structure. [More...](#)
- struct usb_audio_2_0_stream_specific_iso_endp_desc_t

  Audio as isochronous audio data endpoint descriptor structure. [More...](#)
- 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_2_0_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 usb_audio_2_0_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.
- #define USB_AUDIO_DESCRIPTOR_TYPE_CS_DEVICE (0x21U)
  Audio device class-specific descriptor type.

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_descriptor_endpoint_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.

- usb_status_t USB_HostAudioControlGetCurrentAltsettingSpecificDescriptors (usb_host_class_-handle classHandle, uint32_t DescriptorType, uint32_t DescriptorSubType, void **Descriptor)

   get audio control current altsetting descriptor.

- usb_status_t USB_HostAudioStreamGetCurrentAltsettingSpecificDescriptors (usb_host_class_-handle classHandle, uint32_t DescriptorType, uint32_t DescriptorSubType, void **Descriptor)

   get audio control current altsetting descriptor.

- usb_status_t USB_HostAudioGetSetFeatureUnitRequest (usb_host_class_handle classHandle, uint32_t csAndCn, uint32_t cmdCode, void *buf, uint32_t bufLen, transfer_callback_t callbackFn, void *callbackParam)

   usb audio set/get feature unit request.

- usb_status_t USB_HostAudioGetSetClockSourceRequest (usb_host_class_handle classHandle, uint32_t csAndCn, uint32_t cmdCode, void *buf, uint32_t bufLen, transfer_callback_t callbackFn, void *callbackParam)

   usb audio set/get feature unit request.

- usb_status_t USB_HostAudioGetSetEndpointRequest (usb_host_class_handle classHandle, uint32-_t csAndCn, uint32_t cmdCode, void *buf, uint32_t bufLen, transfer_callback_t callbackFn, void *callbackParam)

   usb audio set/get endp unit request.

- usb_status_t USB_HostAudioSetStreamOutDataInterval (usb_host_class_handle classHandle, uint8_t intervalValue)

   change the ISO out data interval

5.6.6   Data Structure Documentation

5.6.6.1   struct usb_audio_2_0_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 bCategory
    Constant, indicating the primary use of this audio function, as intended by the manufacturer.
- uint8_t wtotallength [2]
    Total number of bytes returned for the class-specific AudioControl interface descriptor.
- uint8_t bmControls
    D1..0: Latency Control D7..2: Reserved.

5.6.6.1.0.14   Field Documentation

5.6.6.1.0.14.1   uint8_t usb_audio_2_0_ctrl_header_desc_t::bCategory

See Appendix A.7, Audio Function Category Codes.

5.6.6.1.0.14.2   uint8_t usb_audio_2_0_ctrl_header_desc_t::wtotallength[2]

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

5.6.6.1.0.14.3   uint8_t usb_audio_2_0_ctrl_header_desc_t::bmControls

Must be set to 0.

5.6.6.2   struct usb_audio_2_0_ctrl_clock_source_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 bClockID
    Constant uniquely identifying the Clock Source Entity within the audio function.
- uint8_t bmAttributes
    D1..0: Clock Type, D2: Clock synchronized to SOF.
- uint8_t bmControls
    D1..0: Clock Frequency D3..2: Clock Validity Control.
- uint8_t bAssocTerminal
    Terminal ID of the Terminal that is associated with this Clock Source.
- uint8_t iClockSource
    Index of a string descriptor, describing the Clock Source Entity.

5.6.6.2.0.15   Field Documentation

5.6.6.2.0.15.1   uint8_t usb_audio_2_0_ctrl_clock_source_desc_t::bClockID

This value is used in all requests to address this Entity.

5.6.6.2.0.15.2   uint8_t usb_audio_2_0_ctrl_clock_source_desc_t::bmControls

5.6.6.2.0.15.3   uint8_t usb_audio_2_0_ctrl_clock_source_desc_t::bAssocTerminal

5.6.6.2.0.15.4   uint8_t usb_audio_2_0_ctrl_clock_source_desc_t::iClockSource

5.6.6.3   struct usb_audio_2_0_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 bCSourceID
  - *ID of the Clock Entity to which this Input Terminal is connected.*
- uint8_t bNrChannels
  - *Describes the spatial location of the logical channels.*
- uint8_t bmChannelConfig [4]
  - *Describes the spatial location of the logical channels.*
- uint8_t iChannelNames
  - *Index of a string descriptor, describing the Input Terminal.*
- uint8_t bmControls [2]
  - *D1..0: Copy Protect Control D3..2: Connector Control D5..4: Overload Control D7..6: Cluster Control D9..8: Underflow Control D11..10: Overflow Control D15..12: Reserved.*
- uint8_t iTerminal
  - *Index of a string descriptor, describing the Input Terminal.*

5.6.6.3.0.16   Field Documentation

5.6.6.3.0.16.1   uint8_t usb_audio_2_0_ctrl_it_desc_t::bterminalid

This value is used in all requests to address this Terminal

5.6.6.3.0.16.2   uint8_t usb_audio_2_0_ctrl_it_desc_t::bCSourceID

5.6.6.3.0.16.3   uint8_t usb_audio_2_0_ctrl_it_desc_t::bNrChannels

5.6.6.3.0.16.4   uint8_t usb_audio_2_0_ctrl_it_desc_t::bmChannelConfig[4]

5.6.6.3.0.16.5   uint8_t usb_audio_2_0_ctrl_it_desc_t::bmControls[2]

Must be set to 0.

5.6.6.3.0.16.6   uint8_t usb_audio_2_0_ctrl_it_desc_t::iTerminal

5.6.6.4   struct usb_audio_2_0_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
  IConstant, 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 bCSourceID
  ID of the Clock Entity to which this Output Terminal is connected.
- uint8_t bmControls [2]
  D1..0: Copy Protect Control D3..2: Connector Control D5..4: Overload Control D7..6: Underflow Control D9..8: Overflow Control D15..10: Reserved.
- uint8_t iTerminal
  Index of a string descriptor, describing the Input Terminal.

5.6.6.4.0.17   Field Documentation

5.6.6.4.0.17.1   uint8_t usb_audio_2_0_ctrl_ot_desc_t::bterminalid

This value is used in all requests to address this Terminal

5.6.6.4.0.17.2 uint8_t usb_audio_2_0_ctrl_ot_desc_t::bassocterminal

5.6.6.4.0.17.3 uint8_t usb_audio_2_0_ctrl_ot_desc_t::bSourceID

5.6.6.4.0.17.4 uint8_t usb_audio_2_0_ctrl_ot_desc_t::bCSourceID

5.6.6.4.0.17.5 uint8_t usb_audio_2_0_ctrl_ot_desc_t::bmControls[2]

Must be set to 0.

5.6.6.4.0.17.6 uint8_t usb_audio_2_0_ctrl_ot_desc_t::iTerminal

5.6.6.5 struct usb_audio_2_0_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 bmaControls0 [4]
  The Controls bitmap for master channel 0:

5.6.6.5.0.18 Field Documentation

5.6.6.5.0.18.1 uint8_t usb_audio_2_0_ctrl_fu_desc_t::bunitid

This value is used in all requests to address this unit

5.6.6.6 struct usb_audio_2_0_ctrl_mu_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 bNrInPins
  Number of Input Pins of this Unit: p.

5.6.6.6.0.19 Field Documentation

5.6.6.6.0.19.1 uint8_t usb_audio_2_0_ctrl_mu_desc_t::bunitid

This value is used in all requests to address this unit

5.6.6.7 struct usb_audio_2_0_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 bmControls
  D1..0: Pitch Control D3..2: Data Overrun Control D5..4: Data Underrun Control D7..6: Reserved.
- 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.

5.6.6.7.0.20 Field Documentation

5.6.6.7.0.20.1 uint8_t usb_audio_2_0_stream_specific_iso_endp_desc_t::bmControls

Must be set to 0.

5.6.6.7.0.20.2 uint8_t usb_audio_2_0_stream_specific_iso_endp_desc_t::wlockdelay[2]

Units used depend on the value of the bLockDelayUnits field.

5.6.6.8 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.

5.6.6.8.0.21   Field Documentation

5.6.6.8.0.21.1   uint8_t usb_audio_ctrl_header_desc_t::wtotallength[2]

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

5.6.6.9   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.

5.6.6.9.0.22   Field Documentation

5.6.6.9.0.22.1   uint8_t usb_audio_ctrl_it_desc_t::bterminalid

This value is used in all requests to address this Terminal

5.6.6.9.0.22.2   uint8_t usb_audio_ctrl_it_desc_t::wchannelconfig[2]

5.6.6.10   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.*

5.6.6.10.0.23   Field Documentation

5.6.6.10.0.23.1   uint8_t usb_audio_ctrl_ot_desc_t::bterminalid

This value is used in all requests to address this Terminal

5.6.6.11   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.*

5.6.6.11.0.24   Field Documentation

5.6.6.11.0.24.1   uint8_t usb_audio_ctrl_fu_desc_t::bunitid

This value is used in all requests to address this unit

5.6.6.12   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.

5.6.6.12.0.25   Field Documentation

5.6.6.12.0.25.1   uint8_t usb_audio_stream_specific_iso_endp_desc_t::wlockdelay[2]

Units used depend on the value of the bLockDelayUnits field.

5.6.6.13   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.

5.6.6.14   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.

5.6.6.15 struct usb_audio_2_0_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 bmControls

  D1..0: Active Alternate Setting Control D3..2: Valid Alternate Settings Control D7..4: Reserved.
- uint8_t bFormatType

  Constant identifying the Format Type the AudioStreaming interface is using.
- uint8_t bmFormats [4]

  The Audio Data Format(s) that can be used to communicate with this interface.
- uint8_t bNrChannels

  Number of physical channels in the AS Interface audio channel cluster.
- uint8_t bmChannelConfig [4]

  Describes the spatial location of the physical channels.
- uint8_t biChannelNames

  Index of a string descriptor, describing the name of the first physical channel.

5.6.6.15.0.26 Field Documentation

5.6.6.15.0.26.1 uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::bmControls

Must be set to 0.

5.6.6.15.0.26.2 uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::bFormatType

5.6.6.15.0.26.3 uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::bmFormats[4]

See the USB Audio Data Formats document for further details

5.6.6.15.0.26.4   uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::bNrChannels

5.6.6.15.0.26.5   uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::bmChannelConfig[4]

5.6.6.15.0.26.6   uint8_t usb_audio_2_0_stream_spepific_as_intf_desc_t::biChannelNames

5.6.6.16   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.

5.6.6.17   struct usb_audio_2_0_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 bSubslotSize
  The number of bytes occupied by one audio subslot.
- uint8_t bBitResolution
  The number of effectively used bits from the available bits in an audio subslot.

5.6.6.17.0.27   Field Documentation

5.6.6.17.0.27.1   uint8_t usb_audio_2_0_stream_format_type_desc_t::bSubslotSize

Can be 1, 2, 3 or 4.

5.6.6.18   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.*
- void ∗ asIntfDesc
    - *Audio class class-specific as interface descriptor pointer.*
- void ∗ formatTypeDesc
    - *Audio class class-specific format type descriptor pointer.*
- usb_descriptor_endpoint_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.*
- void ∗ headerDesc
    - *Audio class header descriptor pointer.*
- void ∗ itDesc
    - *Audio class input terminal descriptor pointer.*
- void ∗ otDesc
    - *Audio class output terminal descriptor pointer.*
- void ∗ fuDesc
    - *Audio class feature unit descriptor pointer.*
- void ∗ clockSource
    - *Audio class clock source 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.
- uint16_t deviceAudioVersion

    device's current Audio version, 16bit to aligned with Spec
- 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.

5.6.7 Function Documentation

5.6.7.1 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.

5.6.7.2 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.

5.6.7.3 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.

5.6.7.4 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.

5.6.7.5 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.

5.6.7.6 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.

5.6.7.7 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.

5.6.7.8 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_descriptor_endpoint_t ∗∗ *isoEndpDesc*)

Deprecated Do not use this function. It has been superceded by USB_HostAudioStreamGetCurrent-AltsettingSpecificDescriptors.

This function implements the get audio stream current altsetting descriptor.

Parameters

classHandle	The class handle.
asIntfDesc	The pointer of class-specific AS interface descriptor.
formatType-Desc	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_Invalid-Handle	The classHandle is NULL pointer.

5.6.7.9 usb_status_t USB_HostAudioFeatureUnitRequest (usb_host_class_handle *classHandle,* uint8_t *channelNo,* void ∗ *buf,* uint32_t *cmdCode,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Deprecated Do not use this function. It has been superceded by USB_HostAudioGetSetFeatureUnit-Request.

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.

5.6.7.10 usb_status_t USB_HostAudioEndpointRequest (usb_host_class_handle *classHandle,* void ∗ *buf,* uint32_t *cmdCode,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

Deprecated Do not use this function. It has been superceded by USB_HostAudioGetSetEndpoint-Request.

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.

5.6.7.11 usb_status_t USB_HostAudioControlGetCurrentAltsettingSpecificDescriptors (usb_host_class_handle *classHandle,* uint32_t *DescriptorType,* uint32_t *DescriptorSubType,* void ∗∗ *Descriptor*)

This function implements get audio stream current altsetting descriptor.

Parameters

classHandle	The class handle.
DescriptorType	The descriptor type.
DescriptorSub-Type	The descriptor subtype, 0 for no subtype, for standard endpoint , 0 stand for data endpoint.
Descriptor	The pointer of descriptor pointer.

Return values

kStatus_USB_Success	Get audio stream current altsetting descriptor request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

5.6.7.12 usb_status_t USB_HostAudioStreamGetCurrentAltsettingSpecificDescriptors (usb_host_class_handle *classHandle,* uint32_t *DescriptorType,* uint32_t *DescriptorSubType,* void ∗∗ *Descriptor*)

This function implements get audio stream current altsetting descriptor.

Parameters

classHandle	The class handle.
DescriptorType	The descriptor type.
DescriptorSub-Type	The descriptor subtype, 0 for no subtype.
Descriptor	The pointer of descriptor pointer.

Return values

kStatus_USB_Success	Get audio stream current altsetting descriptor request successfully.
kStatus_USB_Invalid-Handle	The classHandle is NULL pointer.

5.6.7.13   usb_status_t USB_HostAudioGetSetFeatureUnitRequest (usb_host_class_handle *classHandle,* uint32_t *csAndCn,* uint32_t *cmdCode,* void ∗ *buf,* uint32_t *bufLen,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

This function implements usb audio feature unit request.

Parameters

classHandle	The class handle.
csAndCn	The CS and CN or MCN for wValue field in setup Request.
cmdCode	The bRequest code in lower 8bit of lower word and get feature(1)/set feature(0) flag in higher 8bit of lower word.
buf	The feature unit request buffer pointer.
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, please reference to USB_HostSendSetup.

5.6.7.14 usb_status_t USB_HostAudioGetSetClockSourceRequest (usb_host_class_-handle *classHandle,* uint32_t *csAndCn,* uint32_t *cmdCode,* void ∗ *buf,* uint32_t *bufLen,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

This function implements usb audio feature unit request.

Parameters

classHandle	The class handle.
csAndCn	The CS and CN or MCN for wValue field in setup Request.
cmdCode	The bRequest code in lower 8bit of lower word and get clock(1)/set clock(0) flag in higher 8bit of lower word.
buf	The feature unit request buffer pointer.
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, please reference to USB_HostSendSetup.

5.6.7.15 usb_status_t USB_HostAudioGetSetEndpointRequest (usb_host_class_handle *classHandle,* uint32_t *csAndCn,* uint32_t *cmdCode,* void ∗ *buf,* uint32_t *bufLen,* transfer_callback_t *callbackFn,* void ∗ *callbackParam*)

This function implements usb audio feature unit request.

Parameters

classHandle	The class handle.
csAndCn	The CS for wValue field in setup Request.
cmdCode	The bRequest code in lower 8bit of lower word and get(1)/set(0) flag in higher 8bit of lower word.
buf	The feature unit request buffer pointer.
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, please reference to USB_HostSendSetup.

5.6.7.16 usb_status_t USB_HostAudioSetStreamOutDataInterval (usb_host_class_handle *classHandle,* uint8_t *intervalValue*)

when the low interval can satisfy the device's data bandwidth requirement, the interval can be increased to decrease the MCU loading. for example: the audio speaker device is 48K/2channels/2Bytes and the original interval is 125us, mps is 256Bytes. If using the 125us interval, the USB interrupt will trigger every 125us, it need much MCU loading (in FreeRTOS environment especially because there is task switch time). Change the interval as 1ms, it sill can satisfy the device's data bandwidth requirement as follow: the data lenght is $48 * 2 * 2 = 192$Bytes every ms, and the 256Bytes (mps) is bigger than 192Bytes, so the inteval can be changed to 1ms. Then host sends 192Bytes in one micro-frame of the 8 micro-frames (1ms), and there are no data transfers in the other 7 micro-frames.

Parameters

classHandle	The class handle.
intervalValue	The new interval value according to the interval descriptor.

USB AUDIO Class driver

Return values

kStatus_USB_Success	the change 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.

5.7 USB PHDC Class driver

5.7.1 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.

5.7.2 USB Host PHDC Initialization

When the personal healthcare device is attached, the PHDC initialization flow is as follows:
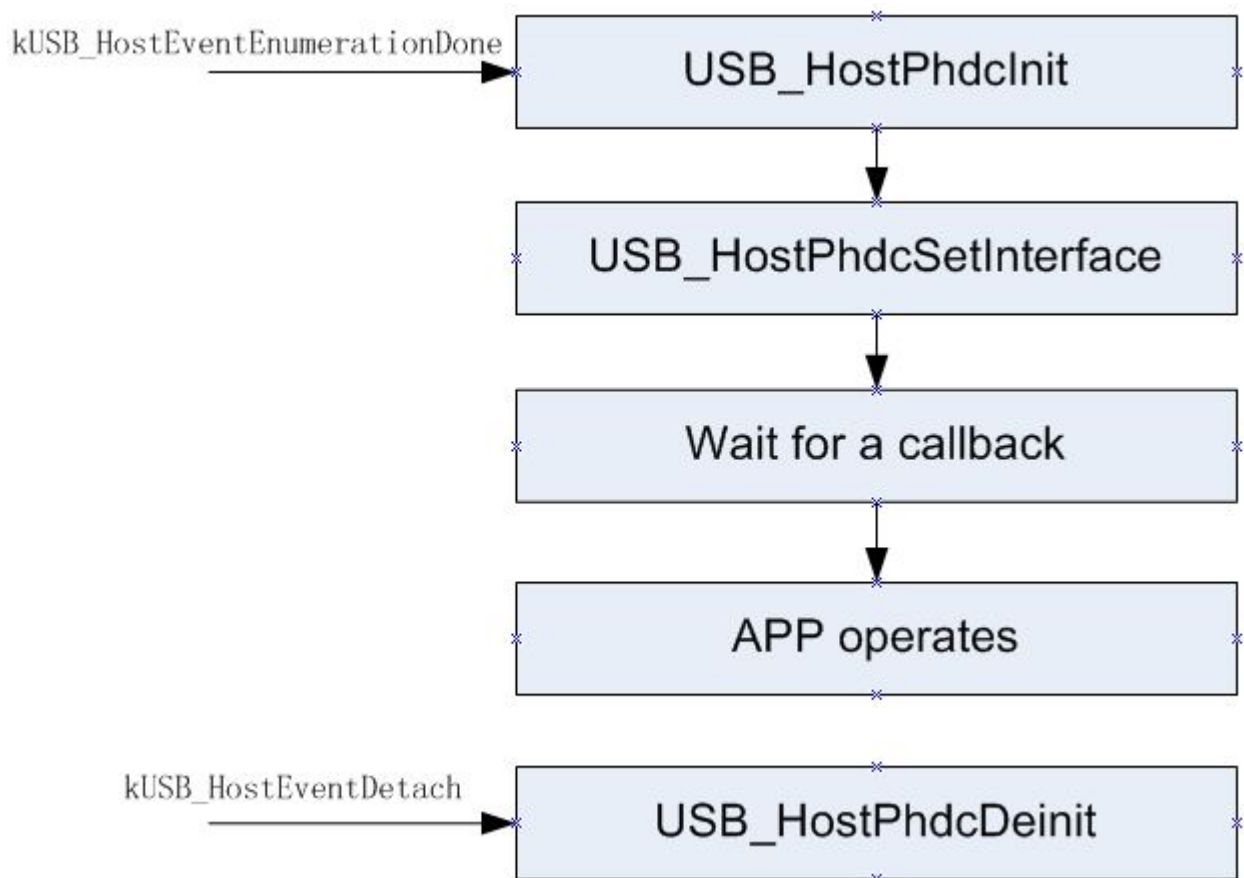
Figure 5.7.1: Host PHDC Initialization

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.

5.7.3 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.

5.7.4 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.

5.7.5 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.

5.7.6 Data Structure Documentation

5.7.6.1 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.

5.7.6.2 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.

5.7.6.3   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.

5.7.6.4   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.

5.7.6.5   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.

5.7.6.6   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.

5.7.7 Function Documentation

5.7.7.1 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.

5.7.7.2   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.

5.7.7.3 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.

5.7.7.4 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.

5.7.7.5 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.

5.7.7.6 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.

5.7.7.7 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.

5.7.7.8 usb_host_ep_t∗ USB_HostPhdcGetEndpointInformation (usb_host_class_handle *classHandle,* uint8_t *pipeType,* uint8_t *direction*)

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

USB PHDC Class driver

Parameters

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.

5.8   USB PRINTER Class driver

5.8.1   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 printer instance.

- 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.
- 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_HostPrinterGetDeviceId (usb_host_class_handle classHandle, uint8_t interface-Index, 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 ∗port-Status, 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.

5.8.2   Data Structure Documentation

5.8.2.1   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.

5.8.3 Function Documentation

5.8.3.1 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.

5.8.3.2 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.

5.8.3.3   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.

5.8.3.4   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.

5.8.3.5 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.

5.8.3.6 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_ISOC-HRONOUS, USB_ENDPOINT_BULK or USB_ENDPOINT_INTER-RUPT. See the usb_spec.h
in	*direction*	Pipe direction.

Return values

0	The classHandle is NULL.
Maximum	Packet size.

5.8.3.7 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.

5.8.3.8 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.

5.8.3.9 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 6
USB OS Adapter

Please reference to MCUXpresso SDK API Reference Manual.

MCUXpresso SDK USB Stack Host Reference Manual

Chapter 7
Data Structure Documentation

7.0.4 usb_host_hub_descriptor_t Struct Reference

HUB descriptor structure.

```
#include <usb_host_hub.h>
```

Data Fields

- uint8_t blength

  Number of bytes in this descriptor.
- uint8_t bdescriptortype

  Descriptor Type.
- uint8_t bnrports

  Number of downstream facing ports that this HUB supports.
- uint8_t whubcharacteristics [2]

  HUB characteristics please reference to Table 11-13 in usb2.0 specification.
- uint8_t bpwron2pwrgood

  Time (in 2 ms intervals) from the time the power-on sequence begins on a port until power is good on that port.
- uint8_t bhubcontrcurrent

  Maximum current requirements of the HUB Controller electronics in mA.
- uint8_t deviceremovable

  Indicates if a port has a removable device attached.

7.0.4.1 Field Documentation

7.0.4.1.1 uint8_t usb_host_hub_descriptor_t::bpwron2pwrgood

7.0.5 usb_host_hub_global_t Struct Reference

HUB application global structure.

```
#include <usb_host_hub_app.h>
```

Data Fields

- usb_host_handle hostHandle

  This HUB list belong to this host.
- usb_host_hub_instance_t ∗ hubProcess

  HUB in processing.

- usb_host_hub_instance_t ∗ hubList
    host's HUB list
- osa_mutex_handle_t hubMutex
    HUB mutex.
- uint32_t mutexBuffer [(OSA_MUTEX_HANDLE_SIZE+3)/4]
    The mutex buffer.

7.0.5.1 Field Documentation

7.0.5.1.1 uint32_t usb_host_hub_global_t::mutexBuffer[(OSA_MUTEX_HANDLE_SIZE+3)/4]

7.0.6 usb_host_hub_instance_t Struct Reference

HUB instance structure.

```
#include <usb_host_hub.h>
```

Data Fields

- struct _usb_host_hub_instance ∗ next
    Next HUB instance.
- usb_host_handle hostHandle
    Host handle.
- usb_device_handle deviceHandle
    Device handle.
- usb_host_interface_handle interfaceHandle
    Interface handle.
- usb_host_pipe_handle controlPipe
    Control pipe handle.
- usb_host_pipe_handle interruptPipe
    HUB interrupt in pipe handle.
- usb_host_hub_port_instance_t ∗ portList
    HUB's port instance list.
- usb_host_transfer_t ∗ controlTransfer
    Control transfer in progress.
- transfer_callback_t inCallbackFn
    Interrupt in callback.
- void ∗ inCallbackParam
    Interrupt in callback parameter.
- transfer_callback_t controlCallbackFn
    Control callback.
- void ∗ controlCallbackParam
    Control callback parameter.
- uint16_t totalThinktime
    HUB total think time.
- uint8_t hubLevel
    HUB level, the root HUB's level is 1.
- uint8_t hubDescriptor [7+(USB_HOST_HUB_MAX_PORT >> 3)+1]
    HUB descriptor buffer.

- uint8_t hubBitmapBuffer [(USB_HOST_HUB_MAX_PORT >> 3)+1]
  HUB receiving bitmap data buffer.
- uint8_t hubStatusBuffer [4]
  HUB status buffer.
- uint8_t portStatusBuffer [4]
  Port status buffer.
- uint8_t hubStatus
  HUB instance running status.
- uint8_t portCount
  HUB port count.
- uint8_t portIndex
  Record the index when processing ports in turn.
- uint8_t portProcess
  The port that is processing.
- uint8_t primeStatus
  Data prime transfer status.
- uint8_t invalid
  0/1, when invalid, cannot send transfer to the class
- uint8_t supportRemoteWakeup
  The HUB supports remote wakeup or not.
- uint8_t controlRetry
  Retry count for set remote wakeup feature.

7.0.7 usb_host_hub_port_instance_t Struct Reference

HUB port instance structure.

```
#include <usb_host_hub.h>
```

Data Fields

- usb_device_handle deviceHandle
  Device handle.
- uint8_t portStatus
  Port running status.
- uint8_t resetCount
  Port reset time.
- uint8_t speed
  Port's device speed.

7.0.8 usb_host_video_common_desc_t Struct Reference

usb video class Common Descriptor

```
#include <usb_host_video.h>
```

7.0.9   usb_host_video_ctrl_ct_desc_t Struct Reference

video control interface camera terminal descriptor structure

```
#include <usb_host_video.h>
```

7.0.10   usb_host_video_ctrl_header_desc_t Struct Reference

Video control interface header descriptor structure.

```
#include <usb_host_video.h>
```

7.0.11   usb_host_video_ctrl_it_desc_t Struct Reference

video control interface iutput terminal descriptor structure

```
#include <usb_host_video.h>
```

7.0.12   usb_host_video_ctrl_ot_desc_t Struct Reference

video control interface output terminal descriptor structure

```
#include <usb_host_video.h>
```

7.0.13   usb_host_video_ctrl_pu_desc_t Struct Reference

video control interface processing unit descriptor structure

```
#include <usb_host_video.h>
```

7.0.14   usb_host_video_ctrl_su_desc_t Struct Reference

video control interface selcetor unit descriptor structure

```
#include <usb_host_video.h>
```

7.0.15   usb_host_video_descriptor_union_t Union Reference

video descriptor uinon

```
#include <usb_host_video.h>
```

7.0.16 usb_host_video_instance_struct_t Struct Reference

Video instance structure, Video usb_host_class_handle pointer to this structure.

```
#include <usb_host_video.h>
```

Data Fields

- usb_host_handle hostHandle
    The handle of the USB host.
- usb_device_handle deviceHandle
    The handle of the USB device structure.
- usb_host_interface_handle streamIntfHandle
    This instance's video stream interface handle.
- usb_host_interface_handle controlIntfHandle
    This instance's control stream interface handle.
- usb_host_pipe_handle controlPipe
    Video class control pipe.
- usb_host_pipe_handle interruptPipe
    Video class interrupt pipe.
- usb_host_pipe_handle streamIsoInPipe
    Video class stream iso in pipe.
- usb_host_video_ctrl_header_desc_t ∗ vcHeaderDesc
    Video class control header descriptor pointer.
- usb_host_video_ctrl_it_desc_t ∗ vcInputTerminalDesc
    Video class control input terminal descriptor pointer.
- usb_host_video_ctrl_ot_desc_t ∗ vcOutputTerminalDesc
    Video class control output terminal descriptor pointer.
- usb_host_video_ctrl_pu_desc_t ∗ vcProcessingUnitDesc
    Video class control processing unit descriptor pointer.
- usb_host_video_stream_input_header_desc_t ∗ vsInputHeaderDesc
    Video class stream input header descriptor pointer.
- transfer_callback_t controlCallbackFn
    Video control transfer callback function.
- void ∗ controlCallbackParam
    Video control transfer callback parameter.
- usb_host_transfer_t ∗ controlTransfer
    On-going control transfer.
- transfer_callback_t streamIsoInCallbackFn
    Video stream ISO in transfer callback function.
- void ∗ streamIsoInCallbackParam
    Video stream ISO in transfer callback parameter.
- uint16_t interruptInPacketSize
    Video Interrupt in maximum packet size.
- uint16_t isoInPacketSize
    Video ISO in maximum packet size.
- uint8_t interruptInEpNum
    Video control interrupt in endpoint number.
- uint8_t isoInEpNum
    Video stream ISO in endpoint number.

7.0.17 usb_host_video_payload_header_t Struct Reference

video stream interface paylaod header descriptor structure

```
#include <usb_host_video.h>
```

7.0.18 usb_host_video_probe_commit_controls_t Struct Reference

video stream interface probe and commit controls descriptor structure

```
#include <usb_host_video.h>
```

7.0.19 usb_host_video_stream_input_header_desc_t Struct Reference

video stream interface input header descriptor structure

```
#include <usb_host_video.h>
```

7.0.20 usb_host_video_stream_output_header_desc_t Struct Reference

video stream interface output header descriptor structure

```
#include <usb_host_video.h>
```

7.0.21 usb_host_video_stream_payload_format_common_desc_t Struct Reference

video stream interface format descriptor structure common

```
#include <usb_host_video.h>
```

7.0.22 usb_host_video_stream_payload_frame_common_desc_t Struct Reference

video stream common frame descriptor structure

```
#include <usb_host_video.h>
```

7.0.23 usb_host_video_stream_payload_mjpeg_format_desc_t Struct Reference

video stream interface Motion-JPEG format descriptor structure

```
#include <usb_host_video.h>
```

7.0.24 usb_host_video_stream_payload_mjpeg_frame_desc_t Struct Reference

video stream interface Motion-JPEG frame descriptor structure

```
#include <usb_host_video.h>
```

7.0.25 usb_host_video_stream_payload_uncompressed_format_desc_t Struct Reference

video stream interface uncompressed format descriptor structure

```
#include <usb_host_video.h>
```

7.0.26 usb_host_video_stream_payload_uncompressed_frame_desc_t Struct Reference

video stream interface uncompressed frame descriptor structure

```
#include <usb_host_video.h>
```