
MCUXpresso SDK USB Stack Device Reference Manual

NXP Semiconductors

Document Number: MCUXSDKUSBDAPIRM

Rev. 0
May 2018



Contents

Chapter Overview

1.1	Introduction	1
1.2	USB Device Callback Work Flow	5
1.3	USB Device Class-Specific Request Work Flow	6

Chapter Definitions and structures

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

Chapter USB Class driver

3.1	Overview	13
3.2	Data Structure Documentation	15
3.2.1	struct usb_device_endpoint_struct_t	15
3.2.2	struct usb_device_endpoint_list_t	15
3.2.3	struct usb_device_interface_struct_t	15
3.2.4	struct usb_device_interfaces_struct_t	16
3.2.5	struct usb_device_interface_list_t	16
3.2.6	struct usb_device_class_struct_t	16
3.2.7	struct usb_device_class_config_struct_t	17
3.2.8	struct usb_device_class_config_list_struct_t	17
3.2.9	struct usb_device_control_request_struct_t	17
3.2.10	struct usb_device_get_descriptor_common_struct_t	18
3.2.11	struct usb_device_get_device_descriptor_struct_t	18
3.2.12	struct usb_device_get_device_qualifier_descriptor_struct_t	19

Contents

Section Number	Title	Page Number
3.2.13	struct usb_device_get_configuration_descriptor_struct_t	19
3.2.14	struct usb_device_get_bos_descriptor_struct_t	19
3.2.15	struct usb_device_get_string_descriptor_struct_t	20
3.2.16	struct usb_device_get_hid_descriptor_struct_t	20
3.2.17	struct usb_device_get_hid_report_descriptor_struct_t	21
3.2.18	struct usb_device_get_hid_physical_descriptor_struct_t	21
3.2.19	union usb_device_get_descriptor_common_union_t	22
3.2.20	struct usb_device_class_map_t	23
3.2.21	struct usb_device_common_class_struct_t	23
3.3	Enumeration Type Documentation	24
3.3.1	usb_device_class_type_t	24
3.3.2	usb_device_class_event_t	24
3.4	Function Documentation	24
3.4.1	USB_DeviceClassInit	24
3.4.2	USB_DeviceClassDeinit	24
3.4.3	USB_DeviceClassGetSpeed	25
3.4.4	USB_DeviceClassEvent	25
3.4.5	USB_DeviceClassCallback	26
3.4.6	USB_DeviceClassGetDeviceHandle	26
3.5	USB MSC Class driver	28
3.5.1	Overview	28
3.5.2	USB MSC driver	29
3.5.3	USB MSC UFI driver	37
3.6	USB CDC Class driver	44
3.6.1	Overview	44
3.6.2	USB CDC ACM Class driver	45
3.6.3	USB CDC RNDIS driver	59
3.7	USB AUDIO Class driver	79
3.7.1	Overview	79
3.7.2	Data Structure Documentation	82
3.7.3	Enumeration Type Documentation	83
3.7.4	Function Documentation	83
3.8	USB CCID Class driver	87
3.8.1	Overview	87
3.8.2	Data Structure Documentation	91
3.8.3	Macro Definition Documentation	114
3.8.4	Enumeration Type Documentation	114
3.8.5	Function Documentation	115
3.9	USB HID Class driver	119

Contents

Section Number	Title	Page Number
3.9.1	Overview	119
3.9.2	Data Structure Documentation	120
3.9.3	Macro Definition Documentation	122
3.9.4	Enumeration Type Documentation	122
3.9.5	Function Documentation	122
3.10	USB PHDC Class driver	126
3.10.1	Overview	126
3.10.2	Data Structure Documentation	127
3.10.3	Enumeration Type Documentation	128
3.10.4	Function Documentation	128
3.11	USB PRINTER Class driver	132
3.11.1	Overview	132
3.11.2	Data Structure Documentation	133
3.11.3	Enumeration Type Documentation	134
3.11.4	Function Documentation	134
3.12	USB VIDEO Class driver	138
3.12.1	Overview	138
3.12.2	Data Structure Documentation	150
3.12.3	Enumeration Type Documentation	157
3.12.4	Function Documentation	158
Chapter	USB Device driver	
4.1	Overview	161
4.2	Data Structure Documentation	164
4.2.1	struct usb_device_endpoint_callback_message_struct_t	164
4.2.2	struct usb_device_endpoint_callback_struct_t	165
4.2.3	struct usb_device_endpoint_init_struct_t	165
4.2.4	struct usb_device_endpoint_status_struct_t	165
4.3	Macro Definition Documentation	165
4.3.1	USB_SETUP_PACKET_SIZE	165
4.3.2	USB_DeviceKhciTaskFunction	165
4.3.3	USB_DeviceEhciTaskFunction	166
4.3.4	USB_DeviceLpcIp3511TaskFunction	166
4.4	Typedef Documentation	166
4.4.1	usb_device_endpoint_callback_t	166
4.4.2	usb_device_callback_t	167
4.5	Enumeration Type Documentation	167
4.5.1	usb_device_status_t	167

Contents

Section Number	Title	Page Number
4.5.2	usb_device_state_t	168
4.5.3	usb_device_endpoint_status_t	168
4.5.4	usb_device_event_t	168
4.6	Function Documentation	169
4.6.1	USB_DeviceInit	169
4.6.2	USB_DeviceRun	170
4.6.3	USB_DeviceStop	171
4.6.4	USB_DeviceDeinit	171
4.6.5	USB_DeviceSendRequest	172
4.6.6	USB_DeviceRecvRequest	173
4.6.7	USB_DeviceCancel	174
4.6.8	USB_DeviceInitEndpoint	175
4.6.9	USB_DeviceDeinitEndpoint	176
4.6.10	USB_DeviceStallEndpoint	177
4.6.11	USB_DeviceUnstallEndpoint	178
4.6.12	USB_DeviceGetStatus	178
4.6.13	USB_DeviceSetStatus	179
4.6.14	USB_DeviceTaskFunction	179
4.6.15	USB_DeviceKhciIsrFunction	180
4.6.16	USB_DeviceEhciIsrFunction	180
4.6.17	USB_DeviceLpcIp3511IsrFunction	180
4.6.18	USB_DeviceGetVersion	180
4.6.19	USB_DeviceUpdateHwTick	181
4.7	USB Device Controller driver	183
4.7.1	Overview	183
4.7.2	Data Structure Documentation	184
4.7.3	Enumeration Type Documentation	186
4.7.4	USB Device Controller KHCI driver	187
4.7.5	USB Device Controller EHCI driver	193
4.7.6	USB Device Controller LPC IP3511 driver	199
4.8	USB Device Spec Chapter 9 driver	206
4.8.1	Overview	206
4.8.2	Enumeration Type Documentation	206
4.8.3	Function Documentation	207
4.9	USB Device Configuration	209
Chapter	USB OS Adapter	
5.1	Overview	211
5.2	Enumeration Type Documentation	213

Contents

Section Number	Title	Page Number
5.2.1	usb_osa_status_t	213
5.2.2	usb_osa_event_mode_t	213
5.3	Function Documentation	213
5.3.1	USB_OsaMemoryAllocate	213
5.3.2	USB_OsaMemoryFree	213
5.3.3	USB_OsaEventCreate	214
5.3.4	USB_OsaEventDestroy	214
5.3.5	USB_OsaEventSet	215
5.3.6	USB_OsaEventWait	216
5.3.7	USB_OsaEventCheck	217
5.3.8	USB_OsaEventClear	218
5.3.9	USB_OsaSemCreate	218
5.3.10	USB_OsaSemDestroy	219
5.3.11	USB_OsaSemPost	219
5.3.12	USB_OsaSemWait	220
5.3.13	USB_OsaMutexCreate	220
5.3.14	USB_OsaMutexDestroy	221
5.3.15	USB_OsaMutexLock	221
5.3.16	USB_OsaMutexUnlock	222
5.3.17	USB_OsaMsgqCreate	222
5.3.18	USB_OsaMsgqDestroy	223
5.3.19	USB_OsaMsgqSend	223
5.3.20	USB_OsaMsgqRecv	224
5.3.21	USB_OsaMsgqCheck	224

Chapter 1

Overview

1.1 Introduction

The USB device stack is composed of the USB controller driver only, which consists of the common controller driver and the controller (like: xHCI in Kinetis) driver. The device class driver and the USB framework to handle the standard enumeration and request defined by USB specification 2.0 are moved to the application layer. These two parts are example-specific to reduce the footprint of the examples.

Note

The xHCI represents either EHCI or KHCI, not the XHCI for USB 3.0.

In the USB Device stack, there are two different USB applications. One is the lite version and the other is similar to the examples in the previous USB stack.

The whole architecture and components of USB stack are shown below:

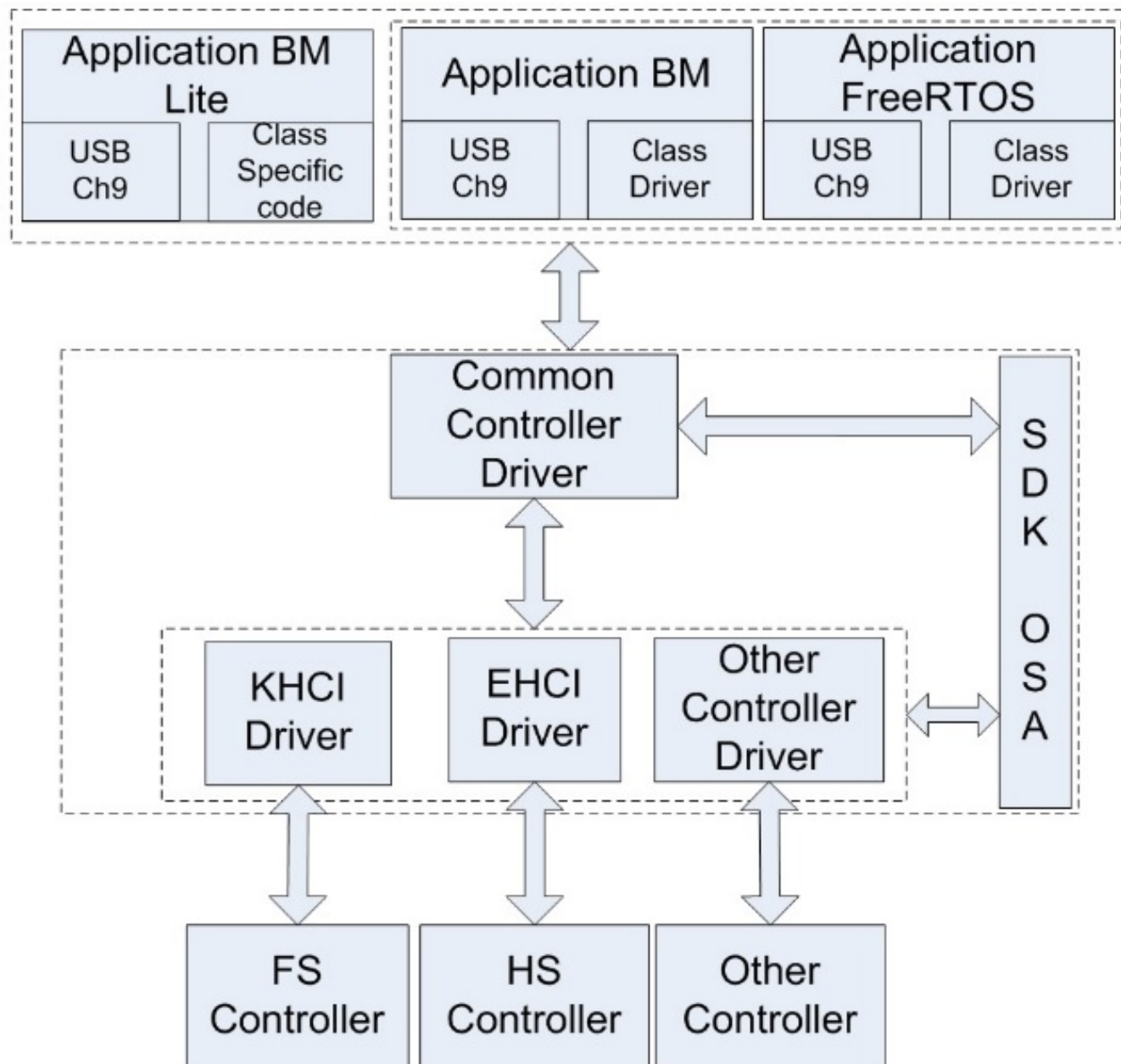


Figure 1.1.1: USB device stack architecture

For the lite version application, the code size is smaller than the non-lite version. However, an obvious drawback of the new architecture is that customers need to use the controller driver API to implement the standard enumeration process, the class-specific process, and the customer-specific functionality.

The device stack initialization sequence for the lite version application is as follows:

1. Initialize the Pin Mux, USB clock, and so on. If the SoC has a USB KHCI-dedicated RAM, the RAM memory needs to be clear after the KHCI clock is enabled. When the demo uses USB EHCI IP, the USB KHCI dedicated-RAM can't be used and the memory can't be accessed.

2. Initialize the USB device stack by calling the API `USB_DeviceInit`.
3. When the device task is enabled, create the USB device task by using the device handle, returned from `USB_DeviceInit`, as the task parameter when the environment is an RTOS.
4. Install the USB ISR.
5. Enable the USB interrupt and the interrupt priority.
6. Start the USB device by calling the `USB_DeviceRun`.

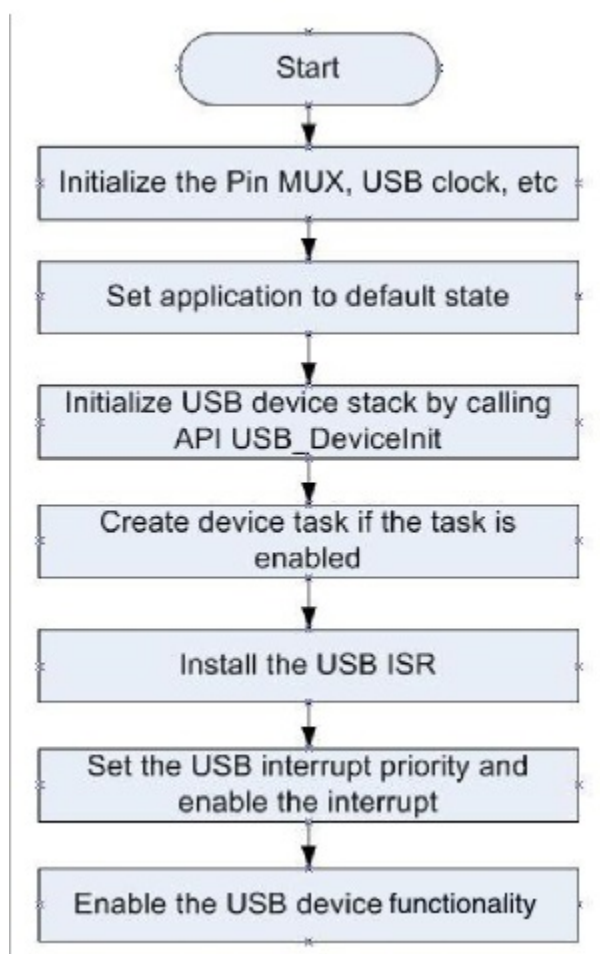


Figure 1.1.2: USB device initialization for lite version

To assist customers with less concerns about the footprint and focus on ease of use of the USB stack, a generic `usb_ch9` implementation is provided and the specified class driver, such as HID class driver, CDC class driver, and so on. This implementation is more generic, it can be reused in different examples and the APIs are easier to use. However, some callback functions need to be implemented and the code size is larger.

The device stack initialization sequence for non-lite version application is as follows:

1. Initialize the Pin Mux, USB clock, and so on. If the SOC has the USB KHCI-dedicated RAM, the RAM memory needs to be clear after the KHCI clock is enabled. When the demo uses USB EHCI

Introduction

IP, the USB KHCI-dedicated RAM can't be used and the memory can't be accessed.

Note

The `USB_GLOBAL`, `USB_BDT`, and `USB_RAM_ADDRESS_ALIGNMENT(n)` are only used for USB device stack. The USB device global variables are put into the section `m_usb_global` or `m_usb_bdt` by using the MACRO `USB_GLOBAL` and `USB_BDT`. In this way, the USB device global variables can be linked into USB dedicated RAM by changing the linker file. This feature can only be enabled when the USB dedicated RAM is not less than 2 K Bytes.

2. Initialize the USB device stack by calling the API `USB_DeviceClassInit`. Initialize each application.
3. Get each class handle from the `usb_device_class_config_struct_t::classHandle`.
4. When the device task is enabled, create the USB device task by using the device handle, returned from `USB_DeviceClassInit`, as the task parameter when the environment is RTOS.
5. Install the USB ISR.
6. Enable the USB interrupt and the interrupt priority.
7. Start the USB device by calling the `USB_DeviceRun`.

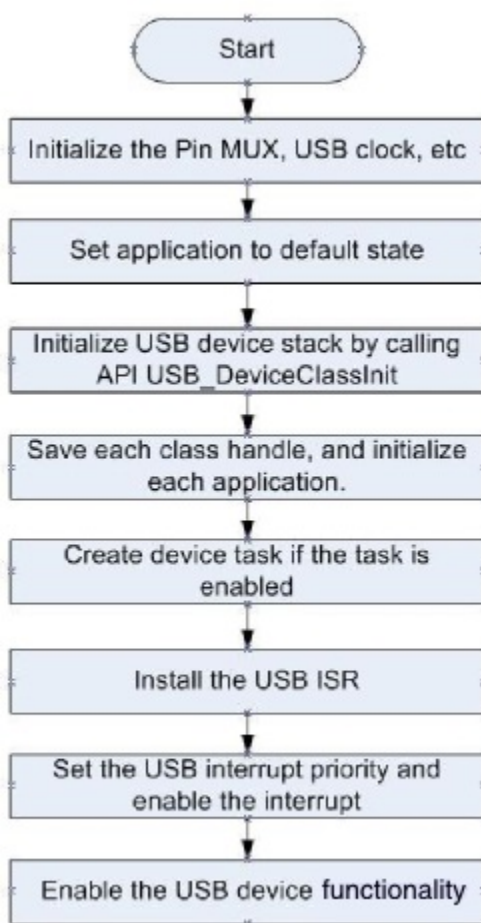


Figure 1.1.3: USB device initialization for non-lite version

To support different RTOSes with the same code base, the OSA is used inside the USB stack to wrap the differences between RTOSes.

Note

The OSA should not be used in the USB application. As a result, from the USB application's viewpoint, the OSA is invisible.

1.2 USB Device Callback Work Flow

The device callback is registered when the `USB_DeviceInit` function is called.

The following events should be processed in this callback function:

- `kUsbDeviceEventBusReset`
When the application receives this event, the device has received a BUS RESET signal. In the event, the control pipe should be initialized. See the work flow. The parameter `eventParam` is not used.
- `kUsbDeviceEventSetConfiguration`
When the application receives this event, the host has sent a set configuration request. The configuration value can be received from the parameter `eventParam`. In the event, the application configuration can be set. Initialize each interface in the current configuration by using zero as an alternate setting.
- `kUsbDeviceEventSetInterface`
When the application receives this event, the host sent a set alternate setting request of an interface. The interface and alternate setting value can be received from the parameter `eventParam`. The `eventParam` points to a `uint16_t` variable. The high 8-bit is interface value and the low 8-bit is alternate setting. In the event, the application changes the alternate setting of this interface if the new alternate setting is not equal to the current setting.
Normally, change the steps as follows:
 1. Cancel all transfers of the current alternate setting in this interface.
 2. De-initialize all pipes of the current alternate setting in this interface.
 3. Initialize all pipes of the new alternate setting in this interface.
 4. Prime the transfers of the new setting.

For example,

```
uint16_t*   temp16 = (uint16_t*)eventParam;
uint8_t     interface = (uint8_t)((*temp16 & 0xFF00) >> 0x08);
currentAlternateSetting[interface] = (uint8_t)(*temp16 & 0x00FF);
```

The device callback event work flow:

USB Device Class-Specific Request Work Flow

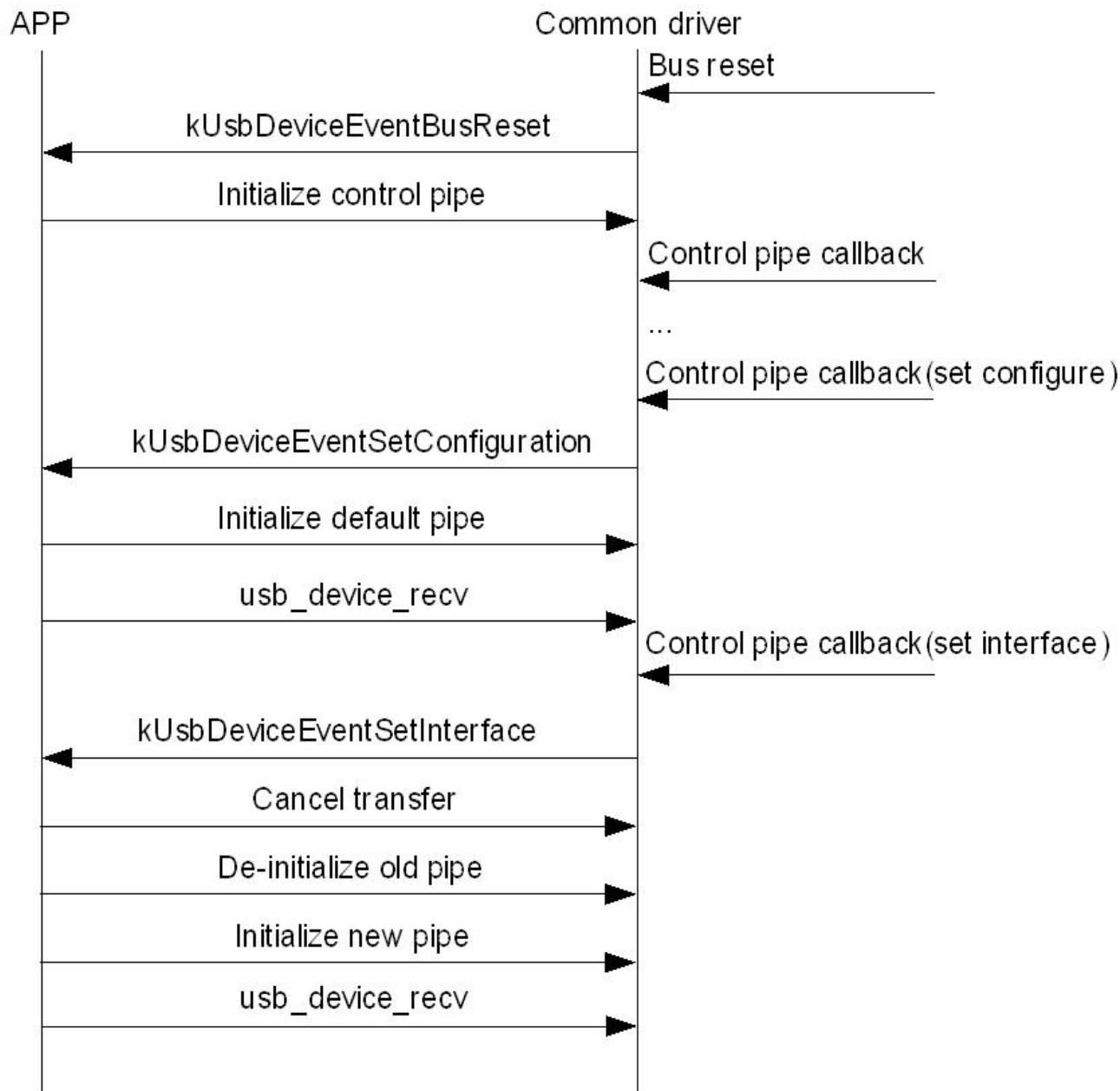


Figure 1.2.1: USB device callback working flow

1.3 USB Device Class-Specific Request Work Flow

The class specific request can be classified into two types according to whether there is the data stage in a setup transfer. The section describes class specific request with data stage only. For the class-specific request without data stage, the case is quite simple, we don't describe here. Depend on the data direction, there are two cases, host wants to send data to device and host wants to get data from device.

USB Device Class-Specific Request with Data Sent from Host

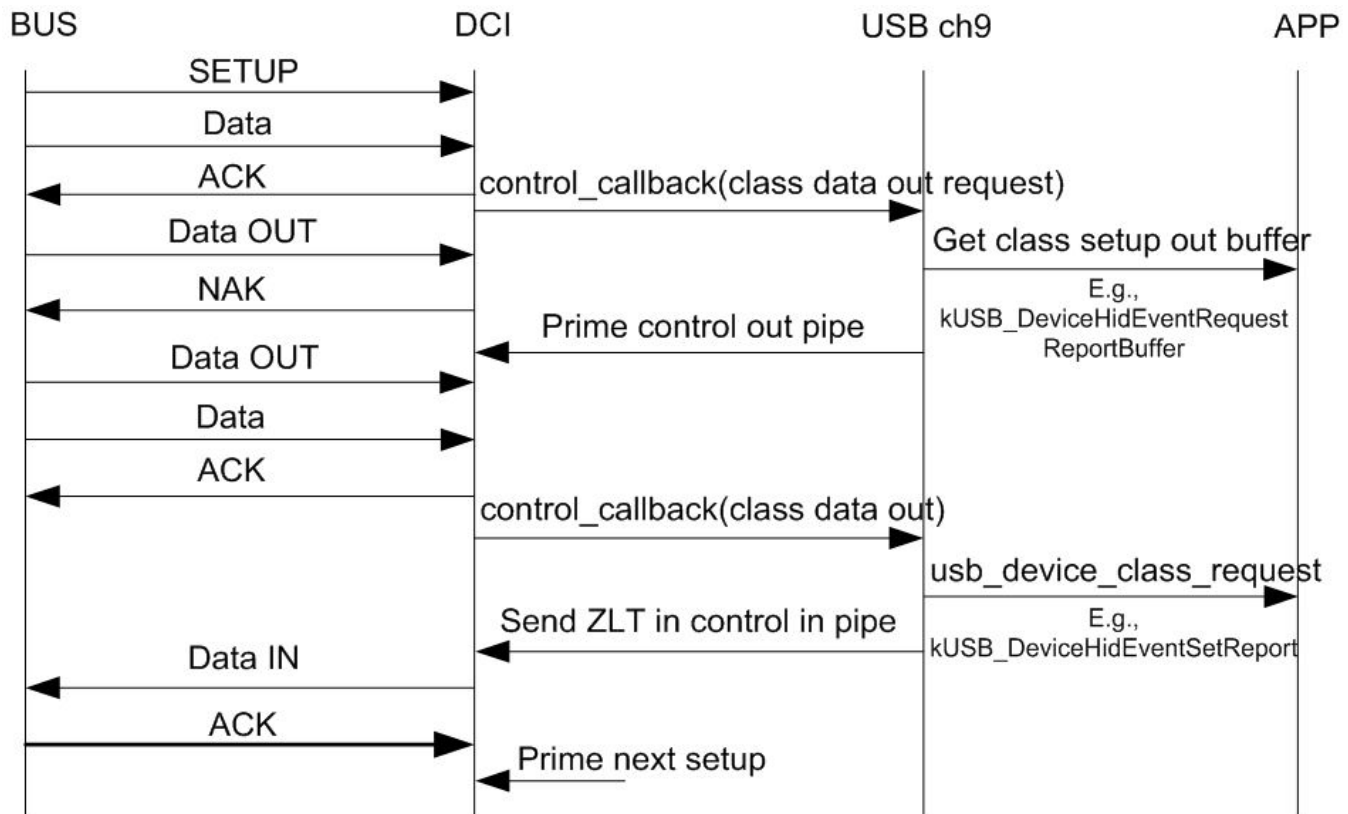


Figure 1.3.1: USB Device Class-Specific Request with Data Sent from Host

USB Device Class-Specific Request Work Flow

USB Device Class-Specific Request with Data Sent to Host

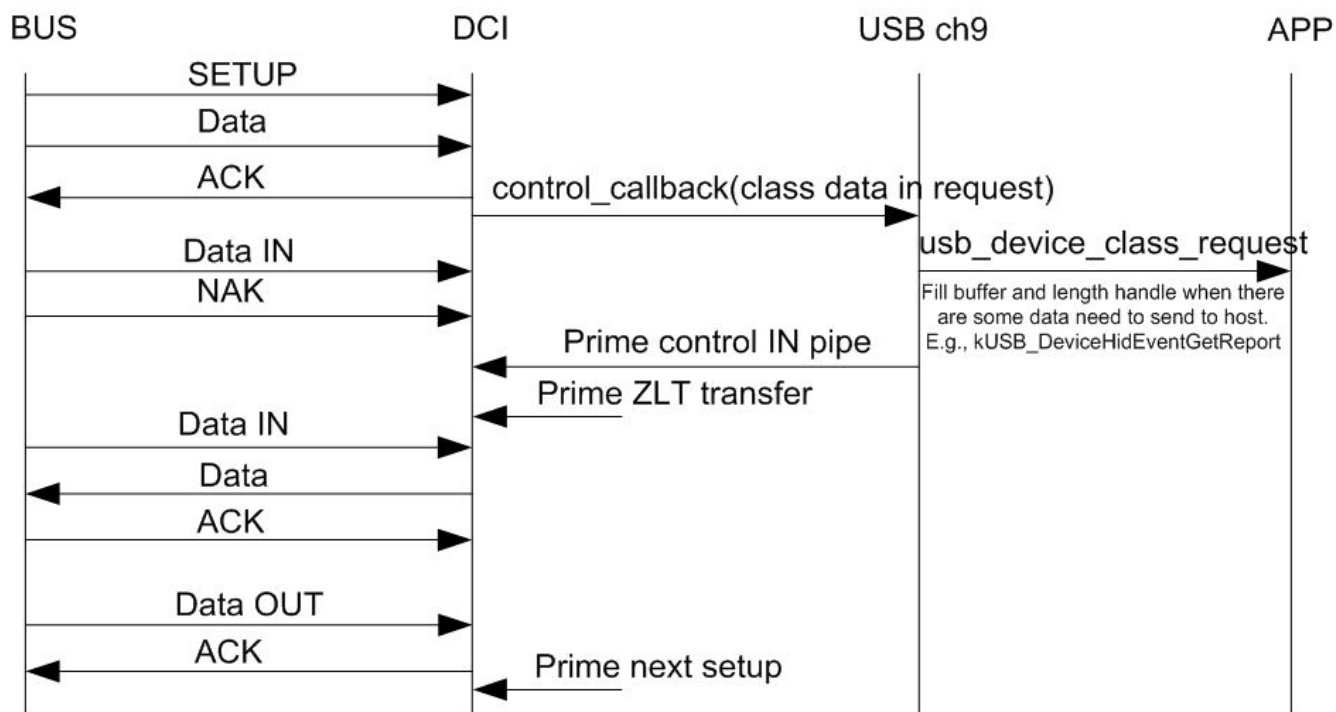


Figure 1.3.2: USB Device Class-Specific Request with Data Sent to Host

Chapter 2

Definitions and structures

2.1 Overview

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

Data Structures

- struct [usb_version_t](#)
USB stack version fields. [More...](#)

Macros

- #define [USB_STACK_VERSION_MAJOR](#) (0x01U)
Defines USB stack major version.
- #define [USB_STACK_VERSION_MINOR](#) (0x00U)
Defines USB stack minor version.
- #define [USB_STACK_VERSION_BUGFIX](#) (0x00U)
Defines USB stack bugfix version.
- #define [USB_MAKE_VERSION](#)(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))
USB stack version definition.

Typedefs

- typedef void * [usb_host_handle](#)
USB host handle type define.
- typedef void * [usb_device_handle](#)
USB device handle type define.
- typedef void * [usb_otg_handle](#)
USB OTG handle type define.

Enumerations

- enum `usb_status_t` {
 `kStatus_USB_Success` = 0x00U,
 `kStatus_USB_Error`,
 `kStatus_USB_Busy`,
 `kStatus_USB_InvalidHandle`,
 `kStatus_USB_InvalidParameter`,
 `kStatus_USB_InvalidRequest`,
 `kStatus_USB_ControllerNotFound`,
 `kStatus_USB_InvalidControllerInterface`,
 `kStatus_USB_NotSupported`,
 `kStatus_USB_Retry`,
 `kStatus_USB_TransferStall`,
 `kStatus_USB_TransferFailed`,
 `kStatus_USB_AllocFail`,
 `kStatus_USB_LackSwapBuffer`,
 `kStatus_USB_TransferCancel`,
 `kStatus_USB_BandwidthFail`,
 `kStatus_USB_MSDDStatusFail` }

USB error code.

- enum `usb_controller_index_t` {
 `kUSB_ControllerKhci0` = 0U,
 `kUSB_ControllerKhci1` = 1U,
 `kUSB_ControllerEhci0` = 2U,
 `kUSB_ControllerEhci1` = 3U,
 `kUSB_ControllerLpcIp3511Fs0` = 4U,
 `kUSB_ControllerLpcIp3511Fs1`,
 `kUSB_ControllerLpcIp3511Hs0` = 6U,
 `kUSB_ControllerLpcIp3511Hs1`,
 `kUSB_ControllerOhci0` = 8U,
 `kUSB_ControllerOhci1` = 9U,
 `kUSB_ControllerIp3516Hs0` = 10U,
 `kUSB_ControllerIp3516Hs1` }

USB controller ID.

2.2 Data Structure Documentation

2.2.1 struct `usb_version_t`

Data Fields

- uint8_t `major`
Major.
- uint8_t `minor`
Minor.

- uint8_t [bugfix](#)
Bug fix.

2.3 Typedef Documentation

2.3.1 typedef void* usb_device_handle

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

2.4 Enumeration Type Documentation

2.4.1 enum usb_status_t

Enumerator

kStatus_USB_Success Success.
kStatus_USB_Error Failed.
kStatus_USB_Busy Busy.
kStatus_USB_InvalidHandle Invalid handle.
kStatus_USB_InvalidParameter Invalid parameter.
kStatus_USB_InvalidRequest Invalid request.
kStatus_USB_ControllerNotFound Controller cannot be found.
kStatus_USB_InvalidControllerInterface Invalid controller interface.
kStatus_USB_NotSupported Configuration is not supported.
kStatus_USB_Retry Enumeration get configuration retry.
kStatus_USB_TransferStall Transfer stalled.
kStatus_USB_TransferFailed Transfer failed.
kStatus_USB_AllocFail Allocation failed.
kStatus_USB_LackSwapBuffer Insufficient swap buffer for KHCI.
kStatus_USB_TransferCancel The transfer cancelled.
kStatus_USB_BandwidthFail Allocate bandwidth failed.
kStatus_USB_MSDStatusFail For MSD, the CSW status means fail.

2.4.2 enum usb_controller_index_t

Enumerator

kUSB_ControllerKhci0 KHCI 0U.
kUSB_ControllerKhci1 KHCI 1U, Currently, there are no platforms which have two KHCI IPs, this is reserved to be used in the future.
kUSB_ControllerEhci0 EHCI 0U.
kUSB_ControllerEhci1 EHCI 1U, Currently, there are no platforms which have two EHCI IPs, this is reserved to be used in the future.
kUSB_ControllerLpcIp3511Fs0 LPC USB IP3511 FS controller 0.

Enumeration Type Documentation

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

kUSB_ControllerLpcIp3511Hs0 LPC USB IP3511 HS controller 0.

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

kUSB_ControllerOhci0 OHCI 0U.

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

kUSB_ControllerIp3516Hs0 IP3516HS 0U.

kUSB_ControllerIp3516Hs1 IP3516HS 1U, Currently, there are no platforms which have two IP3516HS IPs, this is reserved to be used in the future.

Chapter 3

USB Class driver

3.1 Overview

Modules

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

Data Structures

- struct [usb_device_endpoint_struct_t](#)
Obtains the endpoint data structure. [More...](#)
- struct [usb_device_endpoint_list_t](#)
Obtains the endpoint group. [More...](#)
- struct [usb_device_interface_struct_t](#)
Obtains the interface list data structure. [More...](#)
- struct [usb_device_interfaces_struct_t](#)
Obtains the interface data structure. [More...](#)
- struct [usb_device_interface_list_t](#)
Obtains the interface group. [More...](#)
- struct [usb_device_class_struct_t](#)
Obtains the class data structure. [More...](#)
- struct [usb_device_class_config_struct_t](#)
Obtains the device class information structure. [More...](#)
- struct [usb_device_class_config_list_struct_t](#)
Obtains the device class configuration structure. [More...](#)
- struct [usb_device_control_request_struct_t](#)
Obtains the control request structure. [More...](#)
- struct [usb_device_get_descriptor_common_struct_t](#)
Obtains the control get descriptor request common structure. [More...](#)
- struct [usb_device_get_device_descriptor_struct_t](#)
Obtains the control get device descriptor request structure. [More...](#)
- struct [usb_device_get_device_qualifier_descriptor_struct_t](#)
Obtains the control get device qualifier descriptor request structure. [More...](#)
- struct [usb_device_get_configuration_descriptor_struct_t](#)
Obtains the control get configuration descriptor request structure. [More...](#)
- struct [usb_device_get_bos_descriptor_struct_t](#)
Obtains the control get bos descriptor request structure. [More...](#)
- struct [usb_device_get_string_descriptor_struct_t](#)
Obtains the control get string descriptor request structure. [More...](#)

Overview

- struct [usb_device_get_hid_descriptor_struct_t](#)
Obtains the control get HID descriptor request structure. [More...](#)
- struct [usb_device_get_hid_report_descriptor_struct_t](#)
Obtains the control get HID report descriptor request structure. [More...](#)
- struct [usb_device_get_hid_physical_descriptor_struct_t](#)
Obtains the control get HID physical descriptor request structure. [More...](#)
- union [usb_device_get_descriptor_common_union_t](#)
Obtains the control get descriptor request common union. [More...](#)
- struct [usb_device_class_map_t](#)
Define class driver interface structure. [More...](#)
- struct [usb_device_common_class_struct_t](#)
Structure holding common class state information. [More...](#)

Macros

- #define [class_handle_t](#) uint32_t
Macro to define class handle.

Typedefs

- typedef [usb_status_t](#)(* [usb_device_class_init_call_t](#))(uint8_t controllerId, [usb_device_class_config_struct_t](#) *classConfig, [class_handle_t](#) *classHandle)
Define function type for class device instance initialization.
- typedef [usb_status_t](#)(* [usb_device_class_deinit_call_t](#))(class_handle_t handle)
Define function type for class device instance deinitialization, internal.
- typedef [usb_status_t](#)(* [usb_device_class_event_callback_t](#))(void *classHandle, uint32_t event, void *param)
Define function type for class device instance Event change.

Enumerations

- enum [usb_device_class_type_t](#)
Available class types.
- enum [usb_device_class_event_t](#)
Available common class events.

Functions

- [usb_status_t](#) [USB_DeviceClassInit](#) (uint8_t controllerId, [usb_device_class_config_list_struct_t](#) *configList, [usb_device_handle](#) *handle)
Initializes the common class and the supported classes.
- [usb_status_t](#) [USB_DeviceClassDeinit](#) (uint8_t controllerId)
Deinitializes the common class and the supported classes.
- [usb_status_t](#) [USB_DeviceClassGetSpeed](#) (uint8_t controllerId, uint8_t *speed)
Gets the USB bus speed.
- [usb_status_t](#) [USB_DeviceClassEvent](#) ([usb_device_handle](#) handle, [usb_device_class_event_t](#) event, void *param)
Handles the event passed to the class drivers.
- [usb_status_t](#) [USB_DeviceClassCallback](#) ([usb_device_handle](#) handle, uint32_t event, void *param)
Handles the common class callback.

- `usb_status_t USB_DeviceClassGetDeviceHandle (uint8_t controllerId, usb_device_handle *handle)`
Gets the device handle according to the controller ID.

3.2 Data Structure Documentation

3.2.1 struct usb_device_endpoint_struct_t

Define the endpoint data structure.

Data Fields

- `uint8_t endpointAddress`
Endpoint address.
- `uint8_t transferType`
Endpoint transfer type.
- `uint16_t maxPacketSize`
Endpoint maximum packet size.

3.2.2 struct usb_device_endpoint_list_t

Structure representing endpoints and the number of endpoints that the user wants.

Data Fields

- `uint8_t count`
How many endpoints in current interface.
- `usb_device_endpoint_struct_t * endpoint`
Endpoint structure list.

3.2.3 struct usb_device_interface_struct_t

Structure representing an interface.

Data Fields

- `uint8_t alternateSetting`
Alternate setting number.
- `usb_device_endpoint_list_t endpointList`
Endpoints of the interface.
- `void * classSpecific`
Class specific structure handle.

3.2.4 struct usb_device_interfaces_struct_t

Structure representing interface.

Data Fields

- uint8_t [classCode](#)
Class code of the interface.
- uint8_t [subclassCode](#)
Subclass code of the interface.
- uint8_t [protocolCode](#)
Protocol code of the interface.
- uint8_t [interfaceNumber](#)
Interface number.
- [usb_device_interface_struct_t](#) * [interface](#)
Interface structure list.
- uint8_t [count](#)
Number of interfaces in the current interface.

3.2.5 struct usb_device_interface_list_t

Structure representing how many interfaces in one class type.

Data Fields

- uint8_t [count](#)
Number of interfaces of the class.
- [usb_device_interfaces_struct_t](#) * [interfaces](#)
All interfaces.

3.2.6 struct usb_device_class_struct_t

Structure representing how many configurations in one class type.

Data Fields

- [usb_device_interface_list_t](#) * [interfaceList](#)
Interfaces of the class.
- [usb_device_class_type_t](#) [type](#)
Class type.
- uint8_t [configurations](#)
Number of configurations of the class.

3.2.7 struct usb_device_class_config_struct_t

Structure representing the device class information. This structure only can be stored in RAM space.

Data Fields

- `usb_device_class_callback_t` [classCallback](#)
Class callback function to handle the device status-related event for the specified type of class.
- [class_handle_t](#) `classHandle`
The class handle of the class, filled by the common driver.
- `usb_device_class_struct_t *` [classInformation](#)
Detailed information of the class.

3.2.7.0.0.1 Field Documentation

3.2.7.0.0.1.1 `class_handle_t usb_device_class_config_struct_t::classHandle`

3.2.8 struct usb_device_class_config_list_struct_t

Structure representing the device class configuration information.

Data Fields

- `usb_device_class_config_struct_t *` `config`
Array of class configuration structures.
- `usb_device_callback_t` `deviceCallback`
Device callback function.
- `uint8_t` `count`
Number of class supported.

3.2.9 struct usb_device_control_request_struct_t

This structure is used to pass the control request information. The structure is used in following two cases.

1. Case one, the host wants to send data to the device in the control data stage:
 - a. If a setup packet is received, the structure is used to pass the setup packet data and wants to get the buffer to receive data sent from the host. The field `isSetup` is 1. The length is the requested buffer length. The buffer is filled by the class or application by using the valid buffer address. The setup is the setup packet address.
 - b. If the data received is sent by the host, the structure is used to pass the data buffer address and the data length sent by the host. In this way, the field `isSetup` is 0. The buffer is the address of the data sent from the host. The length is the received data length. The setup is the setup packet address.
2. Case two, the host wants to get data from the device in control data stage:

Data Structure Documentation

If the setup packet is received, the structure is used to pass the setup packet data and wants to get the data buffer address to send data to the host. The field `isSetup` is 1. The `length` is the requested data length. The buffer is filled by the class or application by using the valid buffer address. The setup is the setup packet address.

Data Fields

- `usb_setup_struct_t * setup`
The pointer of the setup packet data.
- `uint8_t * buffer`
Pass the buffer address.
- `uint32_t length`
Pass the buffer length or requested length.
- `uint8_t isSetup`
Indicates whether a setup packet is received.

3.2.9.0.0.2 Field Documentation

3.2.9.0.0.2.1 `usb_setup_struct_t* usb_device_control_request_struct_t::setup`

3.2.9.0.0.2.2 `uint8_t* usb_device_control_request_struct_t::buffer`

3.2.9.0.0.2.3 `uint32_t usb_device_control_request_struct_t::length`

3.2.9.0.0.2.4 `uint8_t usb_device_control_request_struct_t::isSetup`

3.2.10 struct `usb_device_get_descriptor_common_struct_t`

Data Fields

- `uint8_t * buffer`
Pass the buffer address.
- `uint32_t length`
Pass the buffer length.

3.2.10.0.0.3 Field Documentation

3.2.10.0.0.3.1 `uint8_t* usb_device_get_descriptor_common_struct_t::buffer`

3.2.10.0.0.3.2 `uint32_t usb_device_get_descriptor_common_struct_t::length`

3.2.11 struct `usb_device_get_device_descriptor_struct_t`

Data Fields

- `uint8_t * buffer`
Pass the buffer address.

- uint32_t [length](#)
Pass the buffer length.

3.2.11.0.0.4 Field Documentation

3.2.11.0.0.4.1 uint8_t* [usb_device_get_device_descriptor_struct_t::buffer](#)

3.2.11.0.0.4.2 uint32_t [usb_device_get_device_descriptor_struct_t::length](#)

3.2.12 struct [usb_device_get_device_qualifier_descriptor_struct_t](#)

Data Fields

- uint8_t * [buffer](#)
Pass the buffer address.
- uint32_t [length](#)
Pass the buffer length.

3.2.12.0.0.5 Field Documentation

3.2.12.0.0.5.1 uint8_t* [usb_device_get_device_qualifier_descriptor_struct_t::buffer](#)

3.2.12.0.0.5.2 uint32_t [usb_device_get_device_qualifier_descriptor_struct_t::length](#)

3.2.13 struct [usb_device_get_configuration_descriptor_struct_t](#)

Data Fields

- uint8_t * [buffer](#)
Pass the buffer address.
- uint32_t [length](#)
Pass the buffer length.
- uint8_t [configuration](#)
The configuration number.

3.2.13.0.0.6 Field Documentation

3.2.13.0.0.6.1 uint8_t* [usb_device_get_configuration_descriptor_struct_t::buffer](#)

3.2.13.0.0.6.2 uint32_t [usb_device_get_configuration_descriptor_struct_t::length](#)

3.2.13.0.0.6.3 uint8_t [usb_device_get_configuration_descriptor_struct_t::configuration](#)

3.2.14 struct [usb_device_get_bos_descriptor_struct_t](#)

Data Fields

- uint8_t * [buffer](#)

Data Structure Documentation

- *Pass the buffer address.*
- uint32_t [length](#)
Pass the buffer length.

3.2.14.0.0.7 Field Documentation

3.2.14.0.0.7.1 uint8_t* usb_device_get_bos_descriptor_struct_t::buffer

3.2.14.0.0.7.2 uint32_t usb_device_get_bos_descriptor_struct_t::length

3.2.15 struct usb_device_get_string_descriptor_struct_t

Data Fields

- uint8_t * [buffer](#)
Pass the buffer address.
- uint32_t [length](#)
Pass the buffer length.
- uint16_t [languageId](#)
Language ID.
- uint8_t [stringIndex](#)
String index.

3.2.15.0.0.8 Field Documentation

3.2.15.0.0.8.1 uint8_t* usb_device_get_string_descriptor_struct_t::buffer

3.2.15.0.0.8.2 uint32_t usb_device_get_string_descriptor_struct_t::length

3.2.15.0.0.8.3 uint16_t usb_device_get_string_descriptor_struct_t::languageId

3.2.15.0.0.8.4 uint8_t usb_device_get_string_descriptor_struct_t::stringIndex

3.2.16 struct usb_device_get_hid_descriptor_struct_t

Data Fields

- uint8_t * [buffer](#)
Pass the buffer address.
- uint32_t [length](#)
Pass the buffer length.
- uint8_t [interfaceNumber](#)
The interface number.

3.2.16.0.0.9 Field Documentation**3.2.16.0.0.9.1** `uint8_t* usb_device_get_hid_descriptor_struct_t::buffer`**3.2.16.0.0.9.2** `uint32_t usb_device_get_hid_descriptor_struct_t::length`**3.2.16.0.0.9.3** `uint8_t usb_device_get_hid_descriptor_struct_t::interfaceNumber`**3.2.17 struct usb_device_get_hid_report_descriptor_struct_t****Data Fields**

- `uint8_t * buffer`
Pass the buffer address.
- `uint32_t length`
Pass the buffer length.
- `uint8_t interfaceNumber`
The interface number.

3.2.17.0.0.10 Field Documentation**3.2.17.0.0.10.1** `uint8_t* usb_device_get_hid_report_descriptor_struct_t::buffer`**3.2.17.0.0.10.2** `uint32_t usb_device_get_hid_report_descriptor_struct_t::length`**3.2.17.0.0.10.3** `uint8_t usb_device_get_hid_report_descriptor_struct_t::interfaceNumber`**3.2.18 struct usb_device_get_hid_physical_descriptor_struct_t****Data Fields**

- `uint8_t * buffer`
Pass the buffer address.
- `uint32_t length`
Pass the buffer length.
- `uint8_t index`
Physical index.
- `uint8_t interfaceNumber`
The interface number.

Data Structure Documentation

3.2.18.0.0.11 Field Documentation

3.2.18.0.0.11.1 `uint8_t* usb_device_get_hid_physical_descriptor_struct_t::buffer`

3.2.18.0.0.11.2 `uint32_t usb_device_get_hid_physical_descriptor_struct_t::length`

3.2.18.0.0.11.3 `uint8_t usb_device_get_hid_physical_descriptor_struct_t::interfaceNumber`

3.2.19 `union usb_device_get_descriptor_common_union_t`

Data Fields

- `usb_device_get_descriptor_common_struct_t commonDescriptor`
Common structure.
- `usb_device_get_device_descriptor_struct_t deviceDescriptor`
The structure to get device descriptor.
- `usb_device_get_device_qualifier_descriptor_struct_t deviceQualifierDescriptor`
The structure to get device qualifier descriptor.
- `usb_device_get_configuration_descriptor_struct_t configurationDescriptor`
The structure to get configuration descriptor.
- `usb_device_get_string_descriptor_struct_t stringDescriptor`
The structure to get string descriptor.
- `usb_device_get_hid_descriptor_struct_t hidDescriptor`
The structure to get HID descriptor.
- `usb_device_get_hid_report_descriptor_struct_t hidReportDescriptor`
The structure to get HID report descriptor.
- `usb_device_get_hid_physical_descriptor_struct_t hidPhysicalDescriptor`
The structure to get HID physical descriptor.

3.2.19.0.0.12 Field Documentation

- 3.2.19.0.0.12.1 `usb_device_get_descriptor_common_struct_t` `usb_device_get_descriptor_common_union_t::commonDescriptor`
- 3.2.19.0.0.12.2 `usb_device_get_device_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::deviceDescriptor`
- 3.2.19.0.0.12.3 `usb_device_get_device_qualifier_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::deviceQualifierDescriptor`
- 3.2.19.0.0.12.4 `usb_device_get_configuration_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::configurationDescriptor`
- 3.2.19.0.0.12.5 `usb_device_get_string_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::stringDescriptor`
- 3.2.19.0.0.12.6 `usb_device_get_hid_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::hidDescriptor`
- 3.2.19.0.0.12.7 `usb_device_get_hid_report_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::hidReportDescriptor`
- 3.2.19.0.0.12.8 `usb_device_get_hid_physical_descriptor_struct_t` `usb_device_get_descriptor_common_union_t::hidPhysicalDescriptor`

3.2.20 `struct usb_device_class_map_t`

Data Fields

- [usb_device_class_init_call_t](#) `classInit`
Class driver initialization- entry of the class driver.
- [usb_device_class_deinit_call_t](#) `classDeinit`
Class driver de-initialization.
- [usb_device_class_event_callback_t](#) `classEventCallback`
Class driver event callback.
- [usb_device_class_type_t](#) `type`
Class type.

3.2.21 `struct usb_device_common_class_struct_t`

Data Fields

- [usb_device_handle](#) `handle`
USB device handle.
- [usb_device_class_config_list_struct_t](#) * `configList`
USB device configure list.

Function Documentation

- uint8_t * [setupBuffer](#)
Setup packet data buffer.
- uint16_t [standardTranscationBuffer](#)
This variable is used in: get status request get configuration request get interface request set interface request get sync frame request.
- uint8_t [controllerId](#)
Controller ID.

3.3 Enumeration Type Documentation

3.3.1 enum usb_device_class_type_t

3.3.2 enum usb_device_class_event_t

3.4 Function Documentation

3.4.1 usb_status_t USB_DeviceClassInit (uint8_t *controllerId*, usb_device-_class_config_list_struct_t * *configList*, usb_device_handle * *handle*)

This function is used to initialize the common class and the supported classes.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>configList</i>	The class configurations. The pointer must point to the global variable. See the structure usb_device_class_config_list_struct_t .
out	<i>handle</i>	A parameter used to return pointer of the device handle to the caller. The value of the parameter is a pointer to the device handle. This design is used to make a simple device align with the composite device. For the composite device, there are many kinds of class handles. However, there is only one device handle. Therefore, the handle points to a device instead of a class. The class handle can be received from the usb_device-_class_config_struct_t::classHandle after the function successfully.

Returns

A USB error code or kStatus_USB_Success.

3.4.2 usb_status_t USB_DeviceClassDeinit (uint8_t *controllerId*)

This function is used to deinitialize the common class and the supported classes.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
----	---------------------	-----------------------------------------------------------------------------------------------

Returns

A USB error code or kStatus_USB_Success.

3.4.3 **usb_status_t USB_DeviceClassGetSpeed (uint8_t *controllerId*, uint8_t * *speed*)**

This function is used to get the USB bus speed.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
out	<i>speed</i>	It is an OUT parameter, which returns the current speed of the controller.

Returns

A USB error code or kStatus_USB_Success.

3.4.4 **usb_status_t USB_DeviceClassEvent (usb_device_handle *handle*, usb_device_class_event_t *event*, void * *param*)**

This function handles the event passed to the class drivers.

Parameters

in	<i>handle</i>	The device handle received from the USB_DeviceInit .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or kStatus_USB_Success.

Function Documentation

Return values

<i>kStatus_USB_Success</i>	A valid request has been handled.
<i>kStatus_USB_Invalid-Parameter</i>	The device handle not be found.
<i>kStatus_USB_Invalid-Request</i>	The request is invalid, and the control pipe is stalled by the caller.

3.4.5 **usb_status_t USB_DeviceClassCallback (usb_device_handle *handle*, uint32_t *event*, void * *param*)**

This function handles the common class callback.

Parameters

in	<i>handle</i>	The device handle received from the USB_DeviceInit .
in	<i>event</i>	The event codes. See the enumeration usb_device_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or `kStatus_USB_Success`.

3.4.6 **usb_status_t USB_DeviceClassGetDeviceHandle (uint8_t *controllerId*, usb_device_handle * *handle*)**

This function gets the device handle according to the controller ID.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
out	<i>handle</i>	An out parameter used to return the pointer of the device handle to the caller.

Return values

<i>kStatus_USB_Success</i>	Get device handle successfully.
<i>kStatus_USB_Invalid-Parameter</i>	The device handle can't be found.

USB MSC Class driver

3.5 USB MSC Class driver

3.5.1 Overview

The USB mass storage device class defines the protocols for file transfers between the host and the device. The MCUXpresso SDK USB stack provides support for MSC class driver which implements the bulk only transport specification and the UFI command specification.

Modules

- [USB MSC UFI driver](#)
- [USB MSC driver](#)

3.5.2 USB MSC driver

3.5.2.1 Overview

Data Structures

- struct `usb_device_msc_cbw_t`
Command Block Wrapper(CBW) [More...](#)
- struct `usb_device_msc_csw_t`
Command Status Wrapper(CSW) [More...](#)
- struct `usb_lba_transfer_information_struct_t`
Read/write information. [More...](#)
- struct `usb_device_lba_information_struct_t`
device information [More...](#)
- struct `usb_device_lba_app_struct_t`
Data transfer information. [More...](#)
- struct `usb_device_ufi_app_struct_t`
command and Data transfer information for UFI command [More...](#)
- struct `usb_device_msc_thirteen_case_struct_t`
The thirteen possible use cases of host expectations and device intent in the absence of overriding error conditions. [More...](#)
- struct `usb_device_msc_ufi_struct_t`
The MSC device UFI command status structure. [More...](#)
- struct `usb_device_msc_struct_t`
The MSC device structure. [More...](#)

Macros

- #define `USB_DEVICE_CONFIG_MSC_SUPPORT_DISK_LOCKING_MECHANISM` (0U)
prevent media removal flag
- #define `USB_DEVICE_CONFIG_MSC_CLASS_CODE` (0x08U)
The class code of the MSC class.
- #define `USB_DEVICE_MSC_BULK_ONLY_MASS_STORAGE_RESET` (0xFFU)
Bulk-Only Mass Storage Reset (class-specific request)
- #define `USB_DEVICE_MSC_GET_MAX_LUN` (0xFEU)
Get Maximum LUN (class-specific request)
- #define `USB_DEVICE_MSC_DCBWSIGNATURE` USB_LONG_TO_BIG_ENDIAN(0x55534243-U)
CBW dCBWSignature.
- #define `USB_DEVICE_MSC_DCSWSIGNATURE` USB_LONG_TO_BIG_ENDIAN(0x55534253-U)
CSW dCSSWSignature.
- #define `USB_DEVICE_MSC_CBW_DIRECTION_BIT` (0x80U)
CSW bmCBWFlags bit7.
- #define `USB_DEVICE_MSC_CBW_LENGTH` (31U)
CBW command length.
- #define `USB_DEVICE_MSC_CSW_LENGTH` (13U)
CSW command length.
- #define `USB_DEVICE_MSC_COMMAND_PASSED` (0x00U)

USB MSC Class driver

Command Block Status Values.

- #define [USB_DEVICE_MSC_INQUIRY_COMMAND](#) (0x12U)
UFI Commands code.

Enumerations

- enum [usb_device_msc_stall_type](#) {
 [USB_DEVICE_MSC_STALL_IN_CBW](#) = 1U,
 [USB_DEVICE_MSC_STALL_IN_DATA](#),
 [USB_DEVICE_MSC_STALL_IN_CSW](#) }
stall flag
- enum [USB_DeviceMscEvent_t](#) {
 [kUSB_DeviceMscEventReadResponse](#) = 0x01U,
 [kUSB_DeviceMscEventWriteResponse](#),
 [kUSB_DeviceMscEventWriteRequest](#),
 [kUSB_DeviceMscEventReadRequest](#),
 [kUSB_DeviceMscEventGetLbaInformation](#),
 [kUSB_DeviceMscEventFormatComplete](#),
 [kUSB_DeviceMscEventTestUnitReady](#),
 [kUSB_DeviceMscEventInquiry](#),
 [kUSB_DeviceMscEventModeSense](#),
 [kUSB_DeviceMscEventModeSelect](#),
 [kUSB_DeviceMscEventModeSelectResponse](#),
 [kUSB_DeviceMscEventRemovalRequest](#),
 [kUSB_DeviceMscEventSendDiagnostic](#),
 [kUSB_DeviceMscEventStopEjectMedia](#) }
Available common EVENT types in MSC class callback.

USB device MSC class APIs

- [usb_status_t](#) [USB_DeviceMscInit](#) ([uint8_t](#) controllerId, [usb_device_class_config_struct_t](#) *config, [class_handle_t](#) *handle)
Initializes the MSC class.
- [usb_status_t](#) [USB_DeviceMscDeinit](#) ([class_handle_t](#) handle)
Deinitializes the device MSC class.

3.5.2.2 Data Structure Documentation

3.5.2.2.1 struct [usb_device_msc_cbw_t](#)

Data Fields

- [uint32_t](#) [signature](#)
Byte 0-3 dCBWSignature.
- [uint32_t](#) [tag](#)

- *Byte 4-7 dCBWTag.*
uint32_t [dataTransferLength](#)
- *Byte 8-11 dCBWDataTransferLength.*
uint8_t [flags](#)
- *Byte 12 bmCBWFlags.*
uint8_t [logicalUnitNumber](#)
- *Byte 13 bCBWLUN.*
uint8_t [cbLength](#)
- *Byte 14 bCBWCBLength.*
uint8_t [cbwcb](#) [16]
Byte 15-30 CBWCB, CBWCB is used to store UFI command.

3.5.2.2.2 struct usb_device_msc_csw_t

Data Fields

- uint32_t [signature](#)
Byte 0-3 dCSWSignature.
- uint32_t [tag](#)
Byte 4-7 dCSWTag.
- uint32_t [dataResidue](#)
Byte 8-11 dCSWDataResidue.
- uint8_t [cswStatus](#)
Byte 12 bCSWStatus.

3.5.2.2.3 struct usb_lba_transfer_information_struct_t

Data Fields

- uint32_t [startingLogicalBlockAddress](#)
The logical block at which the read/write operation shall begin.
- uint32_t [transferNumber](#)
The number of contiguous logical blocks of data that shall be transferred.

3.5.2.2.4 struct usb_device_lba_information_struct_t

Data Fields

- uint32_t [totalLbaNumberSupports](#)
Total blocks number supported.
- uint32_t [lengthOfEachLba](#)
Length of each block.
- uint32_t [bulkInBufferSize](#)
Bulk in buffer size.
- uint32_t [bulkOutBufferSize](#)
Bulk out buffer size.
- uint8_t [logicalUnitNumberSupported](#)
Number of LUN.

USB MSC Class driver

3.5.2.2.5 struct usb_device_lba_app_struct_t

Data Fields

- uint32_t [offset](#)
Offset of the block need to access.
- uint32_t [size](#)
Size of the transferred data.
- uint8_t * [buffer](#)
Buffer address of the transferred data.

3.5.2.2.6 struct usb_device_ufi_app_struct_t

Data Fields

- uint8_t * [cbwcb](#)
current ufi command block stored in the CBW
- uint32_t [size](#)
Size of the transferred data if command has data flow.
- uint8_t * [buffer](#)
Buffer address of the transferred data if command has data flow.
- [usb_device_request_sense_data_struct_t](#) * [requestSense](#)
sense data for the current command

3.5.2.2.7 struct usb_device_msc_thirteen_case_struct_t

Data Fields

- uint32_t [hostExpectedDataLength](#)
The number of bytes of data that the host expects to transfer.
- uint32_t [deviceExpectedDataLength](#)
The number of bytes of data that the device expects to transfer.
- uint8_t * [buffer](#)
Data buffer.
- [usb_lba_transfer_information_struct_t](#) [lbaInformation](#)
Read/write information.
- uint8_t [lbaSendRecvSelect](#)
Whether the command is read or write command.
- uint8_t [hostExpectedDirection](#)
Host expected data direction.
- uint8_t [deviceExpectedDirection](#)
Device expected data direction.

3.5.2.2.8 struct usb_device_msc_ufi_struct_t

Data Fields

- [usb_device_request_sense_data_struct_t](#) * [requestSense](#)

- *Request Sense Standard Data.*
- `usb_device_msc_thirteen_case_struct_t` `thirteenCase`
Thirteen possible cases.
- `usb_device_read_capacity_struct_t` * `readCapacity`
READ CAPACITY Data.
- `usb_device_read_capacity16_data_struct_t` * `readCapacity16`
READ CAPACITY Data.
- `usb_device_mode_parameters_header_struct_t` `ModeParametersHeader`
Mode Parameter Header.
- `uint8_t` `formattedDisk`
**Formatted or unformatted media*
- `uint8_t` * `formatCapacityData`
Capacity List.

3.5.2.2.9 struct usb_device_msc_struct_t

Data Fields

- `usb_device_handle` `handle`
The device handle.
- `usb_device_class_config_struct_t` * `configurationStruct`
The configuration of the class.
- `usb_device_interface_struct_t` * `interfaceHandle`
Current interface handle.
- `uint32_t` `transferRemaining`
Transfer remaining data.
- `uint32_t` `currentOffset`
Current address offset.
- `uint32_t` `totalLogicalBlockNumber`
Total logical block number of device.
- `uint32_t` `lengthOfEachLba`
Length of logical block.
- `uint32_t` `implementingDiskDrive`
Disk drive.
- `uint32_t` `bulkInBufferSize`
Bulk in buffer size.
- `uint32_t` `bulkOutBufferSize`
Bulk out buffer size.
- `usb_device_msc_cbw_t` * `mscCbw`
CBW structure.
- `usb_device_msc_csw_t` * `mscCsw`
CSW structure.
- `usb_device_msc_ufi_struct_t` `mscUfi`
UFI command information structure.
- `uint8_t` `dataOutFlag`
CBW indicating bulk out transfer, clear this flag when data transfer done.
- `uint8_t` `dataInFlag`
CBW indicating bulk in transfer, clear this flag when data transfer done.
- `uint8_t` `inEndpointStallFlag`
In endpoint stall flag.

USB MSC Class driver

- `uint8_t outEndpointStallFlag`
Out endpoint stall flag.
- `uint8_t cbwValidFlag`
The CBW was received after the device had sent a CSW or after a reset ,or else it is invalid.
- `uint8_t performResetRecover`
Device need reset command from host.
- `uint8_t performResetDoneFlag`
Device has perform reset command.
- `uint8_t needInStallFlag`
In endpoint should be stalled.
- `uint8_t needOutStallFlag`
Out endpoint should be stalled.
- `uint8_t cbwPrimeFlag`
CBW prime flag, prime means device MSC has been ready to receive CBW, the bulk out endpoint has got the prepared buffer.
- `uint8_t cswPrimeFlag`
CSW prime flag, prime means device MSC has been ready to receive CSW, the bulk in endpoint has got the prepared buffer.
- `uint8_t stallStatus`
Stall status.
- `uint8_t logicalUnitNumber`
Supported logical units number of device.
- `uint8_t bulkInEndpoint`
Bulk in endpoint number.
- `uint8_t bulkOutEndpoint`
Bulk out endpoint number.
- `uint8_t alternate`
Current alternate setting of the interface.
- `uint8_t configuration`
Current configuration.
- `uint8_t interfaceNumber`
The interface number of the class.
- `uint8_t inEndpointCswCancelFlag`
the state when calcel function happens, and need send the csw after cancel

3.5.2.2.9.1 Field Documentation

3.5.2.2.9.1.1 `uint8_t usb_device_msc_struct_t::logicalUnitNumber`

See bulk only specification 3.2 Get Maximum LUN (class-specific request)

3.5.2.3 Enumeration Type Documentation

3.5.2.3.1 `enum usb_device_msc_stall_type`

Enumerator

`USB_DEVICE_MSC_STALL_IN_CBW` Stall in CBW.

`USB_DEVICE_MSC_STALL_IN_DATA` Stall in data transfer.

`USB_DEVICE_MSC_STALL_IN_CSW` Stall in CSW.

3.5.2.3.2 enum USB_DeviceMscEvent_t

Enumerator

kUSB_DeviceMscEventReadResponse host has already read the whole data from device
kUSB_DeviceMscEventWriteResponse devcie has already received the data from host.
kUSB_DeviceMscEventWriteRequest Host want to write data to device through write command, devcie need prepare one buffer to store the data from host.
kUSB_DeviceMscEventReadRequest Host want to read data from device through read command, device need prepare one buffer containing data pending for transfer.
kUSB_DeviceMscEventGetLbaInformation Get device information.
kUSB_DeviceMscEventFormatComplete Format complete.
kUSB_DeviceMscEventTestUnitReady Test Unit Ready command.
kUSB_DeviceMscEventInquiry Inquiry Command command.
kUSB_DeviceMscEventModeSense mode sense command
kUSB_DeviceMscEventModeSelect mode select command, prepare data buffer and buffer length to store data for mode select
kUSB_DeviceMscEventModeSelectResponse got data of mode select command
kUSB_DeviceMscEventRemovalRequest Prevent_allow_medium_command.
kUSB_DeviceMscEventSendDiagnostic Send Diagnostic command.
kUSB_DeviceMscEventStopEjectMedia Start_stop_unit_command.

3.5.2.4 Function Documentation

3.5.2.4.1 usb_status_t USB_DeviceMscInit (uint8_t *controllerId*, usb_device_class_config_struct_t * *config*, class_handle_t * *handle*)

This function is used to initialize the MSC class.

Parameters

<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t.
<i>config</i>	The class configuration information.
<i>handle</i>	A parameter used to return pointer of the MSC class handle to the caller.

Returns

A USB error code or kStatus_USB_Success.

3.5.2.4.2 usb_status_t USB_DeviceMscDeinit (class_handle_t *handle*)

The function deinitializes the device MSC class.

USB MSC Class driver

Parameters

<i>handle</i>	The MSC class handle received from usb_device_class_config_struct_t::classHandle .
---------------	----------------------------------------------------------------------------------------------------

Returns

A USB error code or kStatus_USB_Success.

3.5.3 USB MSC UFI driver

3.5.3.1 Overview

Data Structures

- struct [usb_device_inquiry_command_struct_t](#)
UFI inquiry command structure. [More...](#)
- struct [usb_device_request_sense_command_struct_t](#)
UFI request sense command structure. [More...](#)
- struct [usb_device_read_format_capacities_command_struct_t](#)
UFI read format capacities command structure. [More...](#)
- struct [usb_device_read_capacities_command_struct_t](#)
UFI read capacities command structure. [More...](#)
- struct [usb_device_read_write_10_command_struct_t](#)
UFI read write 10 structure. [More...](#)
- struct [usb_device_inquiry_data_format_struct_t](#)
UFI inquiry data format structure. [More...](#)
- struct [usb_device_request_sense_data_struct_t](#)
UFI request sense data structure. [More...](#)
- struct [usb_device_read_capacity_struct_t](#)
UFI read capacity data structure. [More...](#)
- struct [usb_device_read_capacity16_data_struct_t](#)
UFI read capacity data structure. [More...](#)
- struct [usb_device_capacity_list_header_struct_t](#)
UFI capacity list header structure. [More...](#)
- struct [usb_device_current_max_capacity_descriptor_struct_t](#)
UFI current maximum capacity structure. [More...](#)
- struct [usb_device_formattable_capacity_descriptor_struct_t](#)
UFI formatting capacity structure. [More...](#)
- struct [usb_device_mode_parameters_header_struct_t](#)
UFI mode parameters header structure. [More...](#)
- struct [usb_device_format_capacity_response_data_struct_t](#)
UFI Capacity List structure. [More...](#)

Macros

- #define [USB_DEVICE_MSC_UFI_NO_SENSE](#) 0x00U
Indicates that there is no specific sense key information to be reported.
- #define [USB_DEVICE_MSC_UFI_RECOVERED_ERROR](#) 0x01U
Indicates that the last command completed successfully with some recovery action performed by the UFI device.
- #define [USB_DEVICE_MSC_UFI_NOT_READY](#) 0x02U
Indicates that the UFI device cannot be accessed.
- #define [USB_DEVICE_MSC_UFI_MEDIUM_ERROR](#) 0x03U
Indicates that the command terminated with a non-recovered error condition that was probably caused by a flaw in the medium or an error in the recorded data.
- #define [USB_DEVICE_MSC_UFI_HARDWARE_ERROR](#) 0x04U
Indicates that the UFI device detected a non-recoverable hardware failure while performing the command

USB MSC Class driver

- or during a self test.*
- #define **USB_DEVICE_MSC_UFI_ILLEGAL_REQUEST** 0x05U
Indicates that there was an illegal parameter in the Command Packet or in the additional parameters supplied as data for some commands.
- #define **USB_DEVICE_MSC_UFI_UNIT_ATTENTION** 0x06U
Indicates that the removable medium may have been changed or the UFI device has been reset.
- #define **USB_DEVICE_MSC_UFI_DATA_PROTECT** 0x07U
Indicates that a command that writes the medium was attempted on a block that is protected from this operation.
- #define **USB_DEVICE_MSC_UFI_BLANK_CHECK** 0x08U
Indicates that a write-once device or a sequential-access device encountered blank medium or format-defined end-of-data indication while reading or a write-once device encountered a non-blank medium while writing.
- #define **USB_DEVICE_MSC_UFI_VENDOR_SPECIFIC_ERROR** 0x09U
This sense key is available for reporting vendor-specific conditions.
- #define **USB_DEVICE_MSC_UFI_ABORTED_COMMAND** 0x0BU
Indicates that the UFI device has aborted the command The host may be able to recover by trying the command again.
- #define **USB_DEVICE_MSC_UFI_VOLUME_OVERFLOW** 0x0DU
Indicates that a buffered peripheral device has reached the end-of-partition and data may remain in the buffer that has not been written to the medium.
- #define **USB_DEVICE_MSC_UFI_MISCOMPARE** 0x0EU
Indicates that the source data did not match the data read from the medium.
- #define **USB_DEVICE_MSC_UFI_INVALID_COMMAND_OPCODE** 0x20U
Invalid command operation code.
- #define **USB_DEVICE_MSC_UFI_WRITE_FAULT** 0x03U
Write fault.
- #define **USB_DEVICE_MSC_UFI_UNRECOVERED_READ_ERROR** 0x11U
Not recovered read error.
- #define **USB_DEVICE_MSC_UFI_UNKNOWN_ERROR** 0xFFU
Unknown error.
- #define **USB_DEVICE_MSC_UFI_INVALID_FIELD_IN_COMMAND_PKT** 0x24U
Invalid field in command packet.
- #define **USB_DEVICE_MSC_UFI_LBA_OUT_OF_RANGE** 0x21U
Invalid logical block address out of range.
- #define **USB_DEVICE_MSC_UFI_REQ_SENSE_VALID_ERROR_CODE** 0x70U
Valid error code, 70h indicate current errors.
- #define **USB_DEVICE_MSC_UFI_REQ_SENSE_ADDITIONAL_SENSE_LEN** 0x0AU
The UFI device sets the value of this field to ten, to indicate that ten more bytes of sense data follow this field.
- #define **USB_DEVICE_MSC_UFI_PREVENT_ALLOW_REMOVAL_MASK** 0x01U
Prevent media removal flag.
- #define **USB_DEVICE_MSC_UFI_LOAD_EJECT_START_MASK** 0x03U
LoEj Start flag.
- #define **USB_DEVICE_MSC_UFI_FORMATTED_MEDIA** 0x02U
Formatted Media - Current media capacity.
- #define **USB_DEVICE_MSC_UFI_UNFORMATTED_MEDIA** 0x01U
Unformatted Media - Maximum formatting capacity for this cartridge.
- #define **USB_DEVICE_MSC_UFI_NO_CARTRIDGE_IN_DRIVE** 0x03U
No Cartridge in Drive - Maximum forming capacity for any cartridge.
- #define **USB_DEVICE_MSC_UFI_INQUIRY_ALLOCATION_LENGTH** 0x24U

- *INQUIRY Data length of INQUIRY Command.*
- #define `USB_DEVICE_MSC_UFI_REQ_SENSE_DATA_LENGTH` 18U
- *Request Sense Data length of REQUEST SENSE Command.*
- #define `USB_DEVICE_MSC_UFI_READ_CAPACITY_DATA_LENGTH` 0x08U
- *READ CAPACITY Data length of READ CAPACITY Command.*
- #define `USB_DEVICE_MSC_UFI_READ_CAPACITY16_DATA_LENGTH` 0x0CU
- *READ CAPACITY Data length of READ CAPACITY Command.*
- #define `USB_DEVICE_MSC_UFI_PERIPHERAL_QUALIFIER` 0U
- *Reserved.*
- #define `USB_DEVICE_MSC_UFI_PERIPHERAL_QUALIFIER_SHIFT` 5U
- *Peripheral Device Type shift.*
- #define `USB_DEVICE_MSC_UFI_VERSIONS` 4U
- *Version value.*
- #define `USB_DEVICE_MSC_UFI_PERIPHERAL_DEVICE_TYPE` 0x00U
- *Peripheral Device Type value of INQUIRY Data.*
- #define `USB_DEVICE_MSC_UFI_REMOVABLE_MEDIUM_BIT` 1U
- *Removable Media Bit value, this shall be set to one to indicate removable media.*
- #define `USB_DEVICE_MSC_UFI_REMOVABLE_MEDIUM_BIT_SHIFT` 7U
- *Removable Media Bit shift.*
- #define `USB_DEVICE_MSC_UFI_ADDITIONAL_LENGTH` 0x20U
- *Additional Length.*

3.5.3.2 Data Structure Documentation

3.5.3.2.1 struct usb_device_inquiry_command_struct_t

Data Fields

- uint8_t `operationCode`
Operation Code.
- uint8_t `logicalUnitNumber`
Specifies the logical unit (0~7) for which Inquiry data should be returned.
- uint8_t `pageCode`
Page Code.
- uint8_t `reserved`
Reserved.
- uint8_t `allocationLength`
Specifies the maximum number of bytes of inquiry data to be returned.
- uint8_t `reserved1` [7]
Reserved.

3.5.3.2.2 struct usb_device_request_sense_command_struct_t

Data Fields

- uint8_t `operationCode`
Operation Code.
- uint8_t `logicalUnitNumber`
Logical Unit Number.

USB MSC Class driver

- uint8_t [reserved](#) [2]
reserved
- uint8_t [allocationLength](#)
Allocation Length.
- uint8_t [reserved1](#) [7]
reserved

3.5.3.2.3 struct usb_device_read_format_capacities_command_struct_t

Data Fields

- uint8_t [operationCode](#)
Operation Code.
- uint8_t [logicalUnitNumber](#)
Logical Unit Number.
- uint8_t [reserved](#) [5]
reserved
- uint16_t [allocationLength](#)
Allocation Length.
- uint8_t [reserved1](#) [3]
reserved

3.5.3.2.4 struct usb_device_read_capacities_command_struct_t

Data Fields

- uint8_t [operationCode](#)
Operation Code.
- uint8_t [logicalUnitNumber](#)
Logical Unit Number.
- uint32_t [lba](#)
Logical Block Address.
- uint8_t [reserved](#) [2]
Reserved.
- uint8_t [pmi](#)
This bit should be set to zero for UFI.
- uint8_t [reserved1](#) [3]
Reserved.

3.5.3.2.5 struct usb_device_read_write_10_command_struct_t

Data Fields

- uint8_t [operationCode](#)
Operation Code.
- uint8_t [lunDpoFuaReladr](#)
Logical Unit Number DPO FUA RelAdr.
- uint32_t [lba](#)

- *Logical Block Address.*
- uint8_t **reserved**
Reserved.
- uint8_t **transferLengthMsb**
Transfer Length (MSB)
- uint8_t **transferLengthLsb**
Transfer Length (LSB)
- uint8_t **reserved1** [3]
Reserved.

3.5.3.2.6 struct usb_device_inquiry_data_format_struct_t

Data Fields

- uint8_t **peripheralDeviceType**
Peripheral Device Type.
- uint8_t **rmb**
Removable Media Bit.
- uint8_t **versions**
ISO Version, ECMA Version, ANSI Version.
- uint8_t **responseDataFormat**
Response Data Format.
- uint8_t **additionalLength**
The Additional Length field shall specify the length in bytes of the parameters.
- uint8_t **reserved** [3]
reserved
- uint8_t **vendorInformatin** [8]
Vendor Identification.
- uint8_t **productId** [16]
Product Identification.
- uint8_t **productVersionLevel** [4]
Product Revision Level.

3.5.3.2.7 struct usb_device_request_sense_data_struct_t

Data Fields

- uint8_t **validErrorCode**
Error Code.
- uint8_t **reserved**
reserved
- uint8_t **senseKey**
Sense Key.
- uint8_t **information** [4]
Information.
- uint8_t **additionalSenseLength**
Additional Sense Length.
- uint8_t **reserved1** [4]
reserved

USB MSC Class driver

- uint8_t [additionalSenseCode](#)
Additional Sense Code.
- uint8_t [additionalSenseQualifier](#)
Additional Sense Code Qualifier.
- uint8_t [reserved2](#) [4]
reserved

3.5.3.2.8 struct usb_device_read_capacity_struct_t

Data Fields

- uint32_t [lastLogicalBlockAddress](#)
Last Logical Block Address.
- uint32_t [blockSize](#)
Block Length In Bytes.

3.5.3.2.9 struct usb_device_read_capacity16_data_struct_t

Data Fields

- uint32_t [lastLogicalBlockAddress0](#)
Last Logical Block Address.
- uint32_t [lastLogicalBlockAddress1](#)
Last Logical Block Address.
- uint32_t [blockSize](#)
Block Length In Bytes.

3.5.3.2.10 struct usb_device_capacity_list_header_struct_t

Data Fields

- uint8_t [reserverd](#) [3]
reserved
- uint8_t [capacityListLength](#)
Capacity List Length.

3.5.3.2.11 struct usb_device_current_max_capacity_descriptor_struct_t

Data Fields

- uint32_t [blockNumber](#)
Number of Blocks.
- uint32_t [descriptorCodeBlockLength](#)
Byte 4 Descriptor Code , byte 5-7 Block Length.

3.5.3.2.12 struct usb_device_formattable_capacity_descriptor_struct_t**Data Fields**

- uint32_t **blockNumber**
Number of Blocks.
- uint32_t **blockLength**
Block Length.

3.5.3.2.13 struct usb_device_mode_parameters_header_struct_t**Data Fields**

- uint16_t **modeDataLength**
Mode Data Length.
- uint8_t **mediumTypeCode**
The Medium Type Code field specifies the inserted medium type.
- uint8_t **wpDpfua**
WP and DPOFUA bit.
- uint8_t **reserved** [4]
Reserved.

3.5.3.2.14 struct usb_device_format_capacity_response_data_struct_t**Data Fields**

- uint8_t **capacityListHead** [sizeof(usb_device_capacity_list_header_struct_t)]
Capacity List Header.
- uint8_t **currentMaxCapacityDescriptor** [sizeof(usb_device_current_max_capacity_descriptor_struct_t)]
Current/Maximum Capacity Header.
- uint8_t **formattableCapacityDescriptor** [sizeof(usb_device_formattable_capacity_descriptor_struct_t)*3]
Formatting Capacity Descriptor.

3.6 USB CDC Class driver

3.6.1 Overview

The USB communications device class (or USB CDC) is a composite Universal Serial Bus device class. The class may include more than one interface, such as a custom control interface, data interface, audio, or mass storage-related interfaces. The MCUXpresso SDK USB stack provides support for CDC ACM, which is defined in CDC PSTN Subclass. In addition, the Microsoft[®] RNDIS is also implemented upon the CDC ACM driver.

Modules

- [USB CDC ACM Class driver](#)
- [USB CDC RNDIS driver](#)

3.6.2 USB CDC ACM Class driver

3.6.2.1 Overview

This section describes the programming interface of the USB CDC ACM class driver. The USB CDC ACM class driver handles the specific control requests for CDC ACM, transfers data packets to and from the host through the bulk pipe, as well as provides notification to host through the interrupt pipe.

3.6.2.2 USB CDC ACM Device structures

The driver uses an instantiation of the `usb_device_cdc_acm_struct_t` structure to maintain the current state of a particular USB CDC ACM instance module driver. This structure holds the USB device handle and keeps track of the configuration value, alternate setting, pipes and interfaces that are enumerated for this USB ACM device.

The USB CDC ACM class driver populates the structure members.

3.6.2.3 USB CDC ACM Initialization

The `usb_device_cdc_acm_init` is called from `usb_device_class_init` when it matches the class type of CDC with the one in configure structure passed from application. In this function it associates the configure structure with the USB CDC ACM device, resets the configuration value and creates mutex for each pipe.

3.6.2.4 USB CDC ACM Endpoint Initialization

After the enumeration procedure is done, all the endpoints, other than the control endpoint, are initialized with their own attributes, for example, endpoint address, transfer type and maximum packet size. Most of the attributes can be drawn from the configure structure. Each endpoint is assigned a callback function to serve the corresponding event.

3.6.2.5 USB CDC ACM Event Handling

The `usb_device_cdc_acm_event` is called from `usb_device_class_event` when there occurs a class-specific event and it matches the class type of CDC with the one in configure structure. For some events which need to notify the application, the callback function defined in application is invoked with the dedicated event type.

3.6.2.6 USB CDC ACM Send data

The `usb_device_cdc_acm_send` is called to send packet to host through the bulk pipe. Users need to specify the USB CDC ACM class handle, the endpoint address, the buffer address and the length of the

USB CDC Class driver

buffer to prime a sending transfer. Note that the transfer is initiated by the host so this transfer is not accomplished until the `kUsbDeviceCdcEventSendResponse` event occurs.

It allows only one transfer at a time, so the call to `usb_device_cdc_acm_send` returns `kStatus_USB_Busy` if the previous transfer is not done yet.

3.6.2.7 USB CDC ACM Receive data

The `usb_device_cdc_acm_rcv` is called to receive packet from host through the bulk pipe. Users need to specify the USB CDC ACM class handle, the endpoint address, the buffer address and the length of the buffer to prime a receiving transfer. Note that the transfer is initiated by the host so this transfer is not accomplished until the `kUsbDeviceCdcEventRcvResponse` event occurs.

It allows only one transfer at a time, so the call to `usb_device_cdc_acm_send` returns `kStatus_USB_Busy` if the previous transfer is not done yet.

Data Structures

- struct `usb_device_cdc_acm_request_param_struct_t`
Definition of parameters for CDC ACM request. [More...](#)
- struct `usb_device_cdc_acm_pipe_t`
Definition of pipe structure. [More...](#)
- struct `usb_device_cdc_acm_struct_t`
Definition of structure for CDC ACM device. [More...](#)

Macros

- #define `USB_DEVICE_CONFIG_CDC_ACM_MAX_INSTANCE` (1)
The maximum number of CDC device instance.
- #define `USB_DEVICE_CONFIG_CDC_COMM_CLASS_CODE` (0x02)
The CDC communication class code.
- #define `USB_DEVICE_CONFIG_CDC_DATA_CLASS_CODE` (0x0A)
The CDC data class code.
- #define `USB_DEVICE_CDC_REQUEST_SEND_ENCAPSULATED_COMMAND` (0x00)
The CDC class request code for `SEND_ENCAPSULATED_COMMAND`.
- #define `USB_DEVICE_CDC_REQUEST_GET_ENCAPSULATED_RESPONSE` (0x01)
The CDC class request code for `GET_ENCAPSULATED_RESPONSE`.
- #define `USB_DEVICE_CDC_REQUEST_SET_COMM_FEATURE` (0x02)
The CDC class request code for `SET_COMM_FEATURE`.
- #define `USB_DEVICE_CDC_REQUEST_GET_COMM_FEATURE` (0x03)
The CDC class request code for `GET_COMM_FEATURE`.
- #define `USB_DEVICE_CDC_REQUEST_CLEAR_COMM_FEATURE` (0x04)
The CDC class request code for `CLEAR_COMM_FEATURE`.
- #define `USB_DEVICE_CDC_REQUEST_SET_AUX_LINE_STATE` (0x10)
The CDC class request code for `SET_AUX_LINE_STATE`.
- #define `USB_DEVICE_CDC_REQUEST_SET_HOOK_STATE` (0x11)
The CDC class request code for `SET_HOOK_STATE`.
- #define `USB_DEVICE_CDC_REQUEST_PULSE_SETUP` (0x12)

- The CDC class request code for PULSE_SETUP.*
- #define **USB_DEVICE_CDC_REQUEST_SEND_PULSE** (0x13)
- The CDC class request code for SEND_PULSE.*
- #define **USB_DEVICE_CDC_REQUEST_SET_PULSE_TIME** (0x14)
- The CDC class request code for SET_PULSE_TIME.*
- #define **USB_DEVICE_CDC_REQUEST_RING_AUX_JACK** (0x15)
- The CDC class request code for RING_AUX_JACK.*
- #define **USB_DEVICE_CDC_REQUEST_SET_LINE_CODING** (0x20)
- The CDC class request code for SET_LINE_CODING.*
- #define **USB_DEVICE_CDC_REQUEST_GET_LINE_CODING** (0x21)
- The CDC class request code for GET_LINE_CODING.*
- #define **USB_DEVICE_CDC_REQUEST_SET_CONTROL_LINE_STATE** (0x22)
- The CDC class request code for SET_CONTROL_LINE_STATE.*
- #define **USB_DEVICE_CDC_REQUEST_SEND_BREAK** (0x23)
- The CDC class request code for SEND_BREAK.*
- #define **USB_DEVICE_CDC_REQUEST_SET_RINGER_PARAMS** (0x30)
- The CDC class request code for SET_RINGER_PARAMS.*
- #define **USB_DEVICE_CDC_REQUEST_GET_RINGER_PARAMS** (0x31)
- The CDC class request code for GET_RINGER_PARAMS.*
- #define **USB_DEVICE_CDC_REQUEST_SET_OPERATION_PARAM** (0x32)
- The CDC class request code for SET_OPERATION_PARAM.*
- #define **USB_DEVICE_CDC_REQUEST_GET_OPERATION_PARAM** (0x33)
- The CDC class request code for GET_OPERATION_PARAM.*
- #define **USB_DEVICE_CDC_REQUEST_SET_LINE_PARAMS** (0x34)
- The CDC class request code for SET_LINE_PARAMS.*
- #define **USB_DEVICE_CDC_REQUEST_GET_LINE_PARAMS** (0x35)
- The CDC class request code for GET_LINE_PARAMS.*
- #define **USB_DEVICE_CDC_REQUEST_DIAL_DIGITS** (0x36)
- The CDC class request code for DIAL_DIGITS.*
- #define **USB_DEVICE_CDC_REQUEST_SET_UNIT_PARAMETER** (0x37)
- The CDC class request code for SET_UNIT_PARAMETER.*
- #define **USB_DEVICE_CDC_REQUEST_GET_UNIT_PARAMETER** (0x38)
- The CDC class request code for GET_UNIT_PARAMETER.*
- #define **USB_DEVICE_CDC_REQUEST_CLEAR_UNIT_PARAMETER** (0x39)
- The CDC class request code for CLEAR_UNIT_PARAMETER.*
- #define **USB_DEVICE_CDC_REQUEST_SET_ETHERNET_MULTICAST_FILTERS** (0x40)
- The CDC class request code for SET_ETHERNET_MULTICAST_FILTERS.*
- #define **USB_DEVICE_CDC_REQUEST_SET_ETHERNET_POW_PATTER_FILTER** (0x41)
- The CDC class request code for SET_ETHERNET_POW_PATTER_FILTER.*
- #define **USB_DEVICE_CDC_REQUEST_GET_ETHERNET_POW_PATTER_FILTER** (0x42)
- The CDC class request code for GET_ETHERNET_POW_PATTER_FILTER.*
- #define **USB_DEVICE_CDC_REQUEST_SET_ETHERNET_PACKET_FILTER** (0x43)
- The CDC class request code for SET_ETHERNET_PACKET_FILTER.*
- #define **USB_DEVICE_CDC_REQUEST_GET_ETHERNET_STATISTIC** (0x44)
- The CDC class request code for GET_ETHERNET_STATISTIC.*
- #define **USB_DEVICE_CDC_REQUEST_SET_ATM_DATA_FORMAT** (0x50)
- The CDC class request code for SET_ATM_DATA_FORMAT.*
- #define **USB_DEVICE_CDC_REQUEST_GET_ATM_DEVICE_STATISTICS** (0x51)
- The CDC class request code for GET_ATM_DEVICE_STATISTICS.*
- #define **USB_DEVICE_CDC_REQUEST_SET_ATM_DEFAULT_VC** (0x52)
- The CDC class request code for SET_ATM_DEFAULT_VC.*

USB CDC Class driver

- #define **USB_DEVICE_CDC_REQUEST_GET_ATM_VC_STATISTICS** (0x53)
The CDC class request code for GET_ATM_VC_STATISTICS.
- #define **USB_DEVICE_CDC_REQUEST_MDLM_SPECIFIC_REQUESTS_MASK** (0x7F)
The CDC class request code for MDLM_SPECIFIC_REQUESTS_MASK.
- #define **USB_DEVICE_CDC_NOTIF_NETWORK_CONNECTION** (0x00)
The CDC class notify code for NETWORK_CONNECTION.
- #define **USB_DEVICE_CDC_NOTIF_RESPONSE_AVAIL** (0x01)
The CDC class notify code for RESPONSE_AVAIL.
- #define **USB_DEVICE_CDC_NOTIF_AUX_JACK_HOOK_STATE** (0x08)
The CDC class notify code for AUX_JACK_HOOK_STATE.
- #define **USB_DEVICE_CDC_NOTIF_RING_DETECT** (0x09)
The CDC class notify code for RING_DETECT.
- #define **USB_DEVICE_CDC_NOTIF_SERIAL_STATE** (0x20)
The CDC class notify code for SERIAL_STATE.
- #define **USB_DEVICE_CDC_NOTIF_CALL_STATE_CHANGE** (0x28)
The CDC class notify code for CALL_STATE_CHANGE.
- #define **USB_DEVICE_CDC_NOTIF_LINE_STATE_CHANGE** (0x29)
The CDC class notify code for LINE_STATE_CHANGE.
- #define **USB_DEVICE_CDC_NOTIF_CONNECTION_SPEED_CHANGE** (0x2A)
The CDC class notify code for CONNECTION_SPEED_CHANGE.
- #define **USB_DEVICE_CDC_FEATURE_ABSTRACT_STATE** (0x01)
The CDC class feature select code for ABSTRACT_STATE.
- #define **USB_DEVICE_CDC_FEATURE_COUNTRY_SETTING** (0x02)
The CDC class feature select code for COUNTRY_SETTING.
- #define **USB_DEVICE_CDC_CONTROL_SIG_BITMAP_CARRIER_ACTIVATION** (0x02)
The CDC class control signal bitmap value for CARRIER_ACTIVATION.
- #define **USB_DEVICE_CDC_CONTROL_SIG_BITMAP_DTE_PRESENCE** (0x01)
The CDC class control signal bitmap value for DTE_PRESENCE.
- #define **USB_DEVICE_CDC_UART_STATE_RX_CARRIER** (0x01)
The UART state bitmap value of RX_CARRIER.
- #define **USB_DEVICE_CDC_UART_STATE_TX_CARRIER** (0x02)
The UART state bitmap value of TX_CARRIER.
- #define **USB_DEVICE_CDC_UART_STATE_BREAK** (0x04)
The UART state bitmap value of BREAK.
- #define **USB_DEVICE_CDC_UART_STATE_RING_SIGNAL** (0x08)
The UART state bitmap value of RING_SIGNAL.
- #define **USB_DEVICE_CDC_UART_STATE_FRAMING** (0x10)
The UART state bitmap value of FRAMING.
- #define **USB_DEVICE_CDC_UART_STATE_PARITY** (0x20)
The UART state bitmap value of PARITY.
- #define **USB_DEVICE_CDC_UART_STATE_OVERRUN** (0x40)
The UART state bitmap value of OVERRUN.

Enumerations

- enum `usb_device_cdc_acm_event_t` {
`kUSB_DeviceCdcEventSendResponse` = 0x01,
`kUSB_DeviceCdcEventRecvResponse`,
`kUSB_DeviceCdcEventSerialStateNotif`,
`kUSB_DeviceCdcEventSendEncapsulatedCommand`,
`kUSB_DeviceCdcEventGetEncapsulatedResponse`,
`kUSB_DeviceCdcEventSetCommFeature`,
`kUSB_DeviceCdcEventGetCommFeature`,
`kUSB_DeviceCdcEventClearCommFeature`,
`kUSB_DeviceCdcEventGetLineCoding`,
`kUSB_DeviceCdcEventSetLineCoding`,
`kUSB_DeviceCdcEventSetControlLineState`,
`kUSB_DeviceCdcEventSendBreak` }

Definition of CDC class event.

USB CDC ACM Class Driver

- `usb_status_t USB_DeviceCdcAcmInit` (`uint8_t` controllerId, `usb_device_class_config_struct_t` *config, `class_handle_t` *handle)
Initializes the USB CDC ACM class.
- `usb_status_t USB_DeviceCdcAcmDeinit` (`class_handle_t` handle)
Deinitializes the USB CDC ACM class.
- `usb_status_t USB_DeviceCdcAcmEvent` (`void` *handle, `uint32_t` event, `void` *param)
Handles the CDC ACM class event.
- `usb_status_t USB_DeviceCdcAcmSend` (`class_handle_t` handle, `uint8_t` ep, `uint8_t` *buffer, `uint32_t` length)
Primes the endpoint to send packet to host.
- `usb_status_t USB_DeviceCdcAcmRecv` (`class_handle_t` handle, `uint8_t` ep, `uint8_t` *buffer, `uint32_t` length)
Primes the endpoint to receive packet from host.

3.6.2.8 Data Structure Documentation

3.6.2.8.1 struct `usb_device_cdc_acm_request_param_struct_t`

Data Fields

- `uint8_t` ** `buffer`
The pointer to the address of the buffer for CDC class request.
- `uint32_t` * `length`
The pointer to the length of the buffer for CDC class request.
- `uint16_t` `interfaceIndex`
The interface index of the setup packet.
- `uint16_t` `setupValue`
The wValue field of the setup packet.

USB CDC Class driver

- `uint8_t isSetup`
The flag indicates if it is a setup packet, 1: yes, 0: no.

3.6.2.8.1.1 Field Documentation

3.6.2.8.1.1.1 `uint8_t** usb_device_cdc_acm_request_param_struct_t::buffer`

3.6.2.8.1.1.2 `uint32_t* usb_device_cdc_acm_request_param_struct_t::length`

3.6.2.8.1.1.3 `uint16_t usb_device_cdc_acm_request_param_struct_t::interfaceIndex`

3.6.2.8.1.1.4 `uint16_t usb_device_cdc_acm_request_param_struct_t::setupValue`

3.6.2.8.1.1.5 `uint8_t usb_device_cdc_acm_request_param_struct_t::isSetup`

3.6.2.8.2 `struct usb_device_cdc_acm_pipe_t`

Data Fields

- `usb_osa_mutex_handle mutex`
The mutex of the pipe.
- `uint8_t * pipeDataBuffer`
pipe data buffer backup when stall
- `uint32_t pipeDataLen`
pipe data length backup when stall
- `uint8_t pipeStall`
pipe is stall
- `uint8_t ep`
The endpoint number of the pipe.
- `uint8_t isBusy`
1: The pipe is transferring packet, 0: The pipe is idle.

3.6.2.8.2.1 Field Documentation

3.6.2.8.2.1.1 `usb_osa_mutex_handle usb_device_cdc_acm_pipe_t::mutex`

3.6.2.8.2.1.2 `uint8_t usb_device_cdc_acm_pipe_t::ep`

3.6.2.8.2.1.3 `uint8_t usb_device_cdc_acm_pipe_t::isBusy`

3.6.2.8.3 `struct usb_device_cdc_acm_struct_t`

Data Fields

- `usb_device_handle handle`
The handle of the USB device.
- `usb_device_class_config_struct_t * configStruct`
The class configure structure.
- `usb_device_interface_struct_t * commInterfaceHandle`
The CDC communication interface handle.
- `usb_device_interface_struct_t * dataInterfaceHandle`

- *The CDC data interface handle.*
 • [usb_device_cdc_acm_pipe_t bulkIn](#)
The bulk in pipe for sending packet to host.
- [usb_device_cdc_acm_pipe_t bulkOut](#)
The bulk out pipe for receiving packet from host.
- [usb_device_cdc_acm_pipe_t interruptIn](#)
The interrupt in pipe for notifying the device state to host.
- [uint8_t configuration](#)
The current configuration value.
- [uint8_t interfaceNumber](#)
The current interface number.
- [uint8_t alternate](#)
The alternate setting value of the interface.
- [uint8_t hasSentState](#)
1: The device has primed the state in interrupt pipe, 0: Not primed the state.

3.6.2.8.3.1 Field Documentation**3.6.2.8.3.1.1** `usb_device_handle usb_device_cdc_acm_struct_t::handle`**3.6.2.8.3.1.2** `usb_device_class_config_struct_t* usb_device_cdc_acm_struct_t::configStruct`**3.6.2.8.3.1.3** `usb_device_interface_struct_t* usb_device_cdc_acm_struct_t::commInterfaceHandle`**3.6.2.8.3.1.4** `usb_device_interface_struct_t* usb_device_cdc_acm_struct_t::dataInterfaceHandle`**3.6.2.8.3.1.5** `usb_device_cdc_acm_pipe_t usb_device_cdc_acm_struct_t::bulkIn`**3.6.2.8.3.1.6** `usb_device_cdc_acm_pipe_t usb_device_cdc_acm_struct_t::bulkOut`**3.6.2.8.3.1.7** `usb_device_cdc_acm_pipe_t usb_device_cdc_acm_struct_t::interruptIn`**3.6.2.8.3.1.8** `uint8_t usb_device_cdc_acm_struct_t::configuration`**3.6.2.8.3.1.9** `uint8_t usb_device_cdc_acm_struct_t::interfaceNumber`**3.6.2.8.3.1.10** `uint8_t usb_device_cdc_acm_struct_t::alternate`**3.6.2.8.3.1.11** `uint8_t usb_device_cdc_acm_struct_t::hasSentState`**3.6.2.9 Macro Definition Documentation****3.6.2.9.1** `#define USB_DEVICE_CONFIG_CDC_ACM_MAX_INSTANCE (1)`**3.6.2.9.2** `#define USB_DEVICE_CONFIG_CDC_COMM_CLASS_CODE (0x02)`**3.6.2.9.3** `#define USB_DEVICE_CONFIG_CDC_DATA_CLASS_CODE (0x0A)`**3.6.2.9.4** `#define USB_DEVICE_CDC_REQUEST_SEND_ENCAPSULATED_COMMAND (0x00)`**3.6.2.9.5** `#define USB_DEVICE_CDC_REQUEST_GET_ENCAPSULATED_RESPONSE (0x01)`**3.6.2.9.6** `#define USB_DEVICE_CDC_REQUEST_SET_COMM_FEATURE (0x02)`**3.6.2.9.7** `#define USB_DEVICE_CDC_REQUEST_GET_COMM_FEATURE (0x03)`**3.6.2.9.8** `#define USB_DEVICE_CDC_REQUEST_CLEAR_COMM_FEATURE (0x04)`**3.6.2.9.9** `#define USB_DEVICE_CDC_REQUEST_SET_AUX_LINE_STATE (0x10)`**3.6.2.9.10** `#define USB_DEVICE_CDC_REQUEST_SET_HOOK_STATE (0x11)`**3.6.2.9.11** `#define USB_DEVICE_CDC_REQUEST_PULSE_SETUP (0x12)`**3.6.2.9.12** `#define USB_DEVICE_CDC_REQUEST_SEND_PULSE (0x13)`**3.6.2.9.13** `#define USB_DEVICE_CDC_REQUEST_SET_PULSE_TIME (0x14)`**3.6.2.9.14** `#define USB_DEVICE_CDC_REQUEST_RING_AUX_JACK (0x15)`

USB CDC Class driver

- kUSB_DeviceCdcEventRecvResponse*** This event indicates the bulk receive transfer is complete.
- kUSB_DeviceCdcEventSerialStateNotif*** This event indicates the serial state has been sent to the host.
- kUSB_DeviceCdcEventSendEncapsulatedCommand*** This event indicates the device received the SEND_ENCAPSULATED_COMMAND request.
- kUSB_DeviceCdcEventGetEncapsulatedResponse*** This event indicates the device received the GET_ENCAPSULATED_RESPONSE request.
- kUSB_DeviceCdcEventSetCommFeature*** This event indicates the device received the SET_COMM_FEATURE request.
- kUSB_DeviceCdcEventGetCommFeature*** This event indicates the device received the GET_COMM_FEATURE request.
- kUSB_DeviceCdcEventClearCommFeature*** This event indicates the device received the CLEAR_COMM_FEATURE request.
- kUSB_DeviceCdcEventGetLineCoding*** This event indicates the device received the GET_LINE_CODING request.
- kUSB_DeviceCdcEventSetLineCoding*** This event indicates the device received the SET_LINE_CODING request.
- kUSB_DeviceCdcEventSetControlLineState*** This event indicates the device received the SET_CONTROL_LINE_STATE request.
- kUSB_DeviceCdcEventSendBreak*** This event indicates the device received the SEND_BREAK request.

3.6.2.11 Function Documentation

3.6.2.11.1 `usb_status_t USB_DeviceCdcAcmlnit (uint8_t controllerId, usb_device_class_config_struct_t * config, class_handle_t * handle)`

This function obtains a USB device handle according to the controller ID, initializes the CDC ACM class with the class configure parameters and creates the mutex for each pipe.

Parameters

<i>controllerId</i>	The ID of the controller. The value can be chosen from the <code>kUSB_ControllerKhci0</code> , <code>kUSB_ControllerKhci1</code> , <code>kUSB_ControllerEhci0</code> , or <code>kUSB_ControllerEhci1</code> .
<i>config</i>	The user configuration structure of type <code>usb_device_class_config_struct_t</code> . The user populates the members of this structure and passes the pointer of this structure into this function.

<i>handle</i>	It is out parameter. The class handle of the CDC ACM class.
---------------	-------------------------------------------------------------

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	The CDC ACM class is initialized successfully.
<i>kStatus_USB_Busy</i>	No CDC ACM device handle available for allocation.
<i>kStatus_USB_Invalid-Handle</i>	The CDC ACM device handle allocation failure.
<i>kStatus_USB_Invalid-Parameter</i>	The USB device handle allocation failure.

3.6.2.11.2 `usb_status_t USB_DeviceCdcAcmDeinit (class_handle_t handle)`

This function destroys the mutex for each pipe, deinitializes each endpoint of the CDC ACM class and frees the CDC ACM class handle.

Parameters

<i>handle</i>	The class handle of the CDC ACM class.
---------------	----------------------------------------

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	The CDC ACM class is de-initialized successfully.
<i>kStatus_USB_Error</i>	The endpoint deinitialization failure.
<i>kStatus_USB_Invalid-Handle</i>	The CDC ACM device handle or the CDC ACM class handle is invalid.

USB CDC Class driver

<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number of the CDC ACM class handle is invalid.
--------------------------------------	-------------------------------------------------------------

3.6.2.11.3 `usb_status_t USB_DeviceCdcAcmEvent (void * handle, uint32_t event, void * param)`

This function responds to various events including the common device events and the class-specific events. For class-specific events, it calls the class callback defined in the application to deal with the class-specific event.

Parameters

<i>handle</i>	The class handle of the CDC ACM class.
<i>event</i>	The event type.
<i>param</i>	The class handle of the CDC ACM class.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	The CDC ACM class is de-initialized successfully.
<i>kStatus_USB_Error</i>	The configure structure of the CDC ACM class handle is invalid.
<i>kStatus_USB_Invalid-Handle</i>	The CDC ACM device handle or the CDC ACM class handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number of the CDC ACM class handle is invalid.
<i>Others</i>	The error code returned by class callback in application.

3.6.2.11.4 `usb_status_t USB_DeviceCdcAcmSend (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

This function checks whether the endpoint is sending packet, then it primes the endpoint with the buffer address and the buffer length if the pipe is not busy. Otherwise, it ignores this transfer by returning an error code.

Parameters

<i>handle</i>	The class handle of the CDC ACM class.
<i>ep</i>	The endpoint number of the transfer.
<i>buffer</i>	The pointer to the buffer to be transferred.
<i>length</i>	The length of the buffer to be transferred.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	Prime to send packet successfully.
<i>kStatus_USB_Busy</i>	The endpoint is busy in transferring.
<i>kStatus_USB_Invalid-Handle</i>	The CDC ACM device handle or the CDC ACM class handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	The controller interface is invalid.

3.6.2.11.5 `usb_status_t USB_DeviceCdcAcmRecv (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

This function checks whether the endpoint is receiving packet, then it primes the endpoint with the buffer address and the buffer length if the pipe is not busy. Otherwise, it ignores this transfer by returning an error code.

Parameters

<i>handle</i>	The class handle of the CDC ACM class.
<i>ep</i>	The endpoint number of the transfer.
<i>buffer</i>	The pointer to the buffer to be transferred.
<i>length</i>	The length of the buffer to be transferred.

Returns

A USB error code or `kStatus_USB_Success`.

USB CDC Class driver

Return values

<i>kStatus_USB_Success</i>	Prime to receive packet successfully.
<i>kStatus_USB_Busy</i>	The endpoint is busy in transferring.
<i>kStatus_USB_Invalid-Handle</i>	The CDC ACM device handle or the CDC ACM class handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	The controller interface is invalid.

3.6.3 USB CDC RNDIS driver

3.6.3.1 Overview

This section describes the programming interface of the USB CDC RNDIS driver. The USB CDC RNDIS driver implements the various control messages and data message defined by Microsoft RNDIS. The control messages is sent through the SEND_ENCAPSULATED_COMMAND and GET_ENCAPSULATED_COMMAND CDC class request.

3.6.3.2 USB CDC RNDIS Device structures

The driver uses an instantiation of the `usb_device_cdc_rndis_struct_t` structure to maintain the current state of a particular CDC RNDIS instance module driver.

The CDC RNDIS driver populates the structure members.

3.6.3.3 CDC RNDIS Initialization

The CDC RNDIS device is initialized with the configure structure of type `usb_device_cdc_rndis_config_struct_t`. It specifies the RNDIS request specific callback function and the maximum transmit size for device. Besides, the device state, hardware state and the media status is set to their initial value.

3.6.3.4 CDC RNDIS Control Message

The control messages is sent through the SEND_ENCAPSULATED_COMMAND and GET_ENCAPSULATED_COMMAND CDC class request. Take the RNDIS_INITIALIZE_MSG as an example, the host sends a SEND_ENCAPSULATED_COMMAND request which carries the message type of RNDIS_INITIALIZE_MSG to the device, then the device sends back a notification through interrupt pipe to indicate that the response is available. Next the host sends a GET_ENCAPSULATED_COMMAND request which carries the message type of RNDIS_INITIALIZE_CMPLT to the device to obtain the proper information.

Data Structures

- struct `rndis_init_msg_struct_t`
Define message structure for REMOTE_NDIS_INITIALIZE_MSG. [More...](#)
- struct `rndis_init_cmplt_struct_t`
Define message structure for REMOTE_NDIS_INITIALIZE_CMPLT. [More...](#)
- struct `rndis_halt_msg_struct_t`
Define message structure for REMOTE_NDIS_HALT_MSG. [More...](#)
- struct `rndis_query_msg_struct_t`
Define message structure for REMOTE_NDIS_QUERY_MSG. [More...](#)
- struct `rndis_query_cmplt_struct_t`
Define message structure for REMOTE_NDIS_QUERY_CMPLT. [More...](#)
- struct `rndis_set_msg_struct_t`

USB CDC Class driver

- Define message structure for REMOTE_NDIS_SET_MSG. [More...](#)
• struct [rndis_set_cmplt_struct_t](#)
- Define message structure for REMOTE_NDIS_SET_CMPLT. [More...](#)
• struct [rndis_reset_msg_struct_t](#)
- Define message structure for REMOTE_NDIS_RESET_MSG. [More...](#)
• struct [rndis_reset_cmplt_struct_t](#)
- Define message structure for REMOTE_NDIS_RESET_CMPLT. [More...](#)
• struct [rndis_indicate_status_msg_struct_t](#)
- Define message structure for REMOTE_NDIS_INDICATE_STATUS_MSG. [More...](#)
• struct [rndis_keepalive_msg_struct_t](#)
- Define message structure for REMOTE_NDIS_KEEPALIVE_MSG. [More...](#)
• struct [rndis_keepalive_cmplt_struct_t](#)
- Define message structure for REMOTE_NDIS_KEEPALIVE_CMPLT. [More...](#)
• struct [rndis_packet_msg_struct_t](#)
- Define message structure for RNDIS_PACKET_MSG. [More...](#)
• struct [usb_device_cdc_rndis_struct_t](#)
- Define structure for CDC RNDIS device. [More...](#)
• struct [usb_device_cdc_rndis_config_struct_t](#)
- Define structure for CDC RNDIS device. [More...](#)
• struct [usb_device_cdc_rndis_request_param_struct_t](#)
- Define parameters for CDC RNDIS request. [More...](#)

Macros

- #define [USB_DEVICE_CONFIG_CDC_RNDIS_MAX_INSTANCE](#) (1U)
The maximum number of USB CDC RNDIS device instance.
- #define [RNDIS_DF_CONNECTIONLESS](#) (0x00000001U)
The Miniport driver type is connectionless.
- #define [RNDIS_DF_CONNECTION_ORIENTED](#) (0x00000002U)
The Miniport driver type is connection-oriented.
- #define [RNDIS_SINGLE_PACKET_TRANSFER](#) (0x00000001U)
The number of RNDIS data messages that the device can handle in a single transfer.
- #define [RNDIS_PACKET_ALIGNMENT_FACTOR](#) (0x00000003U)
The byte alignment that the device expects for each RNDIS message that is part of a multmessage transfer.
- #define [RNDIS_NUM_OIDS_SUPPORTED](#) (25U)
The number of OIDs the RNDIS device supported.
- #define [RNDIS_VENDOR_ID](#) (0xFFFFFFFU)
The vendor ID of the RNDIS device.
- #define [RNDIS_NIC_IDENTIFIER_VENDOR](#) (0x01U)
A single byte that the vendor assigns to identify a particular NIC.
- #define [RNDIS_MAX_EXPECTED_COMMAND_SIZE](#) (76U)
DataLength : Data length of communication feature.
- #define [RNDIS_MAX_EXPECTED_RESPONSE_SIZE](#) (RNDIS_RESPONSE_QUERY_MSG_SIZE + (RNDIS_NUM_OIDS_SUPPORTED << 2U))
This is the maximum observed command size we get on control endpoint – Memory for commands is allocated at initialization, instead of being dynamically allocated when command is received to avoid memory fragmentation.
- #define [RNDIS_ETHER_ADDR_SIZE](#) (6U)
Size of Ethernet address.
- #define [RNDIS_USB_HEADER_SIZE](#) (44U)

- *Size of USB header for RNDIS packet.*
 • #define **RNDIS_MULTICAST_LIST_SIZE** (0U)
Maximum size of multicast address list.

Enumerations

- enum **ndis_physical_medium_enum_t**
Physical Medium Type definitions.
- enum **ndis_state_enum_t** {
RNDIS_UNINITIALIZED = 0,
RNDIS_INITIALIZED,
RNDIS_DATA_INITIALIZED }
Define RNDIS device state.
- enum **ndis_event_enum_t** {
kUSB_DeviceCdcEventAppGetLinkSpeed,
kUSB_DeviceCdcEventAppGetSendPacketSize,
kUSB_DeviceCdcEventAppGetRecvPacketSize,
kUSB_DeviceCdcEventAppGetMacAddress,
kUSB_DeviceCdcEventAppGetLinkStatus,
kUSB_DeviceCdcEventAppGetMaxFrameSize }
Define RNDIS event.

RNDIS Control Message Type

See MSDN for details.

- #define **RNDIS_PACKET_MSG** (0x00000001U)
- #define **RNDIS_INITIALIZE_MSG** (0x00000002U)
- #define **RNDIS_HALT_MSG** (0x00000003U)
- #define **RNDIS_QUERY_MSG** (0x00000004U)
- #define **RNDIS_SET_MSG** (0x00000005U)
- #define **RNDIS_RESET_MSG** (0x00000006U)
- #define **RNDIS_INDICATE_STATUS_MSG** (0x00000007U)
- #define **RNDIS_KEEPLIVE_MSG** (0x00000008U)
- #define **RNDIS_INITIALIZE_CMPLT** (0x80000002U)
- #define **RNDIS_QUERY_CMPLT** (0x80000004U)
- #define **RNDIS_SET_CMPLT** (0x80000005U)
- #define **RNDIS_RESET_CMPLT** (0x80000006U)
- #define **RNDIS_KEEPLIVE_CMPLT** (0x80000008U)

Object Identifiers used by NdisRequest Query/Set Information

See MSDN for details.

- #define **NDIS_OID_GEN_SUPPORTED_LIST** (0x00010101U)
- #define **NDIS_OID_GEN_HARDWARE_STATUS** (0x00010102U)
- #define **NDIS_OID_GEN_MEDIA_SUPPORTED** (0x00010103U)
- #define **NDIS_OID_GEN_MEDIA_IN_USE** (0x00010104U)
- #define **NDIS_OID_GEN_MAXIMUM_LOOKAHEAD** (0x00010105U)

- #define **NDIS_OID_GEN_MAXIMUM_FRAME_SIZE** (0x00010106U)
- #define **NDIS_OID_GEN_LINK_SPEED** (0x00010107U)
- #define **NDIS_OID_GEN_TRANSMIT_BUFFER_SPACE** (0x00010108U)
- #define **NDIS_OID_GEN_RECEIVE_BUFFER_SPACE** (0x00010109U)
- #define **NDIS_OID_GEN_TRANSMIT_BLOCK_SIZE** (0x0001010AU)
- #define **NDIS_OID_GEN_RECEIVE_BLOCK_SIZE** (0x0001010BU)
- #define **NDIS_OID_GEN_VENDOR_ID** (0x0001010CU)
- #define **NDIS_OID_GEN_VENDOR_DESCRIPTION** (0x0001010DU)
- #define **NDIS_OID_GEN_CURRENT_PACKET_FILTER** (0x0001010EU)
- #define **NDIS_OID_GEN_CURRENT_LOOKAHEAD** (0x0001010FU)
- #define **NDIS_OID_GEN_DRIVER_VERSION** (0x00010110U)
- #define **NDIS_OID_GEN_MAXIMUM_TOTAL_SIZE** (0x00010111U)
- #define **NDIS_OID_GEN_PROTOCOL_OPTIONS** (0x00010112U)
- #define **NDIS_OID_GEN_MAC_OPTIONS** (0x00010113U)
- #define **NDIS_OID_GEN_MEDIA_CONNECT_STATUS** (0x00010114U)
- #define **NDIS_OID_GEN_MAXIMUM_SEND_PACKETS** (0x00010115U)
- #define **NDIS_OID_GEN_XMIT_OK** (0x00020101U)
- #define **NDIS_OID_GEN_RCV_OK** (0x00020102U)
- #define **NDIS_OID_GEN_XMIT_ERROR** (0x00020103U)
- #define **NDIS_OID_GEN_RCV_ERROR** (0x00020104U)
- #define **NDIS_OID_GEN_RCV_NO_BUFFER** (0x00020105U)
- #define **NDIS_OID_GEN_DIRECTED_BYTES_XMIT** (0x00020201U)
- #define **NDIS_OID_GEN_DIRECTED_FRAMES_XMIT** (0x00020202U)
- #define **NDIS_OID_GEN_MULTICAST_BYTES_XMIT** (0x00020203U)
- #define **NDIS_OID_GEN_MULTICAST_FRAMES_XMIT** (0x00020204U)
- #define **NDIS_OID_GEN_BROADCAST_BYTES_XMIT** (0x00020205U)
- #define **NDIS_OID_GEN_BROADCAST_FRAMES_XMIT** (0x00020206U)
- #define **NDIS_OID_GEN_DIRECTED_BYTES_RCV** (0x00020207U)
- #define **NDIS_OID_GEN_DIRECTED_FRAMES_RCV** (0x00020208U)
- #define **NDIS_OID_GEN_MULTICAST_BYTES_RCV** (0x00020209U)
- #define **NDIS_OID_GEN_MULTICAST_FRAMES_RCV** (0x0002020AU)
- #define **NDIS_OID_GEN_BROADCAST_BYTES_RCV** (0x0002020BU)
- #define **NDIS_OID_GEN_BROADCAST_FRAMES_RCV** (0x0002020CU)
- #define **NDIS_OID_GEN_RCV_CRC_ERROR** (0x0002020DU)
- #define **NDIS_OID_GEN_TRANSMIT_QUEUE_LENGTH** (0x0002020EU)
- #define **NDIS_OID_GEN_GET_TIME_CAPS** (0x0002020FU)
- #define **NDIS_OID_GEN_GET_NETCARD_TIME** (0x00020210U)
- #define **NDIS_OID_802_3_PERMANENT_ADDRESS** (0x01010101U)
- #define **NDIS_OID_802_3_CURRENT_ADDRESS** (0x01010102U)
- #define **NDIS_OID_802_3_MULTICAST_LIST** (0x01010103U)
- #define **NDIS_OID_802_3_MAXIMUM_LIST_SIZE** (0x01010104U)
- #define **NDIS_OID_802_3_MAC_OPTIONS** (0x01010105U)
- #define **NDIS_802_3_MAC_OPTION_PRIORITY** (0x00000001U)
- #define **NDIS_OID_802_3_RCV_ERROR_ALIGNMENT** (0x01020101U)
- #define **NDIS_OID_802_3_XMIT_ONE_COLLISION** (0x01020102U)
- #define **NDIS_OID_802_3_XMIT_MORE_COLLISIONS** (0x01020103U)
- #define **NDIS_OID_802_3_XMIT_DEFERRED** (0x01020201U)
- #define **NDIS_OID_802_3_XMIT_MAX_COLLISIONS** (0x01020202U)
- #define **NDIS_OID_802_3_RCV_OVERRUN** (0x01020203U)
- #define **NDIS_OID_802_3_XMIT_UNDERRUN** (0x01020204U)
- #define **NDIS_OID_802_3_XMIT_HEARTBEAT_FAILURE** (0x01020205U)
- #define **NDIS_OID_802_3_XMIT_TIMES_CRS_LOST** (0x01020206U)
- #define **NDIS_OID_802_3_XMIT_LATE_COLLISIONS** (0x01020207U)
- #define **NDIS_OID_GEN_VENDOR_DRIVER_VERSION** (0x00010116U)
- #define **NDIS_OID_GEN_SUPPORTED_GUIDS** (0x00010117U)
- #define **NDIS_OID_GEN_NETWORK_LAYER_ADDRESSES** (0x00010118U) /* Set only */

- #define **NDIS_OID_GEN_TRANSPORT_HEADER_OFFSET** (0x00010119U) /* Set only */
- #define **NDIS_OID_GEN_MACHINE_NAME** (0x0001021AU)
- #define **NDIS_OID_GEN_RNDIS_CONFIG_PARAMETER** (0x0001021BU) /* Set only */
- #define **NDIS_OID_GEN_VLAN_ID** (0x0001021CU)
- #define **NDIS_OID_GEN_MEDIA_CAPABILITIES** (0x00010201U)
- #define **NDIS_OID_GEN_PHYSICAL_MEDIUM** (0x00010202U)

NDIS Hardware status codes for OID_GEN_HARDWARE_STATUS

See MSDN for details.

- #define **NDIS_HARDWARE_STATUS_READY** (0x00000000U)
Available and capable of sending and receiving data over the wire.
- #define **NDIS_HARDWARE_STATUS_INITIALIZING** (0x00000001U)
Initializing.
- #define **NDIS_HARDWARE_STATUS_RESET** (0x00000002U)
Resetting.
- #define **NDIS_HARDWARE_STATUS_CLOSING** (0x00000003U)
Closing.
- #define **NDIS_HARDWARE_STATUS_NOT_READY** (0x00000004U)
Not ready.

NDIS media types that the NIC can support

See MSDN for details.

- #define **NDIS_MEDIUM802_3** (0x00000000U)
Ethernet (802.3) is not supported for NDIS 6.0 drivers.
- #define **NDIS_MEDIUM802_5** (0x00000001U)
Token Ring (802.5) is not supported for NDIS 6.0 drivers.
- #define **NDIS_MEDIUM_FDDI** (0x00000002U)
FDDI is not supported on Windows[®] Vista.
- #define **NDIS_MEDIUM_WAN** (0x00000003U)
WAN.
- #define **NDIS_MEDIUM_LOCAL_TALK** (0x00000004U)
LocalTalk.
- #define **NDIS_MEDIUM_DIX** (0x00000005U)
DEC/Intel/Xerox (DIX) Ethernet.
- #define **NDIS_MEDIUM_ARCNET_RAW** (0x00000006U)
ARCNET (raw) is not supported on Windows Vista.
- #define **NDIS_MEDIUM_ARCNET878_2** (0x00000007U)
ARCNET (878.2) is not supported on Windows Vista.
- #define **NDIS_MEDIUM_ATM** (0x00000008U)
ATM is not supported for NDIS 6.0 drivers.
- #define **NDIS_MEDIUM_NATIVE802_11** (0x00000009U)
Native 802.11.
- #define **NDIS_MEDIUM_WIRELESS_WAN** (0x0000000AU)
Various types of NdisWirelessXxx media Note This media type is not available for use beginning with Windows Vista.
- #define **NDIS_MEDIUM_IRDA** (0x0000000BU)

USB CDC Class driver

- *Infrared (IrDA)*
• #define [NDIS_MEDIUM_COWAN](#) (0x0000000CU)
- *Connection-oriented WAN.*
• #define [NDIS_MEDIUM1394](#) (0x0000000DU)
- *IEEE 1394 (firewire) bus.*
• #define [NDIS_MEDIUM_BPC](#) (0x0000000EU)
- *Broadcast PC network.*
• #define [NDIS_MEDIUM_INFINI_BAND](#) (0x0000000FU)
- *InfiniBand network.*
• #define [NDIS_MEDIUM_TUNNEL](#) (0x00000010U)
- *Tunnel network.*
• #define [NDIS_MEDIUM_LOOPBACK](#) (0x00000011U)
- *NDIS loopback network.*

NDIS Packet Filter Bits for `OID_GEN_CURRENT_PACKET_FILTER`.

See MSDN for details.

- #define [NDIS_PACKET_TYPE_DIRECTED](#) (0x0001U)
Directed packets.
- #define [NDIS_PACKET_TYPE_MULTICAST](#) (0x0002U)
Multicast address packets sent to addresses in the multicast address list.
- #define [NDIS_PACKET_TYPE_ALL_MULTICAST](#) (0x0004U)
All multicast address packets, not just the ones enumerated in the multicast address list.
- #define [NDIS_PACKET_TYPE_BROADCAST](#) (0x0008U)
Broadcast packets.
- #define [NDIS_PACKET_TYPE_SOURCE_ROUTING](#) (0x0010U)
All source routing packets.
- #define [NDIS_PACKET_TYPE_PROMISCUOUS](#) (0x0020U)
Specifies all packets.
- #define [NDIS_PACKET_TYPE_SMT](#) (0x0040U)
SMT packets that an FDDI NIC receives.
- #define [NDIS_PACKET_TYPE_ALL_LOCAL](#) (0x0080U)
All packets sent by installed protocols and all packets indicated by the NIC that is identified by a given NdisBindingHandle.
- #define [NDIS_PACKET_TYPE_MAC_FRAME](#) (0x8000U)
NIC driver frames that a Token Ring NIC receives.
- #define [NDIS_PACKET_TYPE_FUNCTIONAL](#) (0x4000U)
Functional address packets sent to addresses included in the current functional address.
- #define [NDIS_PACKET_TYPE_ALL_FUNCTIONAL](#) (0x2000U)
All functional address packets, not just the ones in the current functional address.
- #define [NDIS_PACKET_TYPE_GROUP](#) (0x1000U)
Packets sent to the current group address.

RNDIS status values

See MSDN for details.

- #define [RNDIS_STATUS_SUCCESS](#) (0x00000000U)

- *The requested operation completed successfully.*
- #define **RNDIS_STATUS_NOT_RECOGNIZED** (0x00010001U)
- *The underlying driver does not support the requested operation.*
- #define **RNDIS_STATUS_NOT_SUPPORTED** (0xC00000BBU)
- *Unsupported request error (equivalent to STATUS_NOT_SUPPORTED).*
- #define **RNDIS_STATUS_NOT_ACCEPTED** (0x00010003U)
- *The underlying driver attempted the requested operation, usually a set, on its NIC but it was aborted by the Netcard.*
- #define **RNDIS_STATUS_FAILURE** (0xC0000001U)
- *This value usually is a non specific default, returned when none of the more specific NDIS_STATUS_XXX causes the underlying driver to fail the request.*
- #define **RNDIS_STATUS_RESOURCES** (0xC000009AU)
- *The request can't be satisfied due to a resource shortage.*
- #define **RNDIS_STATUS_CLOSING** (0xC0010002U)
- *The underlying driver failed the requested operation because a close is in progress.*
- #define **RNDIS_STATUS_CLOSING_INDICATING** (0xC001000EU)
- *The underlying driver failed the requested operation because indicating a close is in progress.*
- #define **RNDIS_STATUS_RESET_IN_PROGRESS** (0xC001000DU)
- *The underlying NIC driver cannot satisfy the request at this time because it is currently resetting the Netcard.*
- #define **RNDIS_STATUS_INVALID_LENGTH** (0xC0010014U)
- *The value specified in the InformationBufferLength member of the NDIS_REQUEST-structured buffer at NdisRequest does not match the requirements for the given OID_XXX code.*
- #define **RNDIS_STATUS_BUFFER_TOO_SHORT** (0xC0010016U)
- *The information buffer is too small.*
- #define **RNDIS_STATUS_INVALID_DATA** (0xC0010015U)
- *The data supplied at InformationBuffer in the given NDIS_REQUEST structure is invalid for the given OID_XXX code.*
- #define **RNDIS_STATUS_INVALID_OID** (0xC0010017U)
- *The OID_XXX code specified in the OID member of the NDIS_REQUEST-structured buffer at NdisRequest is invalid or unsupported by the underlying driver.*
- #define **RNDIS_STATUS_MEDIA_CONNECT** (0x4001000BU)
- *Device is connected to network medium.*
- #define **RNDIS_STATUS_MEDIA_DISCONNECT** (0x4001000CU)
- *Device is disconnected from network medium.*

RNDIS Response sizes

Definitions of the size of response of various message types.

- #define **RNDIS_RESPONSE_INITIALIZE_MSG_SIZE** (52U)
- *Response size of INITIALIZE_MSG.*
- #define **RNDIS_RESPONSE_QUERY_MSG_SIZE** (24U)
- *Response size of QUERY_MSG.*
- #define **RNDIS_RESPONSE_SET_MSG_SIZE** (16U)
- *Response size of SET_MSG.*
- #define **RNDIS_RESPONSE_RESET_MSG_SIZE** (16U)
- *Response size of RESET_MSG.*
- #define **RNDIS_RESPONSE_KEEPALIVE_MSG_SIZE** (16U)
- *Response size of KEEPALIVE_MSG.*

USB CDC Class driver

RNDIS device connection status

Definitions of the status value of NIC connection.

- #define **NDIS_MEDIA_STATE_CONNECTED** (0x00000000U)
The network connection has been lost.
- #define **NDIS_MEDIA_STATE_DISCONNECTED** (0x00000001U)
The network connection has been restored.
- #define **NDIS_MEDIA_STATE_UNKNOWN** (0xFFFFFFFFU)
The initial value of the connection status.

Reserved for connection oriented devices. Set value to zero.

- #define **RNDIS_AF_LIST_OFFSET** (0x00000000U)
- #define **RNDIS_AF_LIST_SIZE** (0x00000000U)

USB CDC ACM Class Driver

- **usb_status_t USB_DeviceCdcRndisInit** (class_handle_t classHandle, usb_device_cdc_rndis_config_struct_t *config, usb_device_cdc_rndis_struct_t **handle)
Initializes the USB CDC RNDIS device.
- **usb_status_t USB_DeviceCdcRndisDeinit** (usb_device_cdc_rndis_struct_t *handle)
Deinitializes the USB CDC RNDIS device.
- **usb_status_t USB_DeviceCdcRndisMessageSet** (usb_device_cdc_rndis_struct_t *handle, uint8_t **message, uint32_t *len)
Responds to kUSB_DeviceCdcEventSendEncapsulatedCommand.
- **usb_status_t USB_DeviceCdcRndisMessageGet** (usb_device_cdc_rndis_struct_t *handle, uint8_t **message, uint32_t *len)
Responds to kUSB_DeviceCdcEventGetEncapsulatedResponse.
- **usb_status_t USB_DeviceCdcRndisResetCommand** (usb_device_cdc_rndis_struct_t *handle, uint8_t **message, uint32_t *len)
Soft reset the RNDIS device.
- **usb_status_t USB_DeviceCdcRndisHaltCommand** (usb_device_cdc_rndis_struct_t *handle)
Halts the RNDIS device.

3.6.3.5 Data Structure Documentation

- 3.6.3.5.1 struct rdis_init_msg_struct_t
- 3.6.3.5.2 struct rdis_init_cmplt_struct_t
- 3.6.3.5.3 struct rdis_halt_msg_struct_t
- 3.6.3.5.4 struct rdis_query_msg_struct_t
- 3.6.3.5.5 struct rdis_query_cmplt_struct_t
- 3.6.3.5.6 struct rdis_set_msg_struct_t
- 3.6.3.5.7 struct rdis_set_cmplt_struct_t
- 3.6.3.5.8 struct rdis_reset_msg_struct_t
- 3.6.3.5.9 struct rdis_reset_cmplt_struct_t
- 3.6.3.5.10 struct rdis_indicate_status_msg_struct_t
- 3.6.3.5.11 struct rdis_keepalive_msg_struct_t
- 3.6.3.5.12 struct rdis_keepalive_cmplt_struct_t
- 3.6.3.5.13 struct rdis_packet_msg_struct_t
- 3.6.3.5.14 struct usb_device_cdc_rdis_struct_t

Data Fields

- [class_handle_t cdcAcmHandle](#)
USB CDC ACM class handle.
- [uint8_t * rdisCommand](#)
The pointer to the buffer of the RNDIS request.
- [uint8_t * responseData](#)
The pointer to the buffer of the RNDIS response.
- [uint32_t rdisHostMaxTxSize](#)
The maximum transmit size in byte of the host.
- [uint32_t rdisDevMaxTxSize](#)
The maximum transmit size in byte of the device.
- [uint32_t rdisHwState](#)
The hardware state of the RNDIS device.
- [uint32_t rdisPacketFilter](#)
The packet filter of the RNDIS device.
- [uint32_t rdisMediaConnectStatus](#)
The media connection status of the RNDIS device.

USB CDC Class driver

- uint32_t [numFramesTxOk](#)
The number of the frames sent successfully.
- uint32_t [numFramesRxOk](#)
The number of the frames received successfully.
- uint32_t [numFramesTxError](#)
The number of the frames sent failed.
- uint32_t [numFramesRxError](#)
The number of the frames received failed.
- uint32_t [numRecvFramesMissed](#)
The number of the frames missed to receive.
- uint32_t [numRecvFramesAlignmentError](#)
The number of the frames received that has alignment error.
- uint32_t [numFramesTxOneCollision](#)
The number of the frames sent that has one collision.
- uint32_t [numFramesTxManyCollision](#)
The number of the frames sent that has many collision.
- uint8_t [rndisDeviceState](#)
The RNDIS device state.
- [usb_osa_mutex_handle](#) statusMutex
The mutex to guarantee the consistent access to the device state.
- [usb_status_t](#)(* [rndisCallback](#))(class_handle_t handle, uint32_t event, void *param)
The callback function provided by application for the RNDIS request.

3.6.3.5.14.1 Field Documentation

- 3.6.3.5.14.1.1 `class_handle_t usb_device_cdc_rndis_struct_t::cdcAcmHandle`
- 3.6.3.5.14.1.2 `uint8_t* usb_device_cdc_rndis_struct_t::rndisCommand`
- 3.6.3.5.14.1.3 `uint8_t* usb_device_cdc_rndis_struct_t::responseData`
- 3.6.3.5.14.1.4 `uint32_t usb_device_cdc_rndis_struct_t::rndisHostMaxTxSize`
- 3.6.3.5.14.1.5 `uint32_t usb_device_cdc_rndis_struct_t::rndisDevMaxTxSize`
- 3.6.3.5.14.1.6 `uint32_t usb_device_cdc_rndis_struct_t::rndisHwState`
- 3.6.3.5.14.1.7 `uint32_t usb_device_cdc_rndis_struct_t::rndisPacketFilter`
- 3.6.3.5.14.1.8 `uint32_t usb_device_cdc_rndis_struct_t::rndisMediaConnectStatus`
- 3.6.3.5.14.1.9 `uint32_t usb_device_cdc_rndis_struct_t::numFramesTxOk`
- 3.6.3.5.14.1.10 `uint32_t usb_device_cdc_rndis_struct_t::numFramesRxOk`
- 3.6.3.5.14.1.11 `uint32_t usb_device_cdc_rndis_struct_t::numFramesTxError`
- 3.6.3.5.14.1.12 `uint32_t usb_device_cdc_rndis_struct_t::numFramesRxError`
- 3.6.3.5.14.1.13 `uint32_t usb_device_cdc_rndis_struct_t::numRecvFramesMissed`
- 3.6.3.5.14.1.14 `uint32_t usb_device_cdc_rndis_struct_t::numRecvFramesAlignmentError`
- 3.6.3.5.14.1.15 `uint32_t usb_device_cdc_rndis_struct_t::numFramesTxOneCollision`
- 3.6.3.5.14.1.16 `uint32_t usb_device_cdc_rndis_struct_t::numFramesTxManyCollision`
- 3.6.3.5.14.1.17 `uint8_t usb_device_cdc_rndis_struct_t::rndisDeviceState`
- 3.6.3.5.14.1.18 `usb_osa_mutex_handle usb_device_cdc_rndis_struct_t::statusMutex`
- 3.6.3.5.14.1.19 `usb_status_t(* usb_device_cdc_rndis_struct_t::rndisCallback)(class_handle_t handle, uint32_t event, void *param)`
- 3.6.3.5.15 `struct usb_device_cdc_rndis_config_struct_t`

Data Fields

- `uint32_t devMaxTxSize`
The maximum transmit size in byte of the device.
- `usb_status_t(* rndisCallback)(class_handle_t handle, uint32_t event, void *param)`
The callback function provided by application for the RNDIS request.

USB CDC Class driver

3.6.3.5.15.1 Field Documentation

3.6.3.5.15.1.1 `uint32_t usb_device_cdc_rndis_config_struct_t::devMaxTxSize`

This value is configured by application.

3.6.3.5.15.1.2 `usb_status_t(* usb_device_cdc_rndis_config_struct_t::rndisCallback)(class_handle_t handle, uint32_t event, void *param)`

3.6.3.5.16 `struct usb_device_cdc_rndis_request_param_struct_t`

Data Fields

- `uint8_t * buffer`
The pointer to the buffer for RNDIS request.
- `uint32_t length`
The length of the buffer for RNDIS request.

3.6.3.5.16.1 Field Documentation

3.6.3.5.16.1.1 `uint8_t* usb_device_cdc_rndis_request_param_struct_t::buffer`

3.6.3.5.16.1.2 `uint32_t usb_device_cdc_rndis_request_param_struct_t::length`

3.6.3.6 Macro Definition Documentation

3.6.3.6.1 `#define NDIS_MEDIUM802_3 (0x00000000U)`

Note NDIS 5.x Miniport drivers that conform to the IEEE[®] 802.11 interface must use this media type. For more information about the 802.11 interface, see 802.11 Wireless LAN Miniport Drivers.

3.6.3.6.2 `#define NDIS_MEDIUM802_5 (0x00000001U)`

3.6.3.6.3 `#define NDIS_MEDIUM_FDDI (0x00000002U)`

3.6.3.6.4 `#define NDIS_MEDIUM_ARCNET_RAW (0x00000006U)`

3.6.3.6.5 `#define NDIS_MEDIUM_ARCNET878_2 (0x00000007U)`

3.6.3.6.6 `#define NDIS_MEDIUM_ATM (0x00000008U)`

3.6.3.6.7 `#define NDIS_MEDIUM_NATIVE802_11 (0x00000009U)`

This media type is used by Miniport drivers that conform to the Native 802.11 interface. For more information about this interface, see Native 802.11 Wireless LAN Miniport Drivers. Note: Native 802.11 interface is supported in NDIS 6.0 and later versions

3.6.3.6.8 #define NDIS_MEDIUM_BPC (0x0000000EU)

3.6.3.6.9 #define NDIS_MEDIUM_INFINI_BAND (0x0000000FU)

3.6.3.6.10 #define NDIS_MEDIUM_TUNNEL (0x00000010U)

3.6.3.6.11 #define NDIS_MEDIUM_LOOPBACK (0x00000011U)

3.6.3.6.12 #define NDIS_PACKET_TYPE_DIRECTED (0x0001U)

Directed packets contain a destination address equal to the station address of the NIC.

3.6.3.6.13 #define NDIS_PACKET_TYPE_MULTICAST (0x0002U)

A protocol driver can receive Ethernet (802.3) multicast packets or Token Ring (802.5) functional address packets by specifying the multicast or functional address packet type. Setting the multicast address list or functional address determines which multicast address groups the NIC driver enables.

3.6.3.6.14 #define NDIS_PACKET_TYPE_BROADCAST (0x0008U)

3.6.3.6.15 #define NDIS_PACKET_TYPE_SOURCE_ROUTING (0x0010U)

If the protocol driver sets this bit, the NDIS library attempts to act as a source routing bridge.

3.6.3.6.16 #define NDIS_PACKET_TYPE_PROMISCUOUS (0x0020U)

3.6.3.6.17 #define NDIS_PACKET_TYPE_SMT (0x0040U)

3.6.3.6.18 #define NDIS_PACKET_TYPE_MAC_FRAME (0x8000U)

3.6.3.6.19 #define NDIS_PACKET_TYPE_GROUP (0x1000U)

3.6.3.6.20 #define RNDIS_STATUS_SUCCESS (0x00000000U)

3.6.3.6.21 #define RNDIS_STATUS_NOT_RECOGNIZED (0x00010001U)

3.6.3.6.22 #define RNDIS_STATUS_NOT_SUPPORTED (0xC00000BBU)

3.6.3.6.23 #define RNDIS_STATUS_NOT_ACCEPTED (0x00010003U)

For example, an attempt to set too many multicast addresses might cause the return of this value.

3.6.3.6.24 **#define RNDIS_STATUS_RESOURCES (0xC00009AU)**

Usually, this return indicates that an attempt to allocate memory was unsuccessful, but it does not necessarily indicate that the same request, submitted later, it is aborted for the same reason.

3.6.3.6.25 **#define RNDIS_STATUS_CLOSING (0xC0010002U)**

3.6.3.6.26 **#define RNDIS_STATUS_CLOSING_INDICATING (0xC001000EU)**

3.6.3.6.27 **#define RNDIS_STATUS_INVALID_LENGTH (0xC0010014U)**

If the information buffer is too small, the BytesNeeded member contains the correct value for Information-BufferLength on return from NdisRequest.

3.6.3.6.28 **#define RNDIS_STATUS_BUFFER_TOO_SHORT (0xC0010016U)**

3.6.3.6.29 **#define RNDIS_STATUS_MEDIA_CONNECT (0x4001000BU)**

3.6.3.6.30 **#define RNDIS_STATUS_MEDIA_DISCONNECT (0x4001000CU)**

3.6.3.6.31 **#define RNDIS_RESPONSE_INITIALIZE_MSG_SIZE (52U)**

3.6.3.6.32 **#define RNDIS_RESPONSE_QUERY_MSG_SIZE (24U)**

3.6.3.6.33 **#define RNDIS_RESPONSE_SET_MSG_SIZE (16U)**

3.6.3.6.34 **#define RNDIS_RESPONSE_RESET_MSG_SIZE (16U)**

3.6.3.6.35 **#define RNDIS_RESPONSE_KEEPALIVE_MSG_SIZE (16U)**

3.6.3.6.36 **#define RNDIS_DF_CONNECTIONLESS (0x00000001U)**

3.6.3.6.37 **#define RNDIS_DF_CONNECTION_ORIENTED (0x00000002U)**

3.6.3.6.38 **#define RNDIS_SINGLE_PACKET_TRANSFER (0x00000001U)**

3.6.3.6.39 **#define RNDIS_PACKET_ALIGNMENT_FACTOR (0x00000003U)**

3.6.3.6.40 **#define RNDIS_NUM_OIDS_SUPPORTED (25U)**

3.6.3.6.41 **#define RNDIS_VENDOR_ID (0xFFFFFFFFU)**

Vendors without an IEEE-registered code should use the value 0xFFFFFFFF.

3.6.3.6.42 **#define** NDIS_MEDIA_STATE_CONNECTED (0x00000000U)

3.6.3.6.43 **#define** NDIS_MEDIA_STATE_DISCONNECTED (0x00000001U)

3.6.3.6.44 **#define** NDIS_MEDIA_STATE_UNKNOWN (0xFFFFFFFFU)

3.6.3.6.45 **#define** RNDIS_MAX_EXPECTED_COMMAND_SIZE (76U)

3.6.3.6.46 **#define** RNDIS_ETHER_ADDR_SIZE (6U)

3.6.3.6.47 **#define** RNDIS_USB_HEADER_SIZE (44U)

3.6.3.6.48 **#define** RNDIS_MULTICAST_LIST_SIZE (0U)

3.6.3.7 Enumeration Type Documentation

3.6.3.7.1 **enum** ndis_physical_medium_enum_t

Used with OID_GEN_PHYSICAL_MEDIUM.

3.6.3.7.2 **enum** rndis_state_enum_t

See MSDN for details.

Enumerator

RNDIS_UNINITIALIZED Following bus-level initialization, the device is said to be in the RNDIS-uninitialized state. If the device receives a REMOTE_NDIS_HALT_MSG, a bus-level disconnects, or a hard-reset at any time, it forces the device to the RNDIS-uninitialized state.

RNDIS_INITIALIZED After the device receives a REMOTE_NDIS_INITIALIZE_MSG and responds with a REMOTE_NDIS_INITIALIZE_CMPLT with a status of RNDIS_STATUS_SUCCESS, the device enters the RNDIS-initialized state. If the device is in the RNDIS-data-initialized state when it receives a REMOTE_NDIS_SET_MSG specifying a zero filter value for OID_GEN_CURRENT_PACKET_FILTER, this event forces the device back to the RNDIS-initialized state.

RNDIS_DATA_INITIALIZED If the device receives a REMOTE_NDIS_SET_MSG that specifies a non-zero filter value for OID_GEN_CURRENT_PACKET_FILTER, the device enters the RNDIS-data-initialized state.

3.6.3.7.3 **enum** rndis_event_enum_t

Enumerator

kUSB_DeviceCdcEventAppGetLinkSpeed This event indicates to get the link speed of the Ethernet.

USB CDC Class driver

kUSB_DeviceCdcEventAppGetSendPacketSize This event indicates to get the USB send packet size.

kUSB_DeviceCdcEventAppGetRecvPacketSize This event indicates to get the USB receive packet size.

kUSB_DeviceCdcEventAppGetMacAddress This event indicates to get the mac address of the device.

kUSB_DeviceCdcEventAppGetLinkStatus This event indicates to get the link status of the Ethernet.

kUSB_DeviceCdcEventAppGetMaxFrameSize This event indicates to get the Ethernet maximum frame size.

3.6.3.8 Function Documentation

3.6.3.8.1 **usb_status_t** USB_DeviceCdcRndisInit (**class_handle_t** *classHandle*, **usb_device_cdc_rndis_config_struct_t** * *config*, **usb_device_cdc_rndis_struct_t** ** *handle*)

This function sets the initial value for RNDIS device state, hardware state and media connection status, configures the maximum transmit size and the RNDIS request callback according to the user configuration structure. It also creates the mutex for accessing the device state.

Parameters

<i>classHandle</i>	The class handle of the CDC ACM class.
<i>config</i>	The configure structure of the RNDIS device.
<i>handle</i>	This is a out parameter. It points to the address of the USB CDC RNDIS device handle.

Returns

A USB error code or **kStatus_USB_Success**.

Return values

<i>kStatus_USB_Success</i>	Initialize the RNDIS device successfully.
<i>kStatus_USB_Error</i>	Fails to allocate for the RNDIS device handle.

3.6.3.8.2 **usb_status_t** USB_DeviceCdcRndisDeinit (**usb_device_cdc_rndis_struct_t** * *handle*)

This function destroys the mutex of the device state and frees the RNDIS device handle.

Parameters

<i>handle</i>	This is a pointer to the USB CDC RNDIS device handle.
---------------	-------------------------------------------------------

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	De-Initialize the RNDIS device successfully.
<i>kStatus_USB_Error</i>	Fails to free the RNDIS device handle.
<i>kStatus_USB_Invalid-Handle</i>	The RNDIS device handle is invalid.

3.6.3.8.3 `usb_status_t USB_DeviceCdcRndisMessageSet (usb_device_cdc_rndis_struct_t * handle, uint8_t ** message, uint32_t * len)`

This function checks the message length to see if it exceeds the maximum of the RNDIS request size and sets the device state or prepares notification for various message type accordingly.

Parameters

<i>handle</i>	This is a pointer to the USB CDC RNDIS device handle.
<i>message</i>	This is a pointer to the address of the RNDIS request buffer.
<i>len</i>	This is a pointer to the variable of data size for the RNDIS request.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	Responds to the host successfully.
<i>kStatus_USB_Error</i>	The message length exceeds the maximum of the RNDIS request.
<i>kStatus_USB_Invalid-Handle</i>	The RNDIS device handle is invalid.

USB CDC Class driver

3.6.3.8.4 `usb_status_t USB_DeviceCdcRndisMessageGet (usb_device_cdc_rndis_struct_t *
handle, uint8_t ** message, uint32_t * len)`

This function prepares the response for various message type which is stored in SendEncapsulated-Command.

Parameters

<i>handle</i>	This is a pointer to the USB CDC RNDIS device handle.
<i>message</i>	This is an out parameter. It is a pointer to the address of the RNDIS response buffer.
<i>len</i>	This is an out parameter. It is a pointer to the variable of data size for the RNDIS response.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	Prepares for the response to the host successfully.
<i>kStatus_USB_Invalid-Request</i>	The message type is not supported.
<i>kStatus_USB_Invalid-Handle</i>	The RNDIS device handle is invalid.

3.6.3.8.5 `usb_status_t USB_DeviceCdcRndisResetCommand (usb_device_cdc_rndis_struct_t * handle, uint8_t ** message, uint32_t * len)`

This function is called to soft reset the RNDIS device.

Parameters

<i>handle</i>	This is a pointer to the USB CDC RNDIS device handle.
<i>message</i>	This is an out parameter. It is a pointer to the address of the RNDIS response buffer.
<i>len</i>	This is an out parameter. It is a pointer to the variable of data size for the RNDIS response.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

USB CDC Class driver

<i>kStatus_USB_Success</i>	Prepares for the response to the host successfully.
<i>kStatus_USB_Invalid-Handle</i>	The RNDIS device handle is invalid.

3.6.3.8.6 **usb_status_t USB_DeviceCdcRndisHaltCommand (usb_device_cdc_rndis_struct_t * handle)**

This function is called to halt the RNDIS device.

Parameters

<i>handle</i>	This is a pointer to the USB CDC RNDIS device handle.
---------------	-------------------------------------------------------

Returns

A USB error code or *kStatus_USB_Success*.

Return values

<i>kStatus_USB_Success</i>	Halt the RNDIS device successfully.
<i>kStatus_USB_Invalid-Handle</i>	The RNDIS device handle is invalid.

3.7 USB AUDIO Class driver

3.7.1 Overview

Data Structures

- struct `usb_device_audio_entity_struct_t`
The audio device class-specific information. [More...](#)
- struct `usb_device_audio_entities_struct_t`
The audio device class-specific information list. [More...](#)
- struct `usb_device_audio_struct_t`
The audio device class status structure. [More...](#)

Enumerations

- enum `usb_device_audio_event_t` {
`kUSB_DeviceAudioEventStreamSendResponse = 0x01U`,
`kUSB_DeviceAudioEventStreamRecvResponse`,
`kUSB_DeviceAudioEventControlSendResponse` }
Available common EVENT types in audio class callback.

USB Audio class codes

Enables/disables the Audio Class 2.0

- #define `USB_DEVICE_CONFIG_AUDIO_CLASS_CODE` (0x01)
Audio device class code.
- #define `USB_DEVICE_AUDIO_STREAM_SUBCLASS` (0x02)
Audio device subclass code.
- #define `USB_DEVICE_AUDIO_CONTROL_SUBCLASS` (0x01)
- #define `USB_DESCRIPTOR_TYPE_AUDIO_CS_INTERFACE` (0x24)
Audio device class-specific descriptor type.
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_CONTROL_HEADER` (0x01)
Audio device class-specific control interface descriptor subtype.
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_CONTROL_INPUT_TERMINAL` (0x02)
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_CONTROL_OUTPUT_TERMINAL` (0x03)
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_CONTROL_FEATURE_UNIT` (0x06)
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_GENERAL` (0x01)
Audio device class-specific stream interface descriptor subtype.
- #define `USB_DESCRIPTOR_SUBTYPE_AUDIO_STREAMING_FORMAT_TYPE` (0x02)
- #define `USB_DEVICE_AUDIO_GET_CUR_MUTE_CONTROL` (0x8101)
Audio device class-specific GET CUR COMMAND.
- #define `USB_DEVICE_AUDIO_GET_CUR_VOLUME_CONTROL` (0x8102)
- #define `USB_DEVICE_AUDIO_GET_CUR_BASS_CONTROL` (0x8103)
- #define `USB_DEVICE_AUDIO_GET_CUR_MID_CONTROL` (0x8104)
- #define `USB_DEVICE_AUDIO_GET_CUR_TREBLE_CONTROL` (0x8105)
- #define `USB_DEVICE_AUDIO_GET_CUR_GRAPHIC_EQUALIZER_CONTROL` (0x8106)

- #define **USB_DEVICE_AUDIO_GET_CUR_AUTOMATIC_GAIN_CONTROL** (0x8107)
- #define **USB_DEVICE_AUDIO_GET_CUR_DELAY_CONTROL** (0x8108)
- #define **USB_DEVICE_AUDIO_GET_CUR_BASS_BOOST_CONTROL** (0x8109)
- #define **USB_DEVICE_AUDIO_GET_CUR_LOUDNESS_CONTROL** (0x810A)
- #define **USB_DEVICE_AUDIO_GET_MIN_VOLUME_CONTROL** (0x8202)
 - Audio device class-specific GET MIN COMMAND.*
 - #define **USB_DEVICE_AUDIO_GET_MIN_BASS_CONTROL** (0x8203)
 - #define **USB_DEVICE_AUDIO_GET_MIN_MID_CONTROL** (0x8204)
 - #define **USB_DEVICE_AUDIO_GET_MIN_TREBLE_CONTROL** (0x8205)
 - #define **USB_DEVICE_AUDIO_GET_MIN_GRAPHIC_EQUALIZER_CONTROL** (0x8206)
 - #define **USB_DEVICE_AUDIO_GET_MIN_DELAY_CONTROL** (0x8208)
 - #define **USB_DEVICE_AUDIO_GET_MAX_VOLUME_CONTROL** (0x8302)
 - Audio device class-specific GET MAX COMMAND.*
 - #define **USB_DEVICE_AUDIO_GET_MAX_BASS_CONTROL** (0x8303)
 - #define **USB_DEVICE_AUDIO_GET_MAX_MID_CONTROL** (0x8304)
 - #define **USB_DEVICE_AUDIO_GET_MAX_TREBLE_CONTROL** (0x8305)
 - #define **USB_DEVICE_AUDIO_GET_MAX_GRAPHIC_EQUALIZER_CONTROL** (0x8306)
 - #define **USB_DEVICE_AUDIO_GET_MAX_DELAY_CONTROL** (0x8308)
 - #define **USB_DEVICE_AUDIO_GET_RES_VOLUME_CONTROL** (0x8402)
 - Audio device class-specific GET RES COMMAND.*
 - #define **USB_DEVICE_AUDIO_GET_RES_BASS_CONTROL** (0x8403)
 - #define **USB_DEVICE_AUDIO_GET_RES_MID_CONTROL** (0x8404)
 - #define **USB_DEVICE_AUDIO_GET_RES_TREBLE_CONTROL** (0x8405)
 - #define **USB_DEVICE_AUDIO_GET_RES_GRAPHIC_EQUALIZER_CONTROL** (0x8406)
 - #define **USB_DEVICE_AUDIO_GET_RES_DELAY_CONTROL** (0x8408)
 - #define **USB_DEVICE_AUDIO_SET_CUR_MUTE_CONTROL** (0x0101)
 - Audio device class-specific SET CUR COMMAND.*
 - #define **USB_DEVICE_AUDIO_SET_CUR_VOLUME_CONTROL** (0x0102)
 - #define **USB_DEVICE_AUDIO_SET_CUR_BASS_CONTROL** (0x0103)
 - #define **USB_DEVICE_AUDIO_SET_CUR_MID_CONTROL** (0x0104)
 - #define **USB_DEVICE_AUDIO_SET_CUR_TREBLE_CONTROL** (0x0105)
 - #define **USB_DEVICE_AUDIO_SET_CUR_GRAPHIC_EQUALIZER_CONTROL** (0x0106)
 - #define **USB_DEVICE_AUDIO_SET_CUR_AUTOMATIC_GAIN_CONTROL** (0x0107)
 - #define **USB_DEVICE_AUDIO_SET_CUR_DELAY_CONTROL** (0x0108)
 - #define **USB_DEVICE_AUDIO_SET_CUR_BASS_BOOST_CONTROL** (0x0109)
 - #define **USB_DEVICE_AUDIO_SET_CUR_LOUDNESS_CONTROL** (0x010A)
 - #define **USB_DEVICE_AUDIO_SET_CUR_PITCH_CONTROL** (0x010D)
 - #define **USB_DEVICE_AUDIO_SET_MIN_VOLUME_CONTROL** (0x0202)
 - Audio device class-specific SET MIN COMMAND.*
 - #define **USB_DEVICE_AUDIO_SET_MIN_BASS_CONTROL** (0x0203)
 - #define **USB_DEVICE_AUDIO_SET_MIN_MID_CONTROL** (0x0204)
 - #define **USB_DEVICE_AUDIO_SET_MIN_TREBLE_CONTROL** (0x0205)
 - #define **USB_DEVICE_AUDIO_SET_MIN_GRAPHIC_EQUALIZER_CONTROL** (0x0206)
 - #define **USB_DEVICE_AUDIO_SET_MIN_DELAY_CONTROL** (0x0208)
 - #define **USB_DEVICE_AUDIO_SET_MAX_VOLUME_CONTROL** (0x0302)
 - Audio device class-specific SET MAX COMMAND.*
 - #define **USB_DEVICE_AUDIO_SET_MAX_BASS_CONTROL** (0x0303)
 - #define **USB_DEVICE_AUDIO_SET_MAX_MID_CONTROL** (0x0304)
 - #define **USB_DEVICE_AUDIO_SET_MAX_TREBLE_CONTROL** (0x0305)
 - #define **USB_DEVICE_AUDIO_SET_MAX_GRAPHIC_EQUALIZER_CONTROL** (0x0306)
 - #define **USB_DEVICE_AUDIO_SET_MAX_DELAY_CONTROL** (0x0308)
 - #define **USB_DEVICE_AUDIO_SET_RES_VOLUME_CONTROL** (0x0402)
 - Audio device class-specific SET RES COMMAND.*
 - #define **USB_DEVICE_AUDIO_SET_RES_BASS_CONTROL** (0x0403)
 - #define **USB_DEVICE_AUDIO_SET_RES_MID_CONTROL** (0x0404)

- #define **USB_DEVICE_AUDIO_SET_RES_TREBLE_CONTROL** (0x0405)
- #define **USB_DEVICE_AUDIO_SET_RES_GRAPHIC_EQUALIZER_CONTROL** (0x0406)
- #define **USB_DEVICE_AUDIO_SET_RES_DELAY_CONTROL** (0x0408)
- #define **USB_DEVICE_AUDIO_GET_CUR_SAMPLING_FREQ_CONTROL** (0x810C)
- *Audio device class-specific GET SAMPLING_FREQ CONTROL COMMAND.*
- #define **USB_DEVICE_AUDIO_GET_MIN_SAMPLING_FREQ_CONTROL** (0x820C)
- #define **USB_DEVICE_AUDIO_GET_MAX_SAMPLING_FREQ_CONTROL** (0x830C)
- #define **USB_DEVICE_AUDIO_GET_RES_SAMPLING_FREQ_CONTROL** (0x840C)
- #define **USB_DEVICE_AUDIO_SET_CUR_SAMPLING_FREQ_CONTROL** (0x010C)
- *Audio device class-specific SET SAMPLING_FREQ CONTROL COMMAND.*
- #define **USB_DEVICE_AUDIO_SET_MIN_SAMPLING_FREQ_CONTROL** (0x020C)
- #define **USB_DEVICE_AUDIO_SET_MAX_SAMPLING_FREQ_CONTROL** (0x030C)
- #define **USB_DEVICE_AUDIO_SET_RES_SAMPLING_FREQ_CONTROL** (0x040C)
- #define **USB_DEVICE_AUDIO_SET_CUR_VOLUME_REQUEST** (0x01)
- #define **USB_DEVICE_AUDIO_SET_MIN_VOLUME_REQUEST** (0x02)
- #define **USB_DEVICE_AUDIO_SET_MAX_VOLUME_REQUEST** (0x03)
- #define **USB_DEVICE_AUDIO_SET_RES_VOLUME_REQUEST** (0x04)
- #define **USB_DEVICE_AUDIO_GET_CUR_VOLUME_REQUEST** (0x81)
- #define **USB_DEVICE_AUDIO_GET_MIN_VOLUME_REQUEST** (0x82)
- #define **USB_DEVICE_AUDIO_GET_MAX_VOLUME_REQUEST** (0x83)
- #define **USB_DEVICE_AUDIO_GET_RES_VOLUME_REQUEST** (0x84)
- #define **USB_DEVICE_AUDIO_COPY_PROTECT_CONTROL_SELECTOR** (0x01)
- #define **USB_DEVICE_AUDIO_MUTE_CONTROL_SELECTOR** (0x01)
- #define **USB_DEVICE_AUDIO_VOLUME_CONTROL_SELECTOR** (0x02)
- #define **USB_DEVICE_AUDIO_BASS_CONTROL_SELECTOR** (0x03)
- #define **USB_DEVICE_AUDIO_MID_CONTROL_SELECTOR** (0x04)
- #define **USB_DEVICE_AUDIO_TREBLE_CONTROL_SELECTOR** (0x05)
- #define **USB_DEVICE_AUDIO_GRAPHIC_EQUALIZER_CONTROL_SELECTOR** (0x06)
- #define **USB_DEVICE_AUDIO_AUTOMATIC_GAIN_CONTROL_SELECTOR** (0x07)
- #define **USB_DEVICE_AUDIO_DELAY_CONTROL_SELECTOR** (0x08)
- #define **USB_DEVICE_AUDIO_BASS_BOOST_CONTROL_SELECTOR** (0x09)
- #define **USB_DEVICE_AUDIO_LOUDNESS_CONTROL_SELECTOR** (0x0A)
- #define **USB_DEVICE_AUDIO_SAMPLING_FREQ_CONTROL_SELECTOR** (0x01)
- #define **USB_DEVICE_AUDIO_PITCH_CONTROL_SELECTOR** (0x02)

USB Audio class setup request types

- #define **USB_DEVICE_AUDIO_SET_REQUEST_INTERFACE** (0x21)
- *Audio device class setup request set type.*
- #define **USB_DEVICE_AUDIO_SET_REQUEST_ENDPOINT** (0x22)
- #define **USB_DEVICE_AUDIO_GET_REQUEST_INTERFACE** (0xA1)
- *Audio device class setup request get type.*
- #define **USB_DEVICE_AUDIO_GET_REQUEST_ENDPOINT** (0xA2)

USB Audio Class Driver

- **usb_status_t** **USB_DeviceAudioInit** (uint8_t controllerId, **usb_device_class_config_struct_t** *config, **class_handle_t** *handle)
Initializes the USB audio class.
- **usb_status_t** **USB_DeviceAudioDeinit** (**class_handle_t** handle)
Deinitializes the USB audio class.

USB AUDIO Class driver

- [usb_status_t USB_DeviceAudioEvent](#) (void *handle, uint32_t event, void *param)
Handles the USB audio class event.
- [usb_status_t USB_DeviceAudioSend](#) ([class_handle_t](#) handle, uint8_t ep, uint8_t *buffer, uint32_t length)
Primes the endpoint to send a packet to the host.
- [usb_status_t USB_DeviceAudioRecv](#) ([class_handle_t](#) handle, uint8_t ep, uint8_t *buffer, uint32_t length)
Primes the endpoint to receive a packet from the host.

3.7.2 Data Structure Documentation

3.7.2.1 struct usb_device_audio_entity_struct_t

The structure is used to pass the audio entity information filled by application. Such as entity id (unit or terminal ID), entity type (unit or terminal type), and terminal type if the entity is a terminal.

3.7.2.2 struct usb_device_audio_entities_struct_t

The structure is used to pass the audio entity informations filled by the application. The type of each entity is [usb_device_audio_entity_struct_t](#). The structure pointer is kept in the [usb_device_interface_struct_t::classSpecific](#), such as, if there are three entities (an out terminal, camera terminal, and processing unit), the value of the count field is 3 and the entity field saves the every entity information.

3.7.2.3 struct usb_device_audio_struct_t

Data Fields

- [usb_device_handle](#) handle
The device handle.
- [usb_device_class_config_struct_t](#) * configStruct
The configuration of the class.
- [usb_device_interface_struct_t](#) * controlInterfaceHandle
Current control interface handle.
- [usb_device_interface_struct_t](#) * streamInterfaceHandle
Current stream interface handle.
- uint8_t configuration
Current configuration.
- uint8_t controlInterfaceNumber
The control interface number of the class.
- uint8_t controlAlternate
Current alternate setting of the control interface.
- uint8_t streamInterfaceNumber
The stream interface number of the class.
- uint8_t streamAlternate
Current alternate setting of the stream interface.
- uint8_t streamInPipeBusy

- *Stream IN pipe busy flag.*
uint8_t [streamOutPipeBusy](#)
Stream OUT pipe busy flag.

3.7.2.3.0.1 Field Documentation

3.7.2.3.0.1.1 usb_device_class_config_struct_t* usb_device_audio_struct_t::configStruct

3.7.3 Enumeration Type Documentation

3.7.3.1 enum usb_device_audio_event_t

Enumerator

- kUSB_DeviceAudioEventStreamSendResponse* Send data completed in stream pipe.
kUSB_DeviceAudioEventStreamRecvResponse Data received in stream pipe.
kUSB_DeviceAudioEventControlSendResponse Send data completed in audio control pipe.

3.7.4 Function Documentation

3.7.4.1 usb_status_t USB_DeviceAudioInit (uint8_t *controllerId*, usb_device_class_config_struct_t * *config*, class_handle_t * *handle*)

This function obtains a USB device handle according to the controller ID, initializes the audio class with the class configuration parameters, and creates the mutex for each pipe.

Parameters

<i>controllerId</i>	The ID of the controller. The value can be chosen from the kUSB_ControllerKhci0, kUSB_ControllerKhci1, kUSB_ControllerEhci0, or kUSB_ControllerEhci1.
<i>config</i>	The user configuration structure of type usb_device_class_config_struct_t . The user populates the members of this structure and passes the pointer of this structure into this function.
<i>handle</i>	An out parameter. The class handle of the audio class.

Returns

A USB error code or kStatus_USB_Success.

USB AUDIO Class driver

Return values

<i>kStatus_USB_Success</i>	The audio class is initialized successfully.
<i>kStatus_USB_Busy</i>	No audio device handle available for allocation.
<i>kStatus_USB_Invalid-Handle</i>	The audio device handle allocation failure.
<i>kStatus_USB_Invalid-Parameter</i>	The USB device handle allocation failure.

3.7.4.2 **usb_status_t USB_DeviceAudioDeinit (class_handle_t *handle*)**

This function destroys the mutex for each pipe, deinitializes each endpoint of the audio class, and frees the audio class handle.

Parameters

<i>handle</i>	The class handle of the audio class.
---------------	--------------------------------------

Returns

A USB error code or *kStatus_USB_Success*.

Return values

<i>kStatus_USB_Success</i>	The audio class is deinitialized successfully.
<i>kStatus_USB_Error</i>	The endpoint deinitialization failure.
<i>kStatus_USB_Invalid-Handle</i>	The audio device handle or the audio class handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number of the audio class handle is invalid.

3.7.4.3 **usb_status_t USB_DeviceAudioEvent (void * *handle*, uint32_t *event*, void * *param*)**

This function responds to various events including the common device events and the class-specific events. For class-specific events, it calls the class callback defined in the application to deal with the class-specific event.

Parameters

<i>handle</i>	The class handle of the audio class.
<i>event</i>	The event type.
<i>param</i>	The class handle of the audio class.

Returns

A USB error code or kStatus_USB_Success.

Return values

<i>kStatus_USB_Success</i>	The audio class is deinitialized successfully.
<i>kStatus_USB_Error</i>	The configure structure of the audio class handle is invalid.
<i>kStatus_USB_Invalid-Handle</i>	The audio device handle or the audio class handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number of the audio class handle is invalid.
<i>Others</i>	The error code returned by class callback in application.

3.7.4.4 **usb_status_t USB_DeviceAudioSend (class_handle_t *handle*, uint8_t *ep*, uint8_t * *buffer*, uint32_t *length*)**

This function checks whether the endpoint is sending packet, then it primes the endpoint with the buffer address and the buffer length if the pipe is not busy. Otherwise, it ignores this transfer by returning an error code.

Parameters

<i>handle</i>	The class handle of the audio class.
<i>ep</i>	The endpoint number of the transfer.
<i>buffer</i>	The pointer to the buffer to be transferred.
<i>length</i>	The length of the buffer to be transferred.

Returns

A USB error code or kStatus_USB_Success.

USB AUDIO Class driver

Return values

<i>kStatus_USB_Success</i>	Prime to send packet successfully.
<i>kStatus_USB_Busy</i>	The endpoint is busy in transferring.
<i>kStatus_USB_Invalid-Handle</i>	The audio device handle or the audio class handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	The controller interface is invalid.

3.7.4.5 **usb_status_t USB_DeviceAudioRecv (class_handle_t *handle*, uint8_t *ep*, uint8_t * *buffer*, uint32_t *length*)**

This function checks whether the endpoint is receiving packet, then it primes the endpoint with the buffer address and the buffer length if the pipe is not busy. Otherwise, it ignores this transfer by returning an error code.

Parameters

<i>handle</i>	The class handle of the audio class.
<i>ep</i>	The endpoint number of the transfer.
<i>buffer</i>	The pointer to the buffer to be transferred.
<i>length</i>	The length of the buffer to be transferred.

Returns

A USB error code or *kStatus_USB_Success*.

Return values

<i>kStatus_USB_Success</i>	Prime to receive packet successfully.
<i>kStatus_USB_Busy</i>	The endpoint is busy in transferring.
<i>kStatus_USB_Invalid-Handle</i>	The audio device handle or the audio class handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	The controller interface is invalid.

3.8 USB CCID Class driver

3.8.1 Overview

Data Structures

- struct `usb_device_ccid_common_command_t`
Common command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_power_on_command_t`
ICC power on command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_power_off_command_t`
ICC power off command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_get_slot_status_command_t`
Gets the slot status command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_transfer_block_command_t`
Transfer data block command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_get_parameters_command_t`
Gets the ICC parameter command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_reset_parameters_command_t`
Resets the ICC parameter command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_set_parameters_command_t`
Sets the ICC parameter command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_set_parameters_t0_command_t`
Sets the ICC(T=0) parameter command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_set_parameters_t1_command_t`
Sets the ICC(T=1) parameter command structure of the command message in the bulk-out pipe. [More...](#)
- union `usb_device_ccid_set_parameters_command_common_t`
Sets the ICC parameter command union of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_escape_command_t`
Escape command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_clock_command_t`
Controls the ICC clock command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_t0_apdu_command_t`
Controls the ICC clock command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_secure_command_t`
Secures the command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_secure_pin_operation_command_t`
Secures the PIN operation command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_seucre_pin_verification_command_t`
Secures the PIN verification operation command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_secure_pin_modification_command_t`
Secures the PIN modification operation command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_mechanical_command_t`
Manages the motorized type CCID functionality command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_abort_command_t`
Aborts the command structure of the command message in the bulk-out pipe. [More...](#)
- struct `usb_device_ccid_set_data_rate_and_clock_frequency_command_t`
Sets data rate and clock frequency command structure of the command message in the bulk-out pipe.

USB CCID Class driver

[More...](#)

- struct `usb_device_ccid_common_response_t`
Common response structure to respond a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_data_block_response_t`
Data block response structure to respond a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_slot_status_response_t`
Sends a slot status response structure to respond a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_parameters_response_t`
ICC parameter response structure to respond a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_parameters_T0_response_t`
ICC T0 parameter response structure to respond a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_parameters_T1_response_t`
ICC T1 parameter response structure to response a command message in the bulk-in pipe. [More...](#)
- union `usb_device_ccid_parameters_response_common_t`
ICC parameter response union to response a command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_escape_response_t`
Response structure to respond the "PC_to_RDR_Escape" command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_data_rate_and_clock_frequency_response_t`
Response structure to respond the "PC_to_RDR_SetDataRateAndClockFrequency" command message in the bulk-in pipe. [More...](#)
- struct `usb_device_ccid_notify_slot_chnage_notification_t`
Notification structure to notify Host the CCID device slot changed. [More...](#)
- struct `usb_device_ccid_hardware_error_notification_t`
Notification structure to notify Host a hardware error happened in the CCID device. [More...](#)
- struct `usb_device_ccid_transfer_struct_t`
USB device CCID transfer structure. [More...](#)
- struct `usb_device_ccid_control_request_struct_t`
The structure is used to get data rates or clock frequencies if the event is `kUSB_DeviceCcidEventGetClockFrequencies` or `kUSB_DeviceCcidEventGetDataRate`. [More...](#)
- struct `usb_device_ccid_notification_struct_t`
The structure is used to keep the transferred buffer and transferred length if the event is `kUSB_DeviceCcidEventSlotChangeSent` or `kUSB_DeviceCcidEventHardwareErrorSent`. [More...](#)
- struct `usb_device_ccid_command_struct_t`
The structure is used to keep the command data and length and get response data and length if the event is `kUSB_DeviceCcidEventCommandReceived`. [More...](#)
- struct `usb_device_ccid_slot_status_struct_t`
The structure is used to get the slot status if the event is `kUSB_DeviceCcidEventGetSlotStatus`. [More...](#)
- struct `usb_device_ccid_struct_t`
The CCID device class status structure. [More...](#)

Macros

- #define `USB_DEVICE_CCID_CLASS_CODE` (0x0BU)
CCID device class code.
- #define `USB_DEVICE_CCID_SUBCLASS_CODE` (0x00U)
CCID device subclass code.
- #define `USB_DEVICE_CCID_PROTOCOL_CODE` (0x00U)
CCID device protocol code.
- #define `USB_DEVICE_CCID_ABORT` (0x01U)
CCID device class-specific control pipe requests.

- `#define USB_DEVICE_CCID_PC_TO_RDR_ICCPOWERON (0x62U)`
The message type of CCID device class-specific bulk-out pipe (Command pipe)
- `#define USB_DEVICE_CCID_RDR_TO_PC_DATABLOCK (0x80U)`
The message type of CCID device class-specific bulk-in pipe (Response pipe)
- `#define USB_DEVICE_CCID_RDR_TO_PC_NOTIFYSLOTCHANGE (0x50U)`
The message type of CCID device class-specific interrupt-in pipe.
- `#define USB_DEVICE_CCID_SLOT_ERROR_COMMAND_NOT_SUPPORTED (0x00U)`
Reporting slot error and slot status registers in bulk-in messages.
- `#define USB_DEVICE_CCID_COMMAND_HEADER_LENGTH (0x0AU)`
The command header length of the bulk-out pipe message.
- `#define USB_DEVICE_CCID_RESPONSE_HEADER_LENGTH (0x0AU)`
The response header length of the bulk-in pipe message.
- `#define USB_DEVICE_CCID_BUFFER_4BYTE_ALIGN(n) (((n - 1U) & 0xFFFFFFFFCU) + 0x00000004U)`
The definition to make the length aligned to 4-bytes.

Enumerations

- `enum usb_device_ccid_event_t {`
`kUSB_DeviceCcidEventCommandReceived = 0x01U,`
`kUSB_DeviceCcidEventResponseSent,`
`kUSB_DeviceCcidEventGetSlotCount,`
`kUSB_DeviceCcidEventGetSlotStatus,`
`kUSB_DeviceCcidEventCommandAbort,`
`kUSB_DeviceCcidEventGetClockFrequencies,`
`kUSB_DeviceCcidEventGetDataRate,`
`kUSB_DeviceCcidEventSlotChangeSent,`
`kUSB_DeviceCcidEventHardwareErrorSent }`
Available common EVENT types in CCID class callback.
- `enum usb_device_ccid_slot_state_t {`
`kUSB_DeviceCcidSlotStateNoPresent = 0x00U,`
`kUSB_DeviceCcidSlotStatePresent = 0x01U }`
Slot status, present or not.
- `enum usb_device_ccid_hardware_error_t { kUSB_DeviceCcidHardwareErrorOverCurrent = 0x01-U }`
Hardware error status.

Functions

- `usb_status_t USB_DeviceCcidInit (uint8_t controllerId, usb_device_class_config_struct_t *config, class_handle_t *handle)`
Initialize the CCID class.
- `usb_status_t USB_DeviceCcidDeinit (class_handle_t handle)`
Deinitializes the device CCID class.
- `usb_status_t USB_DeviceCcidEvent (void *handle, uint32_t event, void *param)`
Handles the event passed to the CCID class.

USB CCID device class configuration

- #define **USB_DEVICE_CONFIG_CCID_SLOT_MAX** (1U)
MAX slot number of the CCID device.
- #define **USB_DEVICE_CONFIG_CCID_TRANSFER_COUNT** (4U)
MAX transfer entity number of the CCID device.
- #define **USB_DEVICE_CONFIG_CCID_MAX_MESSAGE_LENGTH** (271U)
MAX maximum message length of the CCID device.

USB CCID device class descriptor

- #define **USB_DEVICE_CCID_DESCRIPTOR_LENGTH** (0x36U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_TYPE** (0x21U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_VOLTAGE_SUPPORT_BM_5V** (0x01U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_VOLTAGE_SUPPORT_BM_3V** (0x02U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_VOLTAGE_SUPPORT_BM_1V8** (0x04U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_PROTOCOLS_BM_T0** (0x00000001U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_PROTOCOLS_BM_T1** (0x00000002U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_MECHANICAL_BM_NO** (0x00000000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_MECHANICAL_BM_ACCEPT** (0x00000001-U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_MECHANICAL_BM_EJECTION** (0x00000002-U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_MECHANICAL_BM_CAPTURE** (0x00000004-U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_MECHANICAL_BM_LOCK_UNLOCK** (0x00000008U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_NO** (0x00000000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_CONFIG_BASED_ON_ATR** (0x00000002U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_ACTIVE_ON_INSERTING** (0x00000004U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_VOLTAGE_SELECTION** (0x00000008U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_FREQUENCY_CHANGE** (0x00000010U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_BAUD_RATE_CHANGE** (0x00000020U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_NEGOTIATION** (0x00000040U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_PPS** (0x00000080-U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_CAN_SET_IN_STOP_MODE** (0x00000100U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_NAD_VLAUE** (0x00000200-U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_AUTO_IFSD_EXCHANGE_AS_FIRST** (0x00000400U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_TPDU_LEVEL_EXCHA-**

- NGES (0x00010000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_SHORT_APDU_LEVEL_EXCHANGES** (0x00020000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_SHORT_EXTENDED_APDU_LEVEL_EXCHANGES** (0x00040000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_FEATURES_BM_SUPPORT_SUPPEND** (0x00100000U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_PIN_SUPPORT_BM_NO** (0x00U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_PIN_SUPPORT_BM_VERIFICATION_SUPPORTED** (0x01U)
- #define **USB_DEVICE_CCID_DESCRIPTOR_PIN_SUPPORT_BM_MODIFICATION_SUPPORTED** (0x02U)

USB device CCID class APIs

- [usb_status_t USB_DeviceCcidNotifySlotChange](#) ([class_handle_t](#) handle, [uint8_t](#) slot, [usb_device_ccid_slot_state_t](#) state)
Notifies the slot status changed.
- [usb_status_t USB_DeviceCcidNotifyHardwareError](#) ([class_handle_t](#) handle, [uint8_t](#) slot, [usb_device_ccid_hardware_error_t](#) errorCode)
Notifies the slot status changed.

3.8.2 Data Structure Documentation

3.8.2.1 struct_usb_device_ccid_common_command

Data Fields

- [uint8_t bMessageType](#)
The message type.
- [uint32_t dwLength](#)
Message-specific data length.
- [uint8_t bSlot](#)
Identifies the slot number for this command.
- [uint8_t bSeq](#)
Sequence number for command.
- [uint8_t bParameter1](#)
Parameter1 of the message, message-specific.
- [uint8_t bParameter2](#)
Parameter2 of the message, message-specific.
- [uint8_t bParameter3](#)
Parameter3 of the message, message-specific.

3.8.2.2 struct_usb_device_ccid_power_on_command

A PC_to_RDR_IccPowerOn message to an inactive slot returns an Answer-To-Reset (ATR) data.

USB CCID Class driver

The response to this command message is the RDR_to_PC_DataBlock response message and the data returned is the Answer To Reset (ATR) data.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bPowerSelect**
Voltage that is applied to the ICC.
- uint8_t **BRFU** [2]
Reserved for Future Use.

3.8.2.3 struct _usb_device_ccid_power_off_command

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **BRFU** [3]
Reserved for Future Use.

3.8.2.4 struct _usb_device_ccid_get_slot_status_command

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**

- `uint8_t bSeq`
Identifies the slot number for this command.
- `uint8_t bRFU [3]`
Reserved for Future Use.

3.8.2.5 struct_usb_device_ccid_transfer_block_command

The block should never exceed the `dwMaxCCIDMessageLength-10` in the Class Descriptor. Parameter `bbWI` is only used by CCIDs which use the character level and TPDU level of exchange (as reported in the `dwFeatures` parameter in the CCID Functional Descriptor) and only for protocol `T=1` transfers.

The response to this command message is the `RDR_to_PC_DataBlock` response message.

Note

For reference, the absolute maximum block size for a TPDU `T=0` block is 260U bytes (5U bytes command; 255U bytes data), or for a TPDU `T=1` block is 259U bytes, or for a short APDU `T=1` block is 261U bytes, or for an extended APDU `T=1` block is 65544U bytes.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
Size of `abData` field of this message.
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number for command.
- `uint8_t bbWI`
Used to extend the CCIDs Block Waiting Timeout for this current transfer.
- `uint16_t wLevelParameter`
Use changes depending on the exchange level reported by the class descriptor in `dwFeatures` field.
- `uint8_t abData [1]`
Data block sent to the CCID.

3.8.2.6 struct_usb_device_ccid_get_parameters_command

The response to this command message is the `RDR_to_PC_Parameters` response message.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
Message-specific data length.

USB CCID Class driver

- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **BRFU** [3]
Reserved for Future use.

3.8.2.7 struct _usb_device_ccid_reset_parameters_command

This command resets the slot parameters to their default values.

The response to this command message is the RDR_to_PC_Parameters response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **BRFU** [3]
Reserved for Future Use.

3.8.2.8 struct _usb_device_ccid_set_parameters_command

This command is used to change the parameters for a given slot. A CCID which has no automatic features (dwFeatures=0, 100h, 200h, or 300h) depends on the driver to send this command to set the protocol and other parameters to the right values necessary to correctly talk to the ICC located in the selected slot. A CCID which has automatic features automatically sets the protocol and certain parameters based on data received from the ICC (ATR, PPS, IFSD, or proprietary algorithms). The level of automatism and design requirements determines which parameters the CCID allow the driver to change. If this command tries to change a parameter which is not changeable, then the CCID does not change any parameters and the RDR_to_PC_GetParameters response returns a Command Failed status and the bError field contains the offset of the "offending" parameter.

The response to this command message is the RDR_to_PC_Parameters response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Size of abProtocolDataStructure field of this message.
- uint8_t **bSlot**

- `uint8_t bSeq`
Identifies the slot number for this command.
- `uint8_t bProtocolNum`
Sequence number for command.
- `uint8_t bRFU` [2]
Specifies what protocol data structure follows.
- `uint8_t abProtocolDataStructure` [1]
Reserved for Future Use.
Protocol Data Structure.

3.8.2.8.0.2 Field Documentation

3.8.2.8.0.2.1 `uint8_t usb_device_ccid_set_parameters_command_t::bProtocolNum`

00h = Structure for protocol T=0, 01h = Structure for protocol T=1

3.8.2.8.0.2.2 `uint8_t usb_device_ccid_set_parameters_command_t::abProtocolDataStructure[1]`

For T = 0U, see `usb_device_ccid_set_parameters_t0_command_t`, for T = 1U, see `usb_device_ccid_set_parameters_t1_command_t`.

3.8.2.9 `struct usb_device_ccid_set_parameters_t0_command`

Protocol Data Structure for Protocol T=0 (`bProtocolNum=0`) (`dwLength=00000005h`).

The response to this command message is the `RDR_to_PC_Parameters` response message.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
(dwLength = 0x05U)
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number for command.
- `uint8_t bProtocolNum`
Structure for protocol T=0.
- `uint8_t bRFU` [2]
Reserved for Future Use.
- `uint8_t bmFindexDindex`
Bit7~4 - Fi, Bit3~0 - Di.
- `uint8_t bmTCKST0`
Bit1 - Convention used(0U for direct, 1U for inverse), other bits is 0.
- `uint8_t bGuardTimeT0`
Extra guard time between two characters.
- `uint8_t bWaitingIntegerT0`
WI for T= 0U used to define WWT.

USB CCID Class driver

- uint8_t **bClockStop**
ICC Clock Stop Support.

3.8.2.9.0.3 Field Documentation

3.8.2.9.0.3.1 uint8_t usb_device_ccid_set_parameters_t0_command_t::bmFindexDindex

3.8.2.9.0.3.2 uint8_t usb_device_ccid_set_parameters_t0_command_t::bGuardTimeT0

3.8.2.10 struct _usb_device_ccid_set_parameters_t1_command

Protocol Data Structure for Protocol T=1 (bProtocolNum=1) (dwLength=00000007h)

The response to this command message is the RDR_to_PC_Parameters response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
(dwLength = 0x07U)
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bProtocolNum**
Structure for protocol T=1.
- uint8_t **brFU** [2]
Reserved for Future Use.
- uint8_t **bmFindexDindex**
Bit7~4 - Fi, Bit3~0 - Di.
- uint8_t **bmTCKST1**
Bit0 - Checksum type(0U for LRC, 1U for CRC).
- uint8_t **bGuardTimeT1**
Extra guard time.
- uint8_t **bmWaitingIntegersT1**
Bit7~4 - BWI(0~9 valid), Bit3~0 - CWI(0~0xF valid)
- uint8_t **bClockStop**
ICC Clock Stop Support.
- uint8_t **bIFSC**
Size of negotiated IFSC.
- uint8_t **bNadValue**
Value = 0x00U if CCID doesn't support a value other than the default value.

3.8.2.10.0.4 Field Documentation

3.8.2.10.0.4.1 `uint8_t usb_device_ccid_set_parameters_t1_command_t::bmFindexIndex`

3.8.2.10.0.4.2 `uint8_t usb_device_ccid_set_parameters_t1_command_t::bmTCCKST1`

Bit1 - Convention used(0U for direct, 1U for inverse), Bit7~2 - 0b000100

3.8.2.10.0.4.3 `uint8_t usb_device_ccid_set_parameters_t1_command_t::bGuardTimeT1`

3.8.2.10.0.4.4 `uint8_t usb_device_ccid_set_parameters_t1_command_t::bNadValue`

Else value respects ISO/IEC 7816-3, 9.4.2.1

3.8.2.11 `union usb_device_ccid_set_parameters_command_common_t`

Data Fields

- `usb_device_ccid_set_parameters_command_t common`
Set ICC parameter common structure.
- `usb_device_ccid_set_parameters_t0_command_t t0`
Set ICC parameter structure for T0.
- `usb_device_ccid_set_parameters_t1_command_t t1`
Set ICC parameter structure for T1.

3.8.2.12 `struct _usb_device_ccid_escape_command`

This command allows the CCID manufacturer to define and access extended features. Information sent via this command is processed by the CCID control logic.

The response to this command message is the RDR_to_PC_Escape response message.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
Message-specific data length.
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number for command.
- `uint8_t bRFU [3]`
Reserved for future use.
- `uint8_t abData [1]`
Size of abData field of this message.

3.8.2.13 struct _usb_device_ccid_clock_command

This command stops or restarts the clock.

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bClockCommand**
0x00U - Restart clock, 0x01U - Stop clock in the state shown in the bClockStop field of the PC_to_RDR_SetParameters command and RDR_to_PC_Parameters message.
- uint8_t **brFU** [2]
Reserved for future use.

3.8.2.13.0.5 Field Documentation

3.8.2.13.0.5.1 uint8_t usb_device_ccid_clock_command_t::bClockCommand

3.8.2.14 struct _usb_device_ccid_t0_apdu_command

This command changes the parameters used to perform the transportation of APDU messages by the T=0 protocol. It effects the CLA (class) byte used when issuing a Get Response command or a Envelope command to the ICC.

This command is slot-specific. It only effects the slot specified in the bSlot field. Slots, when not powered, do not change back to using the default behaviour defined in the CCID class descriptor. Any newly inserted ICC has the default behaviour until this command is issued for its slot.

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bmChanges**
The value is bitwise OR operation.

- uint8_t **bClassGetResponse**
Value to force the class byte of the header in a get response command.
- uint8_t **bClassEnvelope**
Value to force the class byte of the header in a envelope command.

3.8.2.14.0.6 Field Documentation

3.8.2.14.0.6.1 uint8_t usb_device_ccid_t0_apdu_command_t::bmChanges

Bit 0U is associated with field bClassGetResponse Bit 1U is associated with field bClassEnvelope Other bits are RFU.

3.8.2.15 struct _usb_device_ccid_secure_command

This is a command message to allow entering the PIN for verification or modification.

The response to this command message is the RDR_to_PC_DataBlock response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Size of abData field of this message.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bbWI**
Used to extend the CCIDs Block Waiting Timeout for this current transfer.
- uint16_t **wLevelParameter**
Use changes depending on the exchange level reported by CCID in the functional descriptor.
- uint8_t **abData** [1]
The value depends of wLevelParameters.

3.8.2.15.0.7 Field Documentation

3.8.2.15.0.7.1 uint8_t usb_device_ccid_secure_command_t::abData[1]

When wLevelParameters is 0000h or 0001h abData = abPINOperationDataStructure. For other values of wLevelParameters this field is the continuation of the previously sent PC_to_RDR_Secure.

3.8.2.16 struct _usb_device_ccid_secure_pin_operation_command

This is a command message to allow entering the PIN for verification or modification.

The response to this command message is the RDR_to_PC_DataBlock response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
1U + Size of abPINDataStructure field of this message
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bbWI**
Used to extend the CCIDs Block Waiting Timeout for this current transfer.
- uint16_t **wLevelParameter**
Use changes depending on the exchange level reported by CCID in the functional descriptor.
- uint8_t **bPINOperation**
Used to indicate the PIN operation: 00h: PIN Verification 01h: PIN Modification 02h: Transfer PIN from secure CCID buffer 03h: Wait ICC response 04h: Cancel PIN function 05h: Re-send last I-Block, valid only if T = 1.

3.8.2.16.0.8 Field Documentation

3.8.2.16.0.8.1 uint8_t usb_device_ccid_secure_pin_operation_command_t::bPINOperation

06h: Send next part of APDU, valid only T = 1.

3.8.2.17 struct_usb_device_ccid_seucre_pin_verification_command

This is a command message to allow entering the PIN for verification.

The response to this command message is the RDR_to_PC_DataBlock response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
12U + Size of abPINApdu field of this message
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bbWI**
Used to extend the CCIDs Block Waiting Timeout for this current transfer.
- uint16_t **wLevelParameter**
Use changes depending on the exchange level reported by CCID in the functional descriptor.
- uint8_t **bPINOperation**
Used to indicate the PIN operation: 00h: PIN Verification 01h: PIN Modification 02h: Transfer PIN from secure CCID buffer 03h: Wait ICC response 04h: Cancel PIN function 05h: Re-send last I-Block, valid only if T = 1.

- `uint8_t bTimeOut`
Number of seconds.
- `uint8_t bmFormatString`
Several parameters for the PIN format options.
- `uint8_t bmPINBlockString`
Defines the length in bytes of the PIN block to present in the APDU command.
- `uint8_t bmPINLengthFormat`
Allows the insertion of the PIN length in the APDU command.
- `uint16_t wPINMaxExtraDigit`
Bit15~8 - Minimum PIN size in digit, Bit7~0 - Maximum PIN size in digit.
- `uint8_t bEntryValidationCondition`
The value is a bit wise OR operation.
- `uint8_t bNumberMessage`
Number of messages to display for the PIN Verification management.
- `uint16_t wLangId`
Language used to display the messages.
- `uint8_t bMsgIndex`
Message index in the Reader CCID message table (should be 00h).
- `uint8_t bTeoPrologue`
T=1 I-block prologue field to use.
- `uint8_t abPINApdu [1]`
APDU to send to the ICC.

3.8.2.17.0.9 Field Documentation

3.8.2.17.0.9.1 `uint8_t usb_device_ccid_seucre_pin_verification_command_t::bPINOperation`

06h: Send next part of APDU, valid only T = 1.

3.8.2.17.0.9.2 `uint8_t usb_device_ccid_seucre_pin_verification_command_t::bEntryValidationCondition`

01h - Maximum size reached, 02h - Validation key pressed, 04h - Timeout occurred

3.8.2.17.0.9.3 `uint8_t usb_device_ccid_seucre_pin_verification_command_t::bNumberMessage`

3.8.2.17.0.9.4 `uint16_t usb_device_ccid_seucre_pin_verification_command_t::wLangId`

3.8.2.17.0.9.5 `uint8_t usb_device_ccid_seucre_pin_verification_command_t::bMsgIndex`

3.8.2.17.0.9.6 `uint8_t usb_device_ccid_seucre_pin_verification_command_t::bTeoPrologue`

3.8.2.18 `struct usb_device_ccid_secure_pin_modification_command`

This is a command message to allow entering the PIN for modification.

The response to this command message is the `RDR_to_PC_DataBlock` response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
20U + Size of abPINApdu field of this message
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **bbWI**
Used to extend the CCIDs Block Waiting Timeout for this current transfer.
- uint16_t **wLevelParameter**
Use changes depending on the exchange level reported by CCID in the functional descriptor.
- uint8_t **bPINOperation**
Used to indicate the PIN operation: 00h: PIN Verification 01h: PIN Modification 02h: Transfer PIN from secure CCID buffer 03h: Wait ICC response 04h: Cancel PIN function 05h: Re-send last I-Block, valid only if T = 1.
- uint8_t **bTimeOut**
Number of seconds.
- uint8_t **bmFormatString**
Several parameters for the PIN format options.
- uint8_t **bmPINBlockString**
Define the length of the PIN to present in the APDU command.
- uint8_t **bmPINLengthFormat**
Allows the length PIN insertion in the APDU command.
- uint8_t **bInsertionOffsetOld**
Insertion position offset in byte for the current PIN.
- uint8_t **bInsertionOffsetNew**
Insertion position offset in byte for the new PIN.
- uint16_t **wPINMaxExtraDigit**
Bit15~8 - Minimum PIN size in digit, Bit7~0 - Maximum PIN size in digit.
- uint8_t **bConfirmPIN**
Indicates if a confirmation is requested before acceptance of a new PIN.
- uint8_t **bEntryValidationCondition**
The value is a bit wise OR operation.
- uint8_t **bNumberMessage**
Number of messages to display for the PIN Verification management.
- uint16_t **wLangId**
Language used to display the messages.
- uint8_t **bMsgIndex1**
Message index in the Reader message table(should be 00h or 01h).
- uint8_t **bMsgIndex2**
Message index in the Reader message table(should be 01h or 02h).
- uint8_t **bMsgIndex3**
Message index in the Reader message table(should be 02h).
- uint8_t **bTeoPrologue** [3]
T=1 I-block prologue field to use.
- uint8_t **abPINApdu** [1]
APDU to send to the ICC.

3.8.2.18.0.10 Field Documentation

3.8.2.18.0.10.1 uint8_t usb_device_ccid_secure_pin_modification_command_t::bPINOperation

06h: Send next part of APDU, valid only T = 1.

3.8.2.18.0.10.2 uint8_t usb_device_ccid_secure_pin_modification_command_t::bEntryValidation-Condition

01h - Maximum size reached, 02h - Validation key pressed, 04h - Timeout occurred

3.8.2.18.0.10.3 uint8_t usb_device_ccid_secure_pin_modification_command_t::bNumber-Message

3.8.2.18.0.10.4 uint16_t usb_device_ccid_secure_pin_modification_command_t::wLangId

3.8.2.18.0.10.5 uint8_t usb_device_ccid_secure_pin_modification_command_t::bMsgIndex1

3.8.2.18.0.10.6 uint8_t usb_device_ccid_secure_pin_modification_command_t::bMsgIndex2

3.8.2.18.0.10.7 uint8_t usb_device_ccid_secure_pin_modification_command_t::bMsgIndex3

3.8.2.18.0.10.8 uint8_t usb_device_ccid_secure_pin_modification_command_t::bTeoPrologue[3]

3.8.2.19 struct_usb_device_ccid_mechanical_command

This command is used to manage motorized type CCID functionality.

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t [bMessageType](#)
The message type.
- uint32_t [dwLength](#)
Message-specific data length.
- uint8_t [bSlot](#)
Identifies the slot number for this command.
- uint8_t [bSeq](#)
Sequence number for command.
- uint8_t [bFunction](#)
This value corresponds to the mechanical function being requested.
- uint8_t [bRFU](#) [2]
Reserved for Future Use.

3.8.2.20 struct_usb_device_ccid_abort_command

This command is used with the control pipe abort request to tell the CCID to stop any current transfer at the specified slot and return to a state where the slot is ready to accept a new command pipe Bulk-OUT

USB CCID Class driver

message.

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **BRFU** [3]
Reserved for future use.

3.8.2.21 struct _usb_device_ccid_set_data_rate_and_clock_frequency_command

This command is used to manually set the data rate and clock frequency of a specific slot.

The response to this command message is the RDR_to_PC_SlotStatus response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length(8U bytes)
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for command.
- uint8_t **BRFU** [3]
Reserved for Future Use.
- uint32_t **dwClockFrequency**
ICC clock frequency in kHz.
- uint32_t **dwDataRate**
ICC data rate in BPD.

3.8.2.21.0.11 Field Documentation

3.8.2.21.0.11.1 uint32_t usb_device_ccid_set_data_rate_and_clock_frequency_command_t::dw-ClockFrequency

This is an integer value

3.8.2.22 struct _usb_device_ccid_common_response

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.
- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **bParameter1**
Parameter1 of the message, message-specific.

3.8.2.23 struct _usb_device_ccid_data_block_response

The device in response to the following command messages: "PC_to_RDR_IccPowerOn", "PC_to_RDR-_Secure" and "PC_to_RDR_XfrBlock" sends this response message. For "PC_to_RDR_IccPowerOn" this response message is the answer to reset (ATR) data associated with the ICC power on. In other use cases, the response message has the following format: the response data contains the optional data returned by the ICC, followed by the 2U byte-size status words SW1-SW2.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.
- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **bChainParameter**
Use changes depending on the exchange level reported by the class descriptor in dwFeatures field.
- uint8_t **abData** [1]
This field contains the data returned by the CCID.

3.8.2.23.0.12 Field Documentation

3.8.2.23.0.12.1 uint8_t usb_device_ccid_data_block_response_t::abData[1]

3.8.2.24 struct_usb_device_ccid_slot_status_response

The device in response to the following command messages: "PC_to_RDR_IccPowerOff", "PC_to_RDR_GetSlotStatus", "PC_to_RDR_IccClock", "PC_to_RDR_T0APDU" and, "PC_to_RDR_Mechanical" sends this response message. Also, the device sends this response message when it has completed aborting a slot after receiving both the Class Specific ABORT request and PC_to_RDR_Abort command message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.
- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **bClockStatus**
0x00U - Clock running, 0x01U - Clock stopped in L, 0x02U - clock stopped in H, and 0x03U - clock stopped in an unknown state.

3.8.2.24.0.13 Field Documentation

3.8.2.24.0.13.1 uint8_t usb_device_ccid_slot_status_response_t::bClockStatus

3.8.2.25 struct_usb_device_ccid_parameters_response

The device in response to the following command messages: "PC_to_RDR_GetParameters", "PC_to_RDR_ResetParameters", and, "PC_to_RDR_SetParameters" sends this response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Size of abProtocolDataStructure field of this message.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.

- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **bProtocolNum**
0x00U = Structure for protocol T=0, 0x01U = Structure for protocol T=1
- uint8_t **abProtocolDataStructure** [1]
Protocol Data Structure.

3.8.2.26 struct _usb_device_ccid_parameters_T0_response

The device in response to the following command messages: "PC_to_RDR_GetParameters", "PC_to_RDR_ResetParameters", and, "PC_to_RDR_SetParameters" sends this response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
The value is 0x05U.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.
- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **bProtocolNum**
0x00U = Structure for protocol T=0
- uint8_t **bmFindexDindex**
Bit7~4 - Fi, Bit3~0 - Di.
- uint8_t **bmTCKKST0**
Bit1 - Convention used(0U for direct, 1U for inverse), other bits is 0.
- uint8_t **bGuardTimeT0**
Extra guard time between two characters.
- uint8_t **bWaitingIntegerT0**
WI for T= 0U used to define WWT.
- uint8_t **bClockStop**
ICC Clock Stop Support.

USB CCID Class driver

3.8.2.26.0.14 Field Documentation

3.8.2.26.0.14.1 `uint8_t usb_device_ccid_parameters_T0_response_t::bmFindexDindex`

3.8.2.26.0.14.2 `uint8_t usb_device_ccid_parameters_T0_response_t::bGuardTimeT0`

3.8.2.27 `struct_usb_device_ccid_parameters_T1_response`

The device in response to the following command messages: "PC_to_RDR_GetParameters", "PC_to_RDR_ResetParameters", and, "PC_to_RDR_SetParameters" sends this response message.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
The value is 0x07U.
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number for the corresponding command.
- `uint8_t bStatus`
Slot status register.
- `uint8_t bError`
Slot error register.
- `uint8_t bProtocolNum`
0x00U = Structure for protocol T=1
- `uint8_t bmFindexDindex`
Bit7~4 - Fi, Bit3~0 - Di.
- `uint8_t bmTCCKST1`
Bit0 - Checksum type(0U for LRC, 1U for CRC).
- `uint8_t bGuardTimeT1`
Extra guard time.
- `uint8_t bmWaitingIntegersT1`
Bit7~4 - BWI(0~9 valid), Bit3~0 - CWI(0~0xF valid)
- `uint8_t bClockStop`
ICC Clock Stop Support.
- `uint8_t bIFSC`
Size of negotiated IFSC.
- `uint8_t bNadValue`
Value = 0x00U if CCID doesn't support a value other than the default value.

3.8.2.27.0.15 Field Documentation

3.8.2.27.0.15.1 `uint8_t usb_device_ccid_parameters_T1_response_t::bmFindexDindex`

3.8.2.27.0.15.2 `uint8_t usb_device_ccid_parameters_T1_response_t::bmTCCKST1`

Bit1 - Convention used(0U for direct, 1U for inverse), Bit7~2 - 0b000100

3.8.2.27.0.15.3 `uint8_t usb_device_ccid_parameters_T1_response_t::bGuardTimeT1`

3.8.2.27.0.15.4 `uint8_t usb_device_ccid_parameters_T1_response_t::bNadValue`

Else value respects ISO/IEC 7816-3, 9.4.2.1

3.8.2.28 `union usb_device_ccid_parameters_response_common_t`

Data Fields

- `usb_device_ccid_parameters_response_t common`
Response ICC parameter common structure.
- `usb_device_ccid_parameters_T0_response_t t0`
Response ICC parameter structure for T0.
- `usb_device_ccid_parameters_T1_response_t t1`
Response ICC parameter structure for T1.

3.8.2.29 `struct _usb_device_ccid_escape_response`

The device in response to the following command messages: "PC_to_RDR_Escape" sends this response message.

Data Fields

- `uint8_t bMessageType`
The message type.
- `uint32_t dwLength`
Size of abData field of this message.
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number for the corresponding command.
- `uint8_t bStatus`
Slot status register.
- `uint8_t bError`
Slot error register.
- `uint8_t bRFU`
Reserved for Future Use.
- `uint8_t abData [1]`
Data sent from CCID.

3.8.2.30 `struct _usb_device_ccid_data_rate_and_clock_frequency_response`

The device in response to the following command messages: "PC_to_RDR_SetDataRateAndClock-Frequency" sends this response message.

Data Fields

- uint8_t **bMessageType**
The message type.
- uint32_t **dwLength**
Message-specific data length.
- uint8_t **bSlot**
Identifies the slot number for this command.
- uint8_t **bSeq**
Sequence number for the corresponding command.
- uint8_t **bStatus**
Slot status register.
- uint8_t **bError**
Slot error register.
- uint8_t **brFU**
Reserved for Future Use.
- uint32_t **dwClockFrequency**
Current setting of the ICC clock frequency in KHz.
- uint32_t **dwDataRate**
Current setting of the ICC data rate in bps.

3.8.2.30.0.16 Field Documentation

3.8.2.30.0.16.1 uint32_t usb_device_ccid_data_rate_and_clock_frequency_response_t::dwClock-Frequency

This is an integer value

3.8.2.30.0.16.2 uint32_t usb_device_ccid_data_rate_and_clock_frequency_response_t::dwData-Rate

This is an integer value

3.8.2.31 struct _usb_device_ccid_notify_slot_chnage_notification

Data Fields

- uint8_t **bMessageType**
The message type.
- uint8_t **bmSlotICCState** [1]
This field is reported on byte granularity.

3.8.2.31.0.17 Field Documentation**3.8.2.31.0.17.1** `uint8_t usb_device_ccid_notify_slot_chnage_notification_t::bmSlotICCState[1]`**3.8.2.32 struct _usb_device_ccid_hardware_error_notification****Data Fields**

- `uint8_t bMessageType`
The message type.
- `uint8_t bSlot`
Identifies the slot number for this command.
- `uint8_t bSeq`
Sequence number of bulk out command when the hardware error occurred.
- `uint8_t bHardwareErrorCode`
0x01U - Over current.

3.8.2.32.0.18 Field Documentation**3.8.2.32.0.18.1** `uint8_t usb_device_ccid_hardware_error_notification_t::bHardwareErrorCode`**3.8.2.33 struct usb_device_ccid_transfer_struct_t****Data Fields**

- `struct`
`_usb_device_ccid_transfer_struct * next`
Next transfer pointer.
- `uint8_t * buffer`
The transfer buffer address need to be sent.
- `uint32_t length`
The transfer length.
- `usb_device_ccid_slot_status_response_t response`
Response buffer is used when dwLength = 0.

3.8.2.33.0.19 Field Documentation**3.8.2.33.0.19.1** `usb_device_ccid_slot_status_response_t usb_device_ccid_transfer_struct_t::response`**3.8.2.34 struct usb_device_ccid_control_request_struct_t****Data Fields**

- `uint8_t * buffer`
The buffer address.
- `uint32_t length`
The data length.

3.8.2.35 struct usb_device_ccid_notification_struct_t

Data Fields

- uint8_t * [buffer](#)
The transferred buffer address.
- uint32_t [length](#)
The transferred data length.

3.8.2.36 struct usb_device_ccid_command_struct_t

Data Fields

- uint8_t * [commandBuffer](#)
The buffer address kept the command from host.
- uint32_t [commandLength](#)
The command length from host.
- uint8_t * [responseBuffer](#)
The response data need to be sent to host.
- uint32_t [responseLength](#)
The response data length.

3.8.2.37 struct usb_device_ccid_slot_status_struct_t

Data Fields

- uint8_t [slot](#)
The slot number need to get.
- uint8_t [present](#)
Is present or not.
- uint8_t [clockStatus](#)
The clock status.

3.8.2.38 struct usb_device_ccid_struct_t

Data Fields

- [usb_device_handle](#) [handle](#)
The device handle.
- [usb_device_class_config_struct_t](#) * [configStruct](#)
The configuration of the class.
- [usb_device_interface_struct_t](#) * [interfaceHandle](#)
Current interface handle.
- [usb_device_ccid_transfer_struct_t](#) * [transferHead](#)
Transfer queue for busy.
- [usb_device_ccid_transfer_struct_t](#) * [transferFree](#)
Transfer queue for idle.

- uint8_t **commandBuffer** [USB_DEVICE_CCID_BUFFER_4BYTE_ALIGN(USB_DEVICE_CONFIG_CCID_MAX_MESSAGE_LENGTH)]
Command buffer for getting command data from host.
- usb_device_ccid_transfer_struct_t **transfers** [USB_DEVICE_CONFIG_CCID_TRANSFER_COUNT]
Transfer entity.
- uint8_t **slotsChangeBuffer** [(USB_DEVICE_CONFIG_CCID_SLOT_MAX *2-1U)/8+1U+1U]
The buffer for saving slot status.
- uint8_t **slotsSendingChangeBuffer** [(USB_DEVICE_CONFIG_CCID_SLOT_MAX *2-1U)/8+1U+1U]
The buffer is used to notify host the slot status changed.
- uint8_t **slotsSequenceNumber** [USB_DEVICE_CONFIG_CCID_SLOT_MAX]
Save each slot sequence number.
- usb_device_ccid_hardware_error_notification_t **hardwareError**
The buffer is used to notify host the hardware error happened.
- uint8_t **configuration**
Current configuration.
- uint8_t **interfaceNumber**
The interface number of the class.
- uint8_t **alternate**
Current alternate setting of the interface.
- uint8_t **endpointInterruptIn**
The endpoint number of the interrupt IN pipe.
- uint8_t **endpointBulkIn**
The endpoint number of the bulk IN pipe.
- uint8_t **endpointBulkOut**
The endpoint number of the bulk OUT pipe.
- uint8_t **slots**
The slot number of the application.
- uint8_t **bulkInBusy**
The bulk IN pipe is busy or not.
- uint8_t **interruptInBusy**
The interrupt IN pipe is busy or not.
- uint8_t **slotsChanged**
The slot status changed.

USB CCID Class driver

3.8.2.38.0.20 Field Documentation

3.8.2.38.0.20.1 `usb_device_class_config_struct_t* usb_device_ccid_struct_t::configStruct`

3.8.2.38.0.20.2 `uint8_t usb_device_ccid_struct_t::bulkInBusy`

3.8.2.38.0.20.3 `uint8_t usb_device_ccid_struct_t::interruptInBusy`

3.8.3 Macro Definition Documentation

3.8.3.1 `#define USB_DEVICE_CONFIG_CCID_SLOT_MAX (1U)`

3.8.3.2 `#define USB_DEVICE_CONFIG_CCID_TRANSFER_COUNT (4U)`

3.8.3.3 `#define USB_DEVICE_CONFIG_CCID_MAX_MESSAGE_LENGTH (271U)`

3.8.3.4 `#define USB_DEVICE_CCID_COMMAND_HEADER_LENGTH (0x0AU)`

3.8.3.5 `#define USB_DEVICE_CCID_RESPONSE_HEADER_LENGTH (0x0AU)`

3.8.4 Enumeration Type Documentation

3.8.4.1 `enum usb_device_ccid_event_t`

Enumerator

kUSB_DeviceCcidEventCommandReceived Command received in BULK OUT pipe.
kUSB_DeviceCcidEventResponseSent Response sent in BULK IN pipe.
kUSB_DeviceCcidEventGetSlotCount Get the slot count.
kUSB_DeviceCcidEventGetSlotStatus Get the slot status, including clock status, ICC present.
kUSB_DeviceCcidEventCommandAbort Command abort request received from control pipe.
kUSB_DeviceCcidEventGetClockFrequencies Get the clock frequencies.
kUSB_DeviceCcidEventGetDataRate Get the data rate.
kUSB_DeviceCcidEventSlotChangeSent Slot changed notification send completed.
kUSB_DeviceCcidEventHardwareErrorSent Hardware error notification send completed.

3.8.4.2 `enum usb_device_ccid_slot_state_t`

Enumerator

kUSB_DeviceCcidSlotStateNoPresent Not present.
kUSB_DeviceCcidSlotStatePresent Present.

3.8.4.3 enum usb_device_ccid_hardware_error_t

Enumerator

kUSB_DeviceCcidHardwareErrorOverCurrent Over current.

3.8.5 Function Documentation

3.8.5.1 usb_status_t USB_DeviceCcidInit (uint8_t *controllerId*, usb_device_class_config_struct_t * *config*, class_handle_t * *handle*)

This function is used to initialize the CCID class. This function only can be called by [USB_DeviceClass-Init](#).

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>config</i>	The class configuration information.
out	<i>handle</i>	An out parameter used to return pointer of the video class handle to the caller.

Returns

A USB error code or kStatus_USB_Success.

3.8.5.2 usb_status_t USB_DeviceCcidDeinit (class_handle_t *handle*)

The function deinitializes the device CCID class. This function can only be called by [USB_DeviceClass-Deinit](#).

Parameters

in	<i>handle</i>	The CCID class handle received from usb_device_class_config_struct_t::classHandle .
----	---------------	-----------------------------------------------------------------------------------------------------

Returns

A USB error code or kStatus_USB_Success.

USB CCID Class driver

3.8.5.3 `usb_status_t USB_DeviceCcidEvent (void * handle, uint32_t event, void * param)`

This function handles the event passed to the CCID class. This function can only be called by [USB_DeviceClassEvent](#).

Parameters

in	<i>handle</i>	The CCID class handle, received from the usb_device_class_config_struct_t::classHandle .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	Free device handle successfully.
<i>kStatus_USB_Invalid-Parameter</i>	The device handle not be found.
<i>kStatus_USB_Invalid-Request</i>	The request is invalid and the control pipe is stalled by the caller.

3.8.5.4 `usb_status_t USB_DeviceCcidNotifySlotChange (class_handle_t handle, uint8_t slot, usb_device_ccid_slot_state_t state)`

The function is used to notify that the slot status changed. This is a non-blocking function. The event `kUSB_DeviceCcidEventSlotChangeSent` is asserted when the transfer completed.

The slot status may not be sent to the host if the interrupt IN pipe is busy. The status is saved internally and sent to the host when the interrupt IN pipe callback called. So, the event `kUSB_DeviceCcidEventSlotChangeSent` happened times does not equal to the function call times of this function.

Parameters

in	<i>handle</i>	The CCID class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>slot</i>	The changed slot number.
in	<i>state</i>	The changed slot status.

Returns

A USB error code or `kStatus_USB_Success`.

USB CCID Class driver

3.8.5.5 `usb_status_t USB_DeviceCcidNotifyHardwareError (class_handle_t handle, uint8_t slot, usb_device_ccid_hardware_error_t errorCode)`

The function is used to notify the hardware error. This is a non-blocking function. The event `kUSB_DeviceCcidEventHardwareErrorSent` is asserted when the transfer completed.

If the interrupt IN pipe is busy, the function returns an error `kStatus_USB_Error`.

Parameters

in	<i>handle</i>	The CCID class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>slot</i>	The changed slot number.
in	<i>errorCode</i>	The hardware error code.

Returns

A USB error code or `kStatus_USB_Success`.

3.9 USB HID Class driver

3.9.1 Overview

Data Structures

- struct `usb_device_hid_report_struct_t`
The device HID GET/SET report structure. [More...](#)
- struct `usb_device_hid_struct_t`
The HID device class status structure. [More...](#)

Macros

- #define `USB_DEVICE_CONFIG_HID_CLASS_CODE` (0x03U)
The class code of the HID class.
- #define `USB_DEVICE_HID_REQUEST_GET_REPORT` (0x01U)
Request code to get report of HID class.
- #define `USB_DEVICE_HID_REQUEST_GET_IDLE` (0x02U)
Request code to get idle of HID class.
- #define `USB_DEVICE_HID_REQUEST_GET_PROTOCOL` (0x03U)
Request code to get protocol of HID class.
- #define `USB_DEVICE_HID_REQUEST_SET_REPORT` (0x09U)
Request code to set report of HID class.
- #define `USB_DEVICE_HID_REQUEST_SET_IDLE` (0x0AU)
Request code to set idle of HID class.
- #define `USB_DEVICE_HID_REQUEST_SET_PROTOCOL` (0x0BU)
Request code to set protocol of HID class.

Enumerations

- enum `usb_device_hid_event_t` {
`kUSB_DeviceHidEventSendResponse` = 0x01U,
`kUSB_DeviceHidEventRecvResponse`,
`kUSB_DeviceHidEventGetReport`,
`kUSB_DeviceHidEventGetIdle`,
`kUSB_DeviceHidEventGetProtocol`,
`kUSB_DeviceHidEventSetReport`,
`kUSB_DeviceHidEventSetIdle`,
`kUSB_DeviceHidEventSetProtocol`,
`kUSB_DeviceHidEventRequestReportBuffer` }
 Available common EVENT types in HID class callback.

USB HID Class driver

Functions

- `usb_status_t USB_DeviceHidInit` (`uint8_t` controllerId, `usb_device_class_config_struct_t` *config, `class_handle_t` *handle)
Initializes the HID class.
- `usb_status_t USB_DeviceHidDeinit` (`class_handle_t` handle)
Deinitializes the device HID class.
- `usb_status_t USB_DeviceHidEvent` (`void` *handle, `uint32_t` event, `void` *param)
Handles the event passed to the HID class.

USB device HID class APIs

- `usb_status_t USB_DeviceHidSend` (`class_handle_t` handle, `uint8_t` ep, `uint8_t` *buffer, `uint32_t` length)
Sends data through a specified endpoint.
- `usb_status_t USB_DeviceHidRecv` (`class_handle_t` handle, `uint8_t` ep, `uint8_t` *buffer, `uint32_t` length)
Receives data through a specified endpoint.

3.9.2 Data Structure Documentation

3.9.2.1 struct usb_device_hid_report_struct_t

This structure is used to pass data when the event type is `kUSB_DeviceHidEventGetReport`, `kUSB_DeviceHidEventSetReport`, and `kUSB_DeviceHidEventRequestReportBuffer`.

1. `kUSB_DeviceHidEventGetReport` The structure is used to save the report buffer and report length got from the application. The `reportBuffer` is the report data buffer address filled by the application. The `reportLength` is the report length. The `reportType` is the requested report type. The `reportId` is the requested report ID.
2. `kUSB_DeviceHidEventSetReport` The structure is used to pass the report data received from the host to the application. The `reportBuffer` is buffer address of the report data received from the host. The `reportLength` is the report data length. The `reportType` is the requested report type. The `reportId` is the requested report ID.
3. `kUSB_DeviceHidEventRequestReportBuffer` The structure is used to get the buffer to save the report data sent by the host. The `reportBuffer` is buffer address to receive to report data. It is filled by the application. The `reportLength` is the requested report data buffer length. The `reportType` is the requested report type. The `reportId` is the requested report ID.

Data Fields

- `uint8_t` * `reportBuffer`
The report buffer address.
- `uint32_t` `reportLength`
The report data length.

- `uint8_t reportType`
The report type.
- `uint8_t reportId`
The report ID.

3.9.2.2 struct usb_device_hid_struct_t

Data Fields

- `usb_device_handle handle`
The device handle.
- `usb_device_class_config_struct_t * configStruct`
The configuration of the class.
- `usb_device_interface_struct_t * interfaceHandle`
Current interface handle.
- `uint8_t * interruptInPipeDataBuffer`
IN pipe data buffer backup when stall.
- `uint32_t interruptInPipeDataLen`
IN pipe data length backup when stall.
- `uint8_t * interruptOutPipeDataBuffer`
OUT pipe data buffer backup when stall.
- `uint32_t interruptOutPipeDataLen`
OUT pipe data length backup when stall.
- `uint8_t configuration`
Current configuration.
- `uint8_t interfaceNumber`
The interface number of the class.
- `uint8_t alternate`
Current alternate setting of the interface.
- `uint8_t idleRate`
The idle rate of the HID device.
- `uint8_t protocol`
Current protocol.
- `uint8_t interruptInPipeBusy`
Interrupt IN pipe busy flag.
- `uint8_t interruptOutPipeBusy`
Interrupt OUT pipe busy flag.
- `uint8_t interruptInPipeStall`
Interrupt IN pipe stall flag.
- `uint8_t interruptOutPipeStall`
Interrupt OUT pipe stall flag.

USB HID Class driver

3.9.2.2.0.21 Field Documentation

3.9.2.2.0.21.1 `usb_device_class_config_struct_t* usb_device_hid_struct_t::configStruct`

3.9.3 Macro Definition Documentation

3.9.3.1 `#define USB_DEVICE_HID_REQUEST_GET_REPORT (0x01U)`

3.9.3.2 `#define USB_DEVICE_HID_REQUEST_GET_IDLE (0x02U)`

3.9.3.3 `#define USB_DEVICE_HID_REQUEST_GET_PROTOCOL (0x03U)`

3.9.3.4 `#define USB_DEVICE_HID_REQUEST_SET_REPORT (0x09U)`

3.9.3.5 `#define USB_DEVICE_HID_REQUEST_SET_IDLE (0x0AU)`

3.9.3.6 `#define USB_DEVICE_HID_REQUEST_SET_PROTOCOL (0x0BU)`

3.9.4 Enumeration Type Documentation

3.9.4.1 `enum usb_device_hid_event_t`

Enumerator

kUSB_DeviceHidEventSendResponse Send data completed.

kUSB_DeviceHidEventRecvResponse Data received.

kUSB_DeviceHidEventGetReport Get report request.

kUSB_DeviceHidEventGetIdle Get idle request.

kUSB_DeviceHidEventGetProtocol Get protocol request.

kUSB_DeviceHidEventSetReport Set report request.

kUSB_DeviceHidEventSetIdle Set idle request.

kUSB_DeviceHidEventSetProtocol Set protocol request.

kUSB_DeviceHidEventRequestReportBuffer Get buffer to save the data of the set report request.

3.9.5 Function Documentation

3.9.5.1 `usb_status_t USB_DeviceHidInit (uint8_t controllerId, usb_device_class_config_struct_t * config, class_handle_t * handle)`

This function is used to initialize the HID class. This function only can be called by [USB_DeviceClassInit](#).

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>config</i>	The class configuration information.
out	<i>handle</i>	An parameter used to return pointer of the HID class handle to the caller.

Returns

A USB error code or kStatus_USB_Success.

3.9.5.2 usb_status_t USB_DeviceHidDeinit (class_handle_t *handle*)

The function deinitializes the device HID class. This function only can be called by [USB_DeviceClassDeinit](#).

Parameters

in	<i>handle</i>	The HID class handle got from usb_device_class_config_struct_t::classHandle .
----	---------------	-----------------------------------------------------------------------------------------------

Returns

A USB error code or kStatus_USB_Success.

3.9.5.3 usb_status_t USB_DeviceHidEvent (void * *handle*, uint32_t *event*, void * *param*)

This function handles the event passed to the HID class. This function only can be called by [USB_DeviceClassEvent](#).

Parameters

in	<i>handle</i>	The HID class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or kStatus_USB_Success.

USB HID Class driver

Return values

<i>kStatus_USB_Success</i>	Free device handle successfully.
<i>kStatus_USB_Invalid-Parameter</i>	The device handle not be found.
<i>kStatus_USB_Invalid-Request</i>	The request is invalid, and the control pipe is stalled by the caller.

3.9.5.4 **usb_status_t USB_DeviceHidSend (class_handle_t *handle*, uint8_t *ep*, uint8_t * *buffer*, uint32_t *length*)**

The function is used to send data through a specified endpoint. The function calls [USB_DeviceSend-Request](#) internally.

Parameters

in	<i>handle</i>	The HID class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length to be sent.

Returns

A USB error code or *kStatus_USB_Success*.

Note

The return value indicates whether the sending request is successful or not. The transfer done is notified by *usb_device_hid_interrupt_in*. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the endpoint callback).

3.9.5.5 **usb_status_t USB_DeviceHidRecv (class_handle_t *handle*, uint8_t *ep*, uint8_t * *buffer*, uint32_t *length*)**

The function is used to receive data through a specified endpoint. The function calls [USB_DeviceRecv-Request](#) internally.

Parameters

in	<i>handle</i>	The HID class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length to be received.

Returns

A USB error code or `kStatus_USB_Success`.

Note

The return value indicates whether the receiving request is successful or not. The transfer done is notified by `usb_device_hid_interrupt_out`. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the endpoint callback).

3.10 USB PHDC Class driver

3.10.1 Overview

Data Structures

- struct `usb_device_phdc_pipe_t`
Definition of pipe structure. [More...](#)
- struct `usb_device_phdc_struct_t`
The PHDC device class status structure. [More...](#)

Macros

- #define `USB_DEVICE_CONFIG_PHDC_CLASS_CODE` (0x0F)
The class code of the PHDC class.
- #define `USB_DEVICE_PHDC_REQUEST_SET_FEATURE` (0x03)
The PHDC class set Meta-data message preamble feature request.
- #define `USB_DEVICE_PHDC_REQUEST_CLEAR_FEATURE` (0x01)
The PHDC class clear Meta-data message preamble feature request.
- #define `USB_DEVICE_PHDC_REQUEST_GET_STATUS` (0x00)
The PHDC class get data status request.

Enumerations

- enum `usb_device_phdc_event_t` {
 `kUSB_DevicePhdcEventInterruptInSendComplete` = 0x01,
 `kUSB_DevicePhdcEventBulkInSendComplete`,
 `kUSB_DevicePhdcEventDataReceived`,
 `kUSB_DevicePhdcEventSetFeature`,
 `kUSB_DevicePhdcEventClearFeature`,
 `kUSB_DevicePhdcEventGetStatus` }
Available common EVENT types in PHDC class callback.

Functions

- `usb_status_t USB_DevicePhdcInit` (uint8_t controllerId, `usb_device_class_config_struct_t` *config, `class_handle_t` *handle)
Initializes the PHDC class.
- `usb_status_t USB_DevicePhdcDeinit` (`class_handle_t` handle)
Deinitializes the device PHDC class.
- `usb_status_t USB_DevicePhdcEvent` (void *handle, uint32_t event, void *param)
Handles the event passed to the PHDC class.

USB device PHDC class APIs

- `usb_status_t USB_DevicePhdcSend` (`class_handle_t` handle, `uint8_t` ep, `uint8_t *buffer`, `uint32_t` length)
Sends data through a specified endpoint.
- `usb_status_t USB_DevicePhdcRecv` (`class_handle_t` handle, `uint8_t` ep, `uint8_t *buffer`, `uint32_t` length)
Receives data through a specified endpoint.

3.10.2 Data Structure Documentation

3.10.2.1 struct usb_device_phdc_pipe_t

Data Fields

- `uint8_t * pipeDataBuffer`
pipe data buffer backup when stall
- `uint32_t pipeDataLen`
pipe data length backup when stall
- `uint8_t pipeStall`
pipe is stall
- `uint8_t ep`
The endpoint number of the pipe.
- `uint8_t isBusy`
1: The pipe is transferring packet, 0: The pipe is idle.

3.10.2.1.0.22 Field Documentation

3.10.2.1.0.22.1 uint8_t usb_device_phdc_pipe_t::ep

3.10.2.1.0.22.2 uint8_t usb_device_phdc_pipe_t::isBusy

3.10.2.2 struct usb_device_phdc_struct_t

Data Fields

- `usb_device_handle` handle
The device handle.
- `usb_device_class_config_struct_t * configStruct`
The configuration of the class.
- `usb_device_interface_struct_t * interfaceHandle`
Current interface handle.
- `usb_device_phdc_pipe_t bulkIn`
The bulk in pipe for sending data.
- `usb_device_phdc_pipe_t bulkOut`
The bulk out pipe for receiving data.
- `usb_device_phdc_pipe_t interruptIn`
The interrupt in pipe for sending data.
- `uint8_t configuration`

USB PHDC Class driver

- *Current configuration.*
uint8_t **interfaceNumber**
The interface number of the class.
- uint8_t **alternate**
Current alternate setting of the interface.

3.10.2.2.0.23 Field Documentation

3.10.2.2.0.23.1 usb_device_class_config_struct_t* usb_device_phdc_struct_t::configStruct

3.10.3 Enumeration Type Documentation

3.10.3.1 enum usb_device_phdc_event_t

Enumerator

kUSB_DevicePhdcEventInterruptInSendComplete Send data completed.
kUSB_DevicePhdcEventBulkInSendComplete Send data completed.
kUSB_DevicePhdcEventDataReceived Data received.
kUSB_DevicePhdcEventSetFeature Set feature request.
kUSB_DevicePhdcEventClearFeature Clear feature request.
kUSB_DevicePhdcEventGetStatus Get status request.

3.10.4 Function Documentation

3.10.4.1 usb_status_t USB_DevicePhdcInit (uint8_t *controllerId*, usb_device_class_config_struct_t * *config*, class_handle_t * *handle*)

This function is used to initialize the PHDC class.

Parameters

<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t.
<i>config</i>	The class configuration information.
<i>handle</i>	An output parameter used to return pointer of the PHDC class handle to the caller.

Return values

<i>kStatus_USB_Success</i>	The PHDC class is initialized successfully.
----------------------------	---------------------------------------------

<i>kStatus_USB_Busy</i>	No PHDC device handle available for allocation.
<i>kStatus_USB_Invalid_Handle</i>	The PHDC device handle allocation failure.
<i>kStatus_USB_Invalid_Parameter</i>	The USB device handle allocation failure.

3.10.4.2 **usb_status_t USB_DevicePhdcDeinit (class_handle_t *handle*)**

The function deinitializes the device PHDC class.

Parameters

<i>handle</i>	The PHDC class handle received from usb_device_class_config_struct_t::classHandle .
---------------	-----------------------------------------------------------------------------------------------------

Return values

<i>kStatus_USB_Invalid_Handle</i>	The device handle is not found.
<i>kStatus_USB_Success</i>	The PHDC class is de-initialized successful.

3.10.4.3 **usb_status_t USB_DevicePhdcEvent (void * *handle*, uint32_t *event*, void * *param*)**

This function handles the event passed to the PHDC class.

Parameters

in	<i>handle</i>	The PHDC class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Return values

<i>kStatus_USB_Success</i>	Free device handle successfully.
----------------------------	----------------------------------

USB PHDC Class driver

<i>kStatus_USB_Invalid-Parameter</i>	The device handle is not found.
<i>kStatus_USB_Invalid-Request</i>	The request is invalid and the control pipe is stalled by the caller.

3.10.4.4 `usb_status_t USB_DevicePhdcSend (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to send data through a specified endpoint. The function calls [USB_DeviceSend-Request](#) internally.

Parameters

in	<i>handle</i>	The PHDC class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data to be sent.
in	<i>length</i>	The data length to be sent.

Return values

<i>kStatus_USB_Invalid-Handle</i>	The device handle is not found.
<i>kStatus_USB_Busy</i>	The previous transfer is pending.
<i>kStatus_USB_Success</i>	The sending is successful.

3.10.4.5 `usb_status_t USB_DevicePhdcRecv (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to receive data through a specified endpoint. The function calls the [USB_Device-RecvRequest](#) internally.

Parameters

in	<i>handle</i>	The PHDC class handle received from usb_device_class_config_struct_t::classHandle .
----	---------------	-----------------------------------------------------------------------------------------------------

in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length want to be received.

Return values

<i>kStatus_USB_Invalid-Handle</i>	The device handle is not found.
<i>kStatus_USB_Busy</i>	The previous transfer is pending.
<i>kStatus_USB_Success</i>	The receiving is successful.

3.11 USB PRINTER Class driver

3.11.1 Overview

Data Structures

- struct `usb_device_printer_struct_t`
The printer device class instance structure. [More...](#)

Macros

- #define `USB_DEVICE_CONFIG_PRINTER_CLASS_CODE` (0x07)
The class code of the printer class.
- #define `USB_DEVICE_PRINTER_GET_DEVICE_ID` (0x00U)
class-specific request GET_DEVICE_ID
- #define `USB_DEVICE_PRINTER_GET_PORT_STATUS` (0x01U)
class-specific request GET_PORT_STATUS
- #define `USB_DEVICE_PRINTER_SOFT_RESET` (0x02U)
class-specific request SOFT_RESET
- #define `USB_DEVICE_PRINTER_PORT_STATUS_PAPER_EMPTY_MASK` (0x20U)
Paper empty bit mask for GET_PORT_STATUS.
- #define `USB_DEVICE_PRINTER_PORT_STATUS_SELECT_MASK` (0x10U)
Select bit mask for GET_PORT_STATUS.
- #define `USB_DEVICE_PRINTER_PORT_STATUS_NOT_ERROR_MASK` (0x08U)
Error bit mask for GET_PORT_STATUS.

Enumerations

- enum `usb_device_printer_event_t` {
 `kUSB_DevicePrinterEventRecvResponse` = 0x01U,
 `kUSB_DevicePrinterEventSendResponse`,
 `kUSB_DevicePrinterEventGetDeviceId`,
 `kUSB_DevicePrinterEventGetPortStatus`,
 `kUSB_DevicePrinterEventSoftReset` }
Available common EVENT types in printer class callback.

Functions

- `usb_status_t USB_DevicePrinterInit` (uint8_t controllerId, `usb_device_class_config_struct_t` *config, `class_handle_t` *handle)
Initializes the printer class.
- `usb_status_t USB_DevicePrinterDeinit` (`class_handle_t` handle)
De-initializes the device printer class.
- `usb_status_t USB_DevicePrinterEvent` (void *handle, uint32_t event, void *param)
Handles the event passed to the printer class.

USB device printer class APIs

- `usb_status_t USB_DevicePrinterSend (class_handle_t handle, uint8_t ep, uint8_t *buffer, uint32_t length)`
Sends data through a specified endpoint.
- `usb_status_t USB_DevicePrinterRecv (class_handle_t handle, uint8_t ep, uint8_t *buffer, uint32_t length)`
Receives data through a specified endpoint.

3.11.2 Data Structure Documentation

3.11.2.1 struct usb_device_printer_struct_t

Data Fields

- `usb_device_handle deviceHandle`
The device handle.
- `usb_device_class_config_struct_t * classConfig`
The configuration of the class.
- `usb_device_interface_struct_t * interfaceHandle`
Current interface handle.
- `uint8_t * bulkInPipeDataBuffer`
IN pipe data buffer backup when stall.
- `uint32_t bulkInPipeDataLen`
IN pipe data length backup when stall.
- `uint8_t * bulkOutPipeDataBuffer`
OUT pipe data buffer backup when stall.
- `uint32_t bulkOutPipeDataLen`
OUT pipe data length backup when stall.
- `uint8_t configuration`
Current configuration.
- `uint8_t interfaceNumber`
Interface number in the device descriptor.
- `uint8_t alternate`
Interface alternate value.
- `uint8_t bulkInBusy`
BULK IN pipe busy flag.
- `uint8_t bulkOutBusy`
BULK OUT pipe busy flag.
- `uint8_t bulkInPipeStall`
bulk IN pipe stall flag
- `uint8_t bulkOutPipeStall`
bulk OUT pipe stall flag

USB PRINTER Class driver

3.11.2.1.0.24 Field Documentation

3.11.2.1.0.24.1 `usb_device_class_config_struct_t* usb_device_printer_struct_t::classConfig`

3.11.3 Enumeration Type Documentation

3.11.3.1 `enum usb_device_printer_event_t`

Enumerator

kUSB_DevicePrinterEventRecvResponse Data received.
kUSB_DevicePrinterEventSendResponse Data send done.
kUSB_DevicePrinterEventGetDeviceId Get device ID request.
kUSB_DevicePrinterEventGetPortStatus Get port status request.
kUSB_DevicePrinterEventSoftReset Soft reset request.

3.11.4 Function Documentation

3.11.4.1 `usb_status_t USB_DevicePrinterInit (uint8_t controllerId, usb_device_class_config_struct_t * config, class_handle_t * handle)`

This function is used to initialize the printer class. This function only can be called by [USB_DeviceClass-Init](#).

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>config</i>	The class configuration information.
out	<i>handle</i>	A parameter used to return a pointer of the printer class handle to the caller.

Returns

A USB error code or `kStatus_USB_Success`.

3.11.4.2 `usb_status_t USB_DevicePrinterDeinit (class_handle_t handle)`

The function de-initializes the device printer class. This function only can be called by [USB_DeviceClass-Deinit](#).

Parameters

in	<i>handle</i>	The printer class handle got from usb_device_class_config_struct_t::classHandle .
----	---------------	---------------------------------------------------------------------------------------------------

Returns

A USB error code or `kStatus_USB_Success`.

3.11.4.3 `usb_status_t USB_DevicePrinterEvent (void * handle, uint32_t event, void * param)`

This function handles the event passed to the printer class. This function only can be called by [USB_DeviceClassEvent](#).

Parameters

in	<i>handle</i>	The printer class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or `kStatus_USB_Success`.

Return values

<i>kStatus_USB_Success</i>	Process event successfully.
<i>kStatus_USB_Invalid_Handle</i>	The device handle or parameter is invalid.
<i>kStatus_USB_Invalid_Request</i>	The request is invalid, and the control pipe is stalled by the caller.

3.11.4.4 `usb_status_t USB_DevicePrinterSend (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to send data through a specified endpoint. The function calls [USB_DeviceSendRequest](#) internally.

USB PRINTER Class driver

Parameters

in	<i>handle</i>	The printer class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length to be sent.

Returns

A USB error code or `kStatus_USB_Success`.

Note

The return value indicates whether the sending request is successful or not. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the callback).

3.11.4.5 `usb_status_t USB_DevicePrinterRecv (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to receive data through a specified endpoint. The function calls [USB_DeviceSendRequest](#) internally.

Parameters

in	<i>handle</i>	The printer class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length to be sent.

Returns

A USB error code or `kStatus_USB_Success`.

Note

The return value indicates whether the sending request is successful or not. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the callback).

3.12 USB VIDEO Class driver

3.12.1 Overview

Data Structures

- struct [usb_device_video_mjpeg_payload_header_struct_t](#)
The payload header structure for MJPEG payload format. [More...](#)
- struct [usb_device_video_probe_and_commit_controls_struct_t](#)
The Video probe and commit controls structure. [More...](#)
- struct [usb_device_video_still_probe_and_commit_controls_struct_t](#)
The Video still probe and still commit controls structure. [More...](#)
- struct [usb_device_video_entity_struct_t](#)
The video device class-specific information. [More...](#)
- struct [usb_device_video_entities_struct_t](#)
The video device class-specific information list. [More...](#)
- struct [usb_device_video_struct_t](#)
The video device class status structure. [More...](#)

Macros

- #define [USB_DEVICE_VIDEO_STILL_IMAGE_TRIGGER_NORMAL_OPERATION](#) (0x00U)
Video device still image trigger control.

Enumerations

- enum [usb_device_video_event_t](#) {
 [kUSB_DeviceVideoEventStreamSendResponse](#) = 0x01U,
 [kUSB_DeviceVideoEventStreamRecvResponse](#),
 [kUSB_DeviceVideoEventControlSendResponse](#),
 [kUSB_DeviceVideoEventClassRequestBuffer](#) }
Available common event types in video class callback.

Functions

- [usb_status_t](#) [USB_DeviceVideoInit](#) (uint8_t controllerId, [usb_device_class_config_struct_t](#) *config, [class_handle_t](#) *handle)
Initializes the video class.
- [usb_status_t](#) [USB_DeviceVideoDeinit](#) ([class_handle_t](#) handle)
Deinitializes the device video class.
- [usb_status_t](#) [USB_DeviceVideoEvent](#) (void *handle, uint32_t event, void *param)
Handles the event passed to the video class.

USB Video class codes

- #define **USB_DEVICE_VIDEO_CC_VIDEO** (0x0EU)
Video device class code.
- #define **USB_DEVICE_VIDEO_SC_UNDEFINED** (0x00U)
Video device subclass code.
- #define **USB_DEVICE_VIDEO_SC_VIDEOCONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_SC_VIDIOSTREAMING** (0x02U)
- #define **USB_DEVICE_VIDEO_SC_VIDEO_INTERFACE_COLLECTION** (0x03U)
- #define **USB_DEVICE_VIDEO_PC_PROTOCOL_UNDEFINED** (0x00U)
Video device protocol code.
- #define **USB_DEVICE_VIDEO_PC_PROTOCOL_15** (0x01U)
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_UNDEFINED** (0x20U)
Video device class-specific descriptor type.
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_DEVICE** (0x21U)
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_CONFIGURATION** (0x22U)
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_STRING** (0x23U)
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_INTERFACE** (0x24U)
- #define **USB_DESCRIPTOR_TYPE_VIDEO_CS_ENDPOINT** (0x25U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_DESCRIPTOR_UNDEFINED** (0x00U)
Video device class-specific VC interface descriptor subtype.
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_HEADER** (0x01U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_INPUT_TERMINAL** (0x02U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_OUTPUT_TERMINAL** (0x03U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_SELECTOR_UNIT** (0x04U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_PROCESSING_UNIT** (0x05U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_EXTENSION_UNIT** (0x06U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VC_ENCODING_UNIT** (0x07U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_UNDEFINED** (0x00U)
Video device class-specific VS interface descriptor subtype.
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_INPUT_HEADER** (0x01U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_OUTPUT_HEADER** (0x02U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_STILL_IMAGE_FRAME** (0x03U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_UNCOMPRESSED** (0x04-U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FRAME_UNCOMPRESSED** (0x05U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_MJPEG** (0x06U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FRAME_MJPEG** (0x07U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_MPEG2TS** (0x0AU)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_DV** (0x0CU)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_COLORFORMAT** (0x0DU)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_FRAME_BASED** (0x10U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FRAME_FRAME_BASED** (0x11U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_STREAM_BASED** (0x12-U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_H264** (0x13U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FRAME_H264** (0x14U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_H264_SIMULCAST** (0x15-U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_VP8** (0x16U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FRAME_VP8** (0x17U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_VS_FORMAT_VP8_SIMULCAST** (0x18-U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_EP_UNDEFINED** (0x00U)

USB VIDEO Class driver

Video device class-specific VC endpoint descriptor subtype.

- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_EP_GENERAL** (0x01U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_EP_ENDPOINT** (0x02U)
- #define **USB_DESCRIPTOR_SUBTYPE_VIDEO_EP_INTERRUPT** (0x03U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_UNDEFINED** (0x00U)

Video device class-specific request code.

- #define **USB_DEVICE_VIDEO_REQUEST_CODE_SET_CUR** (0x01U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_SET_CUR_ALL** (0x11U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_CUR** (0x81U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_MIN** (0x82U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_MAX** (0x83U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_RES** (0x84U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_LEN** (0x85U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_INFO** (0x86U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_DEF** (0x87U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_CUR_ALL** (0x91U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_MIN_ALL** (0x92U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_MAX_ALL** (0x93U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_RES_ALL** (0x94U)
- #define **USB_DEVICE_VIDEO_REQUEST_CODE_GET_DEF_ALL** (0x97U)
- #define **USB_DEVICE_VIDEO_VC_CONTROL_UNDEFINED** (0x00U)

Video device class-specific VideoControl interface control selector.

- #define **USB_DEVICE_VIDEO_VC_VIDEO_POWER_MODE_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_VC_REQUEST_ERROR_CODE_CONTROL** (0x02U)
- #define **USB_DEVICE_VIDEO_TE_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Terminal control selector.

- #define **USB_DEVICE_VIDEO_SU_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Selector Unit control selector.

- #define **USB_DEVICE_VIDEO_SU_INPUT_SELECT_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_CT_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Camera Terminal control selector.

- #define **USB_DEVICE_VIDEO_CT_SCANNING_MODE_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_CT_AE_MODE_CONTROL** (0x02U)
- #define **USB_DEVICE_VIDEO_CT_AE_PRIORITY_CONTROL** (0x03U)
- #define **USB_DEVICE_VIDEO_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x04U)
- #define **USB_DEVICE_VIDEO_CT_EXPOSURE_TIME_RELATIVE_CONTROL** (0x05U)
- #define **USB_DEVICE_VIDEO_CT_FOCUS_ABSOLUTE_CONTROL** (0x06U)
- #define **USB_DEVICE_VIDEO_CT_FOCUS_RELATIVE_CONTROL** (0x07U)
- #define **USB_DEVICE_VIDEO_CT_FOCUS_AUTO_CONTROL** (0x08U)
- #define **USB_DEVICE_VIDEO_CT_IRIS_ABSOLUTE_CONTROL** (0x09U)
- #define **USB_DEVICE_VIDEO_CT_IRIS_RELATIVE_CONTROL** (0x0AU)
- #define **USB_DEVICE_VIDEO_CT_ZOOM_ABSOLUTE_CONTROL** (0x0BU)
- #define **USB_DEVICE_VIDEO_CT_ZOOM_RELATIVE_CONTROL** (0x0CU)
- #define **USB_DEVICE_VIDEO_CT_PANTILT_ABSOLUTE_CONTROL** (0x0DU)
- #define **USB_DEVICE_VIDEO_CT_PANTILT_RELATIVE_CONTROL** (0x0EU)
- #define **USB_DEVICE_VIDEO_CT_ROLL_ABSOLUTE_CONTROL** (0x0FU)
- #define **USB_DEVICE_VIDEO_CT_ROLL_RELATIVE_CONTROL** (0x10U)
- #define **USB_DEVICE_VIDEO_CT_PRIVACY_CONTROL** (0x11U)
- #define **USB_DEVICE_VIDEO_CT_FOCUS_SIMPLE_CONTROL** (0x12U)
- #define **USB_DEVICE_VIDEO_CT_WINDOW_CONTROL** (0x13U)
- #define **USB_DEVICE_VIDEO_CT_REGION_OF_INTEREST_CONTROL** (0x14U)
- #define **USB_DEVICE_VIDEO_PU_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Processing Unit control selector.

- #define **USB_DEVICE_VIDEO_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_PU_BRIGHTNESS_CONTROL** (0x02U)

- #define **USB_DEVICE_VIDEO_PU_CONTRAST_CONTROL** (0x03U)
- #define **USB_DEVICE_VIDEO_PU_GAIN_CONTROL** (0x04U)
- #define **USB_DEVICE_VIDEO_PU_POWER_LINE_FREQUENCY_CONTROL** (0x05U)
- #define **USB_DEVICE_VIDEO_PU_HUE_CONTROL** (0x06U)
- #define **USB_DEVICE_VIDEO_PU_SATURATION_CONTROL** (0x07U)
- #define **USB_DEVICE_VIDEO_PU_SHARPNESS_CONTROL** (0x08U)
- #define **USB_DEVICE_VIDEO_PU_GAMMA_CONTROL** (0x09U)
- #define **USB_DEVICE_VIDEO_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x0AU)
- #define **USB_DEVICE_VIDEO_PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL** (0x0BU)
- #define **USB_DEVICE_VIDEO_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x0CU)
- #define **USB_DEVICE_VIDEO_PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL** (0x0DU)
- #define **USB_DEVICE_VIDEO_PU_DIGITAL_MULTIPLIER_CONTROL** (0x0EU)
- #define **USB_DEVICE_VIDEO_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x0FU)
- #define **USB_DEVICE_VIDEO_PU_HUE_AUTO_CONTROL** (0x10U)
- #define **USB_DEVICE_VIDEO_PU_ANALOG_VIDEO_STANDARD_CONTROL** (0x11U)
- #define **USB_DEVICE_VIDEO_PU_ANALOG_LOCK_STATUS_CONTROL** (0x12U)
- #define **USB_DEVICE_VIDEO_PU_CONTRAST_AUTO_CONTROL** (0x13U)
- #define **USB_DEVICE_VIDEO_EU_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Encoding Unit control selector.

- #define **USB_DEVICE_VIDEO_EU_SELECT_LAYER_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_EU_PROFILE_TOOLSET_CONTROL** (0x02U)
- #define **USB_DEVICE_VIDEO_EU_VIDEO_RESOLUTION_CONTROL** (0x03U)
- #define **USB_DEVICE_VIDEO_EU_MIN_FRAME_INTERVAL_CONTROL** (0x04U)
- #define **USB_DEVICE_VIDEO_EU_SLICE_MODE_CONTROL** (0x05U)
- #define **USB_DEVICE_VIDEO_EU_RATE_CONTROL_MODE_CONTROL** (0x06U)
- #define **USB_DEVICE_VIDEO_EU_AVERAGE_BITRATE_CONTROL** (0x07U)
- #define **USB_DEVICE_VIDEO_EU_CPB_SIZE_CONTROL** (0x08U)
- #define **USB_DEVICE_VIDEO_EU_PEAK_BIT_RATE_CONTROL** (0x09U)
- #define **USB_DEVICE_VIDEO_EU_QUANTIZATION_PARAMS_CONTROL** (0x0AU)
- #define **USB_DEVICE_VIDEO_EU_SYNC_REF_FRAME_CONTROL** (0x0BU)
- #define **USB_DEVICE_VIDEO_EU_LTR_BUFFER_CONTROL** (0x0CU)
- #define **USB_DEVICE_VIDEO_EU_LTR_PICTURE_CONTROL** (0x0DU)
- #define **USB_DEVICE_VIDEO_EU_LTR_VALIDATION_CONTROL** (0x0EU)
- #define **USB_DEVICE_VIDEO_EU_LEVEL_IDC_LIMIT_CONTROL** (0x0FU)
- #define **USB_DEVICE_VIDEO_EU_SEI_PAYLOADTYPE_CONTROL** (0x10U)
- #define **USB_DEVICE_VIDEO_EU_QP_RANGE_CONTROL** (0x11U)
- #define **USB_DEVICE_VIDEO_EU_PRIORITY_CONTROL** (0x12U)
- #define **USB_DEVICE_VIDEO_EU_START_OR_STOP_LAYER_CONTROL** (0x13U)
- #define **USB_DEVICE_VIDEO_EU_ERROR_RESILIENCY_CONTROL** (0x14U)
- #define **USB_DEVICE_VIDEO_XU_CONTROL_UNDEFINED** (0x00U)

Video device class-specific Extension Unit control selector.

- #define **USB_DEVICE_VIDEO_VS_CONTROL_UNDEFINED** (0x00U)

Video device class-specific VideoStreaming Interface control selector.

- #define **USB_DEVICE_VIDEO_VS_PROBE_CONTROL** (0x01U)
- #define **USB_DEVICE_VIDEO_VS_COMMIT_CONTROL** (0x02U)
- #define **USB_DEVICE_VIDEO_VS_STILL_PROBE_CONTROL** (0x03U)
- #define **USB_DEVICE_VIDEO_VS_STILL_COMMIT_CONTROL** (0x04U)
- #define **USB_DEVICE_VIDEO_VS_STILL_IMAGE_TRIGGER_CONTROL** (0x05U)
- #define **USB_DEVICE_VIDEO_VS_STREAM_ERROR_CODE_CONTROL** (0x06U)
- #define **USB_DEVICE_VIDEO_VS_GENERATE_KEY_FRAME_CONTROL** (0x07U)

USB VIDEO Class driver

- #define **USB_DEVICE_VIDEO_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x08U)
- #define **USB_DEVICE_VIDEO_VS_SYNCH_DELAY_CONTROL** (0x09U)

USB Video class terminal types

- #define **USB_DEVICE_VIDEO_TT_VENDOR_SPECIFIC** (0x0100U)
Video device USB terminal type.
- #define **USB_DEVICE_VIDEO_TT_STREAMING** (0x0101U)
- #define **USB_DEVICE_VIDEO_ITT_VENDOR_SPECIFIC** (0x0200U)
Video device input terminal type.
- #define **USB_DEVICE_VIDEO_ITT_CAMERA** (0x0201U)
- #define **USB_DEVICE_VIDEO_ITT_MEDIA_TRANSPORT_INPUT** (0x0202U)
- #define **USB_DEVICE_VIDEO_OTT_VENDOR_SPECIFIC** (0x0300U)
Video device output terminal type.
- #define **USB_DEVICE_VIDEO_OTT_DISPLAY** (0x0301U)
- #define **USB_DEVICE_VIDEO_OTT_MEDIA_TRANSPORT_OUTPUT** (0x0302U)
- #define **USB_DEVICE_VIDEO_ET_VENDOR_SPECIFIC** (0x0400U)
Video device external terminal type.
- #define **USB_DEVICE_VIDEO_ET_COMPOSITE_CONNECTOR** (0x0401U)
- #define **USB_DEVICE_VIDEO_ET_SVIDEO_CONNECTOR** (0x0402U)
- #define **USB_DEVICE_VIDEO_ET_COMPONENT_CONNECTOR** (0x0403U)

USB Video class setup request types

- #define **USB_DEVICE_VIDEO_SET_REQUEST_INTERFACE** (0x21U)
Video device class setup request set type.
- #define **USB_DEVICE_VIDEO_SET_REQUEST_ENDPOINT** (0x22U)
- #define **USB_DEVICE_VIDEO_GET_REQUEST_INTERFACE** (0xA1U)
Video device class setup request get type.
- #define **USB_DEVICE_VIDEO_GET_REQUEST_ENDPOINT** (0xA2U)

USB Video device class-specific request commands

- #define **USB_DEVICE_VIDEO_GET_CUR_VC_POWER_MODE_CONTROL** (0x8101U)
Video device class-specific request GET CUR COMMAND.
- #define **USB_DEVICE_VIDEO_GET_CUR_VC_ERROR_CODE_CONTROL** (0x8102U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x8121U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_BRIGHTNESS_CONTROL** (0x8122U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_CONTRACT_CONTROL** (0x8123U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_GAIN_CONTROL** (0x8124U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_POWER_LINE_FREQUENCY_CONTROL** (0x8125U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_HUE_CONTROL** (0x8126U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_SATURATION_CONTROL** (0x8127U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_SHARRNESS_CONTROL** (0x8128U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_GAMMA_CONTROL** (0x8129U)
- #define **USB_DEVICE_VIDEO_GET_CUR_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x812AU)

- **#define USB_DEVICE_VIDEO_GET_CUR_PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL** (0x812BU)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x812CU)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL** (0x812DU)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_DIGITAL_MULTIPLIER_CONTROL** (0x812EU)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x812FU)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_HUE_AUTO_CONTROL** (0x8130U)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_ANALOG_VIDEO_STANDARD_CONTROL** (0x8131U)
- **#define USB_DEVICE_VIDEO_GET_CUR_PU_ANALOG_LOCK_STATUS_CONTROL** (0x8132U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_SCANNING_MODE_CONTROL** (0x8141U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_AE_MODE_CONTROL** (0x8142U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_AE_PRIORITY_CONTROL** (0x8143U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x8144U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_EXPOSURE_TIME_RELATIVE_CONTROL** (0x8145U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_FOCUS_ABSOLUTE_CONTROL** (0x8146U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_FOCUS_RELATIVE_CONTROL** (0x8147U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_FOCUS_AUTO_CONTROL** (0x8148U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_IRIS_ABSOLUTE_CONTROL** (0x8149U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_IRIS_RELATIVE_CONTROL** (0x814AU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_ZOOM_ABSOLUTE_CONTROL** (0x814BU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_ZOOM_RELATIVE_CONTROL** (0x814CU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_PANTILT_ABSOLUTE_CONTROL** (0x814DU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_PANTILT_RELATIVE_CONTROL** (0x814EU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_ROLL_ABSOLUTE_CONTROL** (0x814FU)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_ROLL_RELATIVE_CONTROL** (0x8150U)
- **#define USB_DEVICE_VIDEO_GET_CUR_CT_PRIVACY_CONTROL** (0x8151U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_PROBE_CONTROL** (0x8161U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_COMMIT_CONTROL** (0x8162U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_STILL_PROBE_CONTROL** (0x8163U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_STILL_COMMIT_CONTROL** (0x8164U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_STILL_IMAGE_TRIGGER_CONTROL** (0x8165U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_STREAM_ERROR_CODE_CONTROL** (0x8166U)
- **#define USB_DEVICE_VIDEO_GET_CUR_VS_GENERATE_KEY_FRAME_CONTROL**

USB VIDEO Class driver

- L** (0x8167U)
- #define **USB_DEVICE_VIDEO_GET_CUR_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x8168U)
- #define **USB_DEVICE_VIDEO_GET_CUR_VS_SYNCH_DELAY_CONTROL** (0x8169U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x8221U)
- Video device class-specific request GET MIN COMMAND.*
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_BRIGHTNESS_CONTROL** (0x8222U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_CONTRACT_CONTROL** (0x8223U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_GAIN_CONTROL** (0x8224U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_HUE_CONTROL** (0x8226U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_SATURATION_CONTROL** (0x8227U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_SHARRNESS_CONTROL** (0x8228U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_GAMMA_CONTROL** (0x8229U)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x822AU)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x822CU)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_DIGITAL_MULTIPLIER_CONTROL** (0x822EU)
- #define **USB_DEVICE_VIDEO_GET_MIN_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x822FU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x8244U)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_FOCUS_ABSOLUTE_CONTROL** (0x8246U)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_FOCUS_RELATIVE_CONTROL** (0x8247U)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_IRIS_ABSOLUTE_CONTROL** (0x8249U)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_ZOOM_ABSOLUTE_CONTROL** (0x824BU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_ZOOM_RELATIVE_CONTROL** (0x824CU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_PANTILT_ABSOLUTE_CONTROL** (0x824DU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_PANTILT_RELATIVE_CONTROL** (0x824EU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_ROLL_ABSOLUTE_CONTROL** (0x824FU)
- #define **USB_DEVICE_VIDEO_GET_MIN_CT_ROLL_RELATIVE_CONTROL** (0x8250U)
- #define **USB_DEVICE_VIDEO_GET_MIN_VS_PROBE_CONTROL** (0x8261U)
- #define **USB_DEVICE_VIDEO_GET_MIN_VS_STILL_PROBE_CONTROL** (0x8263U)
- #define **USB_DEVICE_VIDEO_GET_MIN_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x8268U)
- #define **USB_DEVICE_VIDEO_GET_MIN_VS_SYNCH_DELAY_CONTROL** (0x8269U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x8321U)
- Video device class-specific request GET MAX COMMAND.*
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_BRIGHTNESS_CONTROL** (0x8322U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_CONTRACT_CONTROL** (0x8323U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_GAIN_CONTROL** (0x8324U)

- #define **USB_DEVICE_VIDEO_GET_MAX_PU_HUE_CONTROL** (0x8326U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_SATURATION_CONTROL** (0x8327U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_SHARRNESS_CONTROL** (0x8328U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_GAMMA_CONTROL** (0x8329U)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x832AU)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x832CU)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_DIGITAL_MULTIPLIER_CONTROL** (0x832EU)
- #define **USB_DEVICE_VIDEO_GET_MAX_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x832FU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x8344U)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_FOCUS_ABSOLUTE_CONTROL** (0x8346U)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_FOCUS_RELATIVE_CONTROL** (0x8347U)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_IRIS_ABSOLUTE_CONTROL** (0x8349U)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_ZOOM_ABSOLUTE_CONTROL** (0x834BU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_ZOOM_RELATIVE_CONTROL** (0x834CU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_PANTILT_ABSOLUTE_CONTROL** (0x834DU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_PANTILT_RELATIVE_CONTROL** (0x834EU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_ROLL_ABSOLUTE_CONTROL** (0x834FU)
- #define **USB_DEVICE_VIDEO_GET_MAX_CT_ROLL_RELATIVE_CONTROL** (0x8350U)
- #define **USB_DEVICE_VIDEO_GET_MAX_VS_PROBE_CONTROL** (0x8361U)
- #define **USB_DEVICE_VIDEO_GET_MAX_VS_STILL_PROBE_CONTROL** (0x8363U)
- #define **USB_DEVICE_VIDEO_GET_MAX_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x8368U)
- #define **USB_DEVICE_VIDEO_GET_MAX_VS_SYNCH_DELAY_CONTROL** (0x8369U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x8421U)

Video device class-specific request GET RES COMMAND.

- #define **USB_DEVICE_VIDEO_GET_RES_PU_BRIGHTNESS_CONTROL** (0x8422U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_CONTRACT_CONTROL** (0x8423U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_GAIN_CONTROL** (0x8424U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_HUE_CONTROL** (0x8426U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_SATURATION_CONTROL** (0x8427U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_SHARRNESS_CONTROL** (0x8428U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_GAMMA_CONTROL** (0x8429U)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x842AU)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x842CU)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_DIGITAL_MULTIPLIER_CONTROL**

USB VIDEO Class driver

- L** (0x842EU)
- #define **USB_DEVICE_VIDEO_GET_RES_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x842FU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_AE_MODE_CONTROL** (0x8442U)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x8444U)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_FOCUS_ABSOLUTE_CONTROL** (0x8446U)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_FOCUS_RELATIVE_CONTROL** (0x8447U)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_IRIS_ABSOLUTE_CONTROL** (0x8449U)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_ZOOM_ABSOLUTE_CONTROL** (0x844BU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_ZOOM_RELATIVE_CONTROL** (0x844CU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_PANTILT_ABSOLUTE_CONTROL** (0x844DU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_PANTILT_RELATIVE_CONTROL** (0x844EU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_ROLL_ABSOLUTE_CONTROL** (0x844FU)
- #define **USB_DEVICE_VIDEO_GET_RES_CT_ROLL_RELATIVE_CONTROL** (0x8450U)
- #define **USB_DEVICE_VIDEO_GET_RES_VS_PROBE_CONTROL** (0x8461U)
- #define **USB_DEVICE_VIDEO_GET_RES_VS_STILL_PROBE_CONTROL** (0x8463U)
- #define **USB_DEVICE_VIDEO_GET_RES_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x8468U)
- #define **USB_DEVICE_VIDEO_GET_RES_VS_SYNCH_DELAY_CONTROL** (0x8469U)
- #define **USB_DEVICE_VIDEO_GET_LEN_VS_PROBE_CONTROL** (0x8561U)
- Video device class-specific request GET LEN COMMAND.*
- #define **USB_DEVICE_VIDEO_GET_LEN_VS_COMMIT_CONTROL** (0x8562U)
- #define **USB_DEVICE_VIDEO_GET_LEN_VS_STILL_PROBE_CONTROL** (0x8563U)
- #define **USB_DEVICE_VIDEO_GET_LEN_VS_STILL_COMMIT_CONTROL** (0x8564U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VC_POWER_MODE_CONTROL** (0x8601U)
- Video device class-specific request GET INFO COMMAND.*
- #define **USB_DEVICE_VIDEO_GET_INFO_VC_ERROR_CODE_CONTROL** (0x8602U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x8621U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_BRIGHTNESS_CONTROL** (0x8622U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_CONTRACT_CONTROL** (0x8623U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_GAIN_CONTROL** (0x8624U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_POWER_LINE_FREQUENCY_CONTROL** (0x8625U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_HUE_CONTROL** (0x8626U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_SATURATION_CONTROL** (0x8627U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_SHARRNESS_CONTROL** (0x8628U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_GAMMA_CONTROL** (0x8629U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x862AU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL** (0x862BU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x862CU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_WHITE_BALANCE_COMPONENT_AUTO**

- TO_CONTROL** (0x862DU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_DIGITAL_MULTIPLIER_CONTROL** (0x862EU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x862FU)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_HUE_AUTO_CONTROL** (0x8630U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_ANALOG_VIDEO_STANDARD_CONTROL** (0x8631U)
- #define **USB_DEVICE_VIDEO_GET_INFO_PU_ANALOG_LOCK_STATUS_CONTROL** (0x8632U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_SCANNING_MODE_CONTROL** (0x8641-U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_AE_MODE_CONTROL** (0x8642U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_AE_PRIORITY_CONTROL** (0x8643U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x8644U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_EXPOSURE_TIME_RELATIVE_CONTROL** (0x8645U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_FOCUS_ABSOLUTE_CONTROL** (0x8646-U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_FOCUS_RELATIVE_CONTROL** (0x8647-U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_FOCUS_AUTO_CONTROL** (0x8648U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_IRIS_ABSOLUTE_CONTROL** (0x8649U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_IRIS_RELATIVE_CONTROL** (0x864AU)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_ZOOM_ABSOLUTE_CONTROL** (0x864-BU)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_ZOOM_RELATIVE_CONTROL** (0x864-CU)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_PANTILT_ABSOLUTE_CONTROL** (0x864-DU)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_PANTILT_RELATIVE_CONTROL** (0x864-EU)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_ROLL_ABSOLUTE_CONTROL** (0x864F-U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_ROLL_RELATIVE_CONTROL** (0x8650-U)
- #define **USB_DEVICE_VIDEO_GET_INFO_CT_PRIVACY_CONTROL** (0x8651U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_PROBE_CONTROL** (0x8661U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_COMMIT_CONTROL** (0x8662U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_STILL_PROBE_CONTROL** (0x8663U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_STILL_COMMIT_CONTROL** (0x8664U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_STILL_IMAGE_TRIGGER_CONTROL** (0x8665U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_STREAM_ERROR_CODE_CONTROL** (0x8666U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_GENERATE_KEY_FRAME_CONTROL** (0x8667U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x8668U)
- #define **USB_DEVICE_VIDEO_GET_INFO_VS_SYNCH_DELAY_CONTROL** (0x8669U)

USB VIDEO Class driver

- `#define USB_DEVICE_VIDEO_GET_DEF_PU_BACKLIGHT_COMPENSATION_CONTROL (0x8721U)`
Video device class-specific request GET DEF COMMAND.
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_BRIGHTNESS_CONTROL (0x8722U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_CONTRACT_CONTROL (0x8723U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_GAIN_CONTROL (0x8724U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_POWER_LINE_FREQUENCY_CONTROL (0x8725U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_HUE_CONTROL (0x8726U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_SATURATION_CONTROL (0x8727U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_SHARRNESS_CONTROL (0x8728U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_GAMMA_CONTROL (0x8729U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_WHITE_BALANCE_TEMPERATURE_CONTROL (0x872AU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL (0x872BU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_WHITE_BALANCE_COMPONENT_CONTROL (0x872CU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL (0x872DU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_DIGITAL_MULTIPLIER_CONTROL (0x872EU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL (0x872FU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_PU_HUE_AUTO_CONTROL (0x8730U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_AE_MODE_CONTROL (0x8742U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL (0x8744U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_FOCUS_ABSOLUTE_CONTROL (0x8746U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_FOCUS_RELATIVE_CONTROL (0x8747U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_FOCUS_AUTO_CONTROL (0x8748U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_IRIS_ABSOLUTE_CONTROL (0x8749U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_ZOOM_ABSOLUTE_CONTROL (0x874BU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_ZOOM_RELATIVE_CONTROL (0x874CU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_PANTILT_ABSOLUTE_CONTROL (0x874DU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_PANTILT_RELATIVE_CONTROL (0x874EU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_ROLL_ABSOLUTE_CONTROL (0x874FU)`
- `#define USB_DEVICE_VIDEO_GET_DEF_CT_ROLL_RELATIVE_CONTROL (0x8750U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_VS_PROBE_CONTROL (0x8761U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_VS_STILL_PROBE_CONTROL (0x8763U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_VS_UPDATE_FRAME_SEGMENT_CONTROL (0x8768U)`
- `#define USB_DEVICE_VIDEO_GET_DEF_VS_SYNCH_DELAY_CONTROL (0x8769U)`
- `#define USB_DEVICE_VIDEO_SET_CUR_VC_POWER_MODE_CONTROL (0x0101U)`

Video device class-specific request SET_CUR COMMAND.

- #define **USB_DEVICE_VIDEO_SET_CUR_PU_BACKLIGHT_COMPENSATION_CONTROL** (0x0121U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_BRIGHTNESS_CONTROL** (0x0122U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_CONTRACT_CONTROL** (0x0123U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_GAIN_CONTROL** (0x0124U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_POWER_LINE_FREQUENCY_CONTROL** (0x0125U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_HUE_CONTROL** (0x0126U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_SATURATION_CONTROL** (0x0127U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_SHARRNESS_CONTROL** (0x0128U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_GAMMA_CONTROL** (0x0129U)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_WHITE_BALANCE_TEMPERATURE_CONTROL** (0x012AU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_WHITE_BALANCE_TEMPERATURE_AUTO_CONTROL** (0x012BU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_WHITE_BALANCE_COMPONENT_CONTROL** (0x012CU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_WHITE_BALANCE_COMPONENT_AUTO_CONTROL** (0x012DU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_DIGITAL_MULTIPLIER_CONTROL** (0x012EU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_DIGITAL_MULTIPLIER_LIMIT_CONTROL** (0x012FU)
- #define **USB_DEVICE_VIDEO_SET_CUR_PU_HUE_AUTO_CONTROL** (0x0130U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_SCANNING_MODE_CONTROL** (0x0141U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_AE_MODE_CONTROL** (0x0142U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_AE_PRIORITY_CONTROL** (0x0143U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_EXPOSURE_TIME_ABSOLUTE_CONTROL** (0x0144U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_EXPOSURE_TIME_RELATIVE_CONTROL** (0x0145U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_FOCUS_ABSOLUTE_CONTROL** (0x0146U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_FOCUS_RELATIVE_CONTROL** (0x0147U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_FOCUS_AUTO_CONTROL** (0x0148U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_IRIS_ABSOLUTE_CONTROL** (0x0149U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_IRIS_RELATIVE_CONTROL** (0x014AU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_ZOOM_ABSOLUTE_CONTROL** (0x014BU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_ZOOM_RELATIVE_CONTROL** (0x014CU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_PANTILT_ABSOLUTE_CONTROL** (0x014DU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_PANTILT_RELATIVE_CONTROL** (0x014EU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_ROLL_ABSOLUTE_CONTROL** (0x014FU)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_ROLL_RELATIVE_CONTROL** (0x0150U)
- #define **USB_DEVICE_VIDEO_SET_CUR_CT_PRIVACY_CONTROL** (0x0151U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_PROBE_CONTROL** (0x0161U)

USB VIDEO Class driver

- #define **USB_DEVICE_VIDEO_SET_CUR_VS_COMMIT_CONTROL** (0x0162U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_STILL_PROBE_CONTROL** (0x0163U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_STILL_COMMIT_CONTROL** (0x0164U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_STILL_IMAGE_TRIGGER_CONTROL** (0x0165U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_STREAM_ERROR_CODE_CONTROL** (0x0166U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_GENERATE_KEY_FRAME_CONTROL** (0x0167U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_UPDATE_FRAME_SEGMENT_CONTROL** (0x0168U)
- #define **USB_DEVICE_VIDEO_SET_CUR_VS_SYNCH_DELAY_CONTROL** (0x0169U)

USB device video class APIs

- **usb_status_t USB_DeviceVideoSend** (**class_handle_t** handle, **uint8_t** ep, **uint8_t** *buffer, **uint32_t** length)
Sends data through a specified endpoint.
- **usb_status_t USB_DeviceVideoRecv** (**class_handle_t** handle, **uint8_t** ep, **uint8_t** *buffer, **uint32_t** length)
Receives data through a specified endpoint.

3.12.2 Data Structure Documentation

3.12.2.1 struct_usb_device_video_mjpeg_payload_header_struct

Data Fields

- **uint8_t bHeaderLength**
The payload header length.
- **uint32_t dwPresentationTime**
Presentation time stamp (PTS) field.
- **uint8_t bSourceClockReference** [6]
Source clock reference (SCR) field.
- **uint8_t bmheaderInfo**
The payload header bitmap field.
- **uint8_t frameIdentifier**: 1U
Frame Identifier.
- **uint8_t endOfFrame**: 1U
End of Frame.
- **uint8_t presentationTimeStamp**: 1U
Presentation Time Stamp.
- **uint8_t sourceClockReference**: 1U
Source Clock Reference.
- **uint8_t reserved**: 1U
Reserved.
- **uint8_t stillImage**: 1U
Still Image.

- uint8_t **errorBit**: 1U
Error Bit.
- uint8_t **endOfHeader**: 1U
End of Header.
- uint8_t **FID**: 1U
Frame Identifier.
- uint8_t **EOI**: 1U
End of Frame.
- uint8_t **PTS**: 1U
Presentation Time Stamp.
- uint8_t **SCR**: 1U
Source Clock Reference.
- uint8_t **RES**: 1U
Reserved.
- uint8_t **STI**: 1U
Still Image.
- uint8_t **ERR**: 1U
Error Bit.
- uint8_t **EOH**: 1U
End of Header.

3.12.2.1.0.25 Field Documentation

3.12.2.1.0.25.1 uint8_t usb_device_video_mjpeg_payload_header_struct_t::bHeaderLength

3.12.2.1.0.25.2 uint8_t usb_device_video_mjpeg_payload_header_struct_t::bmheaderInfo

3.12.2.1.0.25.3 uint8_t usb_device_video_mjpeg_payload_header_struct_t::frameIdentifier

This bit toggles at each frame start boundary and stays constant for the rest of the frame.

3.12.2.1.0.25.4 uint8_t usb_device_video_mjpeg_payload_header_struct_t::endOfFrame

This bit indicates the end of a video frame and is set in the last video sample that belongs to a frame.

3.12.2.1.0.25.5 uint8_t usb_device_video_mjpeg_payload_header_struct_t::presentationTime-Stamp

This bit, when set, indicates the presence of a PTS field.

3.12.2.1.0.25.6 uint8_t usb_device_video_mjpeg_payload_header_struct_t::sourceClock-Reference

This bit, when set, indicates the presence of a SCR field.

3.12.2.1.0.25.7 uint8_t usb_device_video_mjpeg_payload_header_struct_t::reserved

Set to 0.

USB VIDEO Class driver

3.12.2.1.0.25.8 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::stillImage`

This bit, when set, identifies a video sample that belongs to a still image.

3.12.2.1.0.25.9 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::errorBit`

This bit, when set, indicates an error in the device streaming.

3.12.2.1.0.25.10 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::endOfHeader`

This bit, when set, indicates the end of the BFH fields.

3.12.2.1.0.25.11 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::FID`

This bit toggles at each frame start boundary and stays constant for the rest of the frame.

3.12.2.1.0.25.12 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::EOI`

This bit indicates the end of a video frame and is set in the last video sample that belongs to a frame.

3.12.2.1.0.25.13 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::PTS`

This bit, when set, indicates the presence of a PTS field.

3.12.2.1.0.25.14 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::SCR`

This bit, when set, indicates the presence of a SCR field.

3.12.2.1.0.25.15 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::RES`

Set to 0.

3.12.2.1.0.25.16 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::STI`

This bit, when set, identifies a video sample that belongs to a still image.

3.12.2.1.0.25.17 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::ERR`

This bit, when set, indicates an error in the device streaming.

3.12.2.1.0.25.18 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::EOH`

This bit, when set, indicates the end of the BFH fields.

3.12.2.1.0.25.19 `uint32_t usb_device_video_mjpeg_payload_header_struct_t::dwPresentation-Time`

3.12.2.1.0.25.20 `uint8_t usb_device_video_mjpeg_payload_header_struct_t::bSourceClock-Reference[6]`

3.12.2.2 `struct _usb_device_video_probe_and_commit_controls_struct`

Data Fields

- `uint8_t bFormatIndex`
Video format index from a format descriptor.
- `uint8_t bFrameIndex`
Video frame index from a frame descriptor.
- `uint32_t dwFrameInterval`
Frame interval in 100ns units.
- `uint16_t wKeyFrameRate`
Key frame rate in key-frame per video-frame units.
- `uint16_t wPFrameRate`
PFrame rate in PFrame/key frame units.
- `uint16_t wCompQuality`
Compression quality control in abstract units 0U (lowest) to 10000U (highest).
- `uint16_t wCompWindowSize`
Window size for average bit rate control.
- `uint16_t wDelay`
Internal video streaming interface latency in ms from video data capture to presentation on the USB.
- `uint32_t dwMaxVideoFrameSize`
Maximum video frame or codec-specific segment size in bytes.
- `uint32_t dwMaxPayloadTransferSize`
Specifies the maximum number of bytes that the device can transmit or receive in a single payload transfer.
- `uint32_t dwClockFrequency`
The device clock frequency in Hz for the specified format.
- `uint8_t bmFramingInfo`
Bit-field control supporting the following values: D0 Frame ID, D1 EOF.
- `uint8_t bPreferredVersion`
The preferred payload format version supported by the host or device for the specified bFormatIndex value.
- `uint8_t bMinVersion`
The minimum payload format version supported by the device for the specified bFormatIndex value.
- `uint8_t bMaxVersion`
The maximum payload format version supported by the device for the specified bFormatIndex value.
- `uint8_t bmHint`
Bit-field control indicating to the function what fields shall be kept fixed.
- `uint8_t dwFrameInterval: 1U`
dwFrameInterval field.
- `uint8_t wKeyFrameRate: 1U`
wKeyFrameRate field.
- `uint8_t wPFrameRate: 1U`
wPFrameRate field.
- `uint8_t wCompQuality: 1U`
wCompQuality field.
- `uint8_t wCompWindowSize: 1U`

USB VIDEO Class driver

- *wCompWindowSize* field.
uint8_t **reserved**: 3U
Reserved field.

3.12.2.2.0.26 Field Documentation

- 3.12.2.2.0.26.1** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bmHint`
- 3.12.2.2.0.26.2** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::dwFrameInterval`
- 3.12.2.2.0.26.3** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::wKeyFrameRate`
- 3.12.2.2.0.26.4** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::wPFrameRate`
- 3.12.2.2.0.26.5** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::wCompQuality`
- 3.12.2.2.0.26.6** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::wCompWindowSize`
- 3.12.2.2.0.26.7** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::reserved`
- 3.12.2.2.0.26.8** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bFormatIndex`
- 3.12.2.2.0.26.9** `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bFrameIndex`
- 3.12.2.2.0.26.10** `uint32_t usb_device_video_probe_and_commit_controls_struct_t::dwFrameInterval`
- 3.12.2.2.0.26.11** `uint16_t usb_device_video_probe_and_commit_controls_struct_t::wKeyFrameRate`
- 3.12.2.2.0.26.12** `uint16_t usb_device_video_probe_and_commit_controls_struct_t::wPFrameRate`
- 3.12.2.2.0.26.13** `uint16_t usb_device_video_probe_and_commit_controls_struct_t::wCompQuality`
- 3.12.2.2.0.26.14** `uint16_t usb_device_video_probe_and_commit_controls_struct_t::wCompWindowSize`
- 3.12.2.2.0.26.15** `uint16_t usb_device_video_probe_and_commit_controls_struct_t::wDelay`
- 3.12.2.2.0.26.16** `uint32_t usb_device_video_probe_and_commit_controls_struct_t::dwMaxVideoFrameSize`
- 3.12.2.2.0.26.17** `uint32_t usb_device_video_probe_and_commit_controls_struct_t::dwMaxPayloadTransferSize`
- 3.12.2.2.0.26.18** `uint32_t usb_device_video_probe_and_commit_controls_struct_t::dwClockFrequency`

This specifies the units used for the time information fields in the Video Payload Headers in the data stream.

USB VIDEO Class driver

3.12.2.2.0.26.19 `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bmFramingInfo`

3.12.2.2.0.26.20 `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bPreferredVersion`

3.12.2.2.0.26.21 `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bMinVersion`

3.12.2.2.0.26.22 `uint8_t usb_device_video_probe_and_commit_controls_struct_t::bMaxVersion`

3.12.2.3 `struct usb_device_video_still_probe_and_commit_controls_struct`

Data Fields

- `uint8_t bFormatIndex`
Video format index from a format descriptor.
- `uint8_t bFrameIndex`
Video frame index from a frame descriptor.
- `uint8_t bCompressionIndex`
Compression index from a frame descriptor.
- `uint32_t dwMaxVideoFrameSize`
Maximum still image size in bytes.
- `uint32_t dwMaxPayloadTransferSize`
Specifies the maximum number of bytes that the device can transmit or receive in a single payload transfer.

3.12.2.3.0.27 Field Documentation

3.12.2.3.0.27.1 `uint8_t usb_device_video_still_probe_and_commit_controls_struct_t::bFormatIndex`

3.12.2.3.0.27.2 `uint8_t usb_device_video_still_probe_and_commit_controls_struct_t::bFrameIndex`

3.12.2.3.0.27.3 `uint8_t usb_device_video_still_probe_and_commit_controls_struct_t::bCompressionIndex`

3.12.2.3.0.27.4 `uint32_t usb_device_video_still_probe_and_commit_controls_struct_t::dwMaxVideoFrameSize`

3.12.2.3.0.27.5 `uint32_t usb_device_video_still_probe_and_commit_controls_struct_t::dwMaxPayloadTransferSize`

3.12.2.4 `struct usb_device_video_entity_struct_t`

The structure is used to pass the video entity information filled by application. Such as entity id (unit or terminal ID), entity type (unit or terminal type), and terminal type if the entity is a terminal.

3.12.2.5 struct usb_device_video_entities_struct_t

The structure is used to pass the video entity informations filled by the application. The type of each entity is the `usb_device_video_entity_struct_t`. The structure pointer is kept in the `usb_device_interface_struct_t::classSpecific`, such as, if there are three entities(out terminal, camera terminal, and processing unit), the value of the count field is 3U and the entity field saves the every entity information.

3.12.2.6 struct usb_device_video_struct_t

Data Fields

- `usb_device_handle` handle
The device handle.
- `usb_device_class_config_struct_t * configStruct`
The configuration of the class.
- `usb_device_interface_struct_t * controlInterfaceHandle`
Current control interface handle.
- `usb_device_interface_struct_t * streamInterfaceHandle`
Current stream interface handle.
- `uint8_t configuration`
Current configuration.
- `uint8_t controlInterfaceNumber`
The control interface number of the class.
- `uint8_t controlAlternate`
Current alternate setting of the control interface.
- `uint8_t streamInterfaceNumber`
The stream interface number of the class.
- `uint8_t streamAlternate`
Current alternate setting of the stream interface.
- `uint8_t streamInPipeBusy`
Stream IN pipe busy flag.
- `uint8_t streamOutPipeBusy`
Stream OUT pipe busy flag.

3.12.2.6.0.28 Field Documentation

3.12.2.6.0.28.1 usb_device_class_config_struct_t* usb_device_video_struct_t::configStruct

3.12.3 Enumeration Type Documentation

3.12.3.1 enum usb_device_video_event_t

Enumerator

- kUSB_DeviceVideoEventStreamSendResponse*** Send data completed in stream pipe.
- kUSB_DeviceVideoEventStreamRecvResponse*** Data received in stream pipe.
- kUSB_DeviceVideoEventControlSendResponse*** Send data completed in video control pipe.
- kUSB_DeviceVideoEventClassRequestBuffer*** Get buffer to save the data of the video class-specific

USB VIDEO Class driver

request.

3.12.4 Function Documentation

3.12.4.1 `usb_status_t USB_DeviceVideoInit (uint8_t controllerId, usb_device_class_config_struct_t * config, class_handle_t * handle)`

This function is used to initialize the video class. This function can only be called by the [USB_Device-ClassInit](#).

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>config</i>	The class configuration information.
in	<i>handle</i>	An parameter used to return pointer of the video class handle to the caller.

Returns

A USB error code or `kStatus_USB_Success`.

3.12.4.2 `usb_status_t USB_DeviceVideoDeinit (class_handle_t handle)`

The function deinitializes the device video class. This function can only be called by the [USB_Device-ClassDeinit](#).

Parameters

in	<i>handle</i>	The video class handle received from usb_device_class_config_struct_t::classHandle .
----	---------------	------------------------------------------------------------------------------------------------------

Returns

A USB error code or `kStatus_USB_Success`.

3.12.4.3 `usb_status_t USB_DeviceVideoEvent (void * handle, uint32_t event, void * param)`

This function handles the event passed to the video class. This function can only be called by the [USB_DeviceClassEvent](#).

Parameters

in	<i>handle</i>	The video class handle received from the usb_device_class_config_struct_t::classHandle .
in	<i>event</i>	The event codes. See the enumeration usb_device_class_event_t .
in, out	<i>param</i>	The parameter type is determined by the event code.

Returns

A USB error code or kStatus_USB_Success.

Return values

<i>kStatus_USB_Success</i>	Free device handle successfully.
<i>kStatus_USB_Invalid-Parameter</i>	The device handle is not found.
<i>kStatus_USB_Invalid-Request</i>	The request is invalid and the control pipe is stalled by the caller.

3.12.4.4 `usb_status_t USB_DeviceVideoSend (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to send data through a specified endpoint. The function calls [USB_DeviceSend-Request](#) internally.

Parameters

in	<i>handle</i>	The video class handle received from usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length to be sent.

Returns

A USB error code or kStatus_USB_Success.

USB VIDEO Class driver

Note

The return value indicates whether the sending request is successful or not. The transfer done is notified by USB_DeviceVideoStreamIn or USB_DeviceVideoControlIn. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the endpoint callback).

3.12.4.5 `usb_status_t USB_DeviceVideoRecv (class_handle_t handle, uint8_t ep, uint8_t * buffer, uint32_t length)`

The function is used to receive data through a specified endpoint. The function calls the [USB_DeviceRecvRequest](#) internally.

Parameters

in	<i>handle</i>	The video class handle got from usb_device_class_config_struct_t::classHandle .
in	<i>ep</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length want to be received.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value indicates whether the receiving request is successful or not. The transfer done is notified by USB_DeviceVideoStreamOut. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the endpoint callback).

Chapter 4

USB Device driver

4.1 Overview

The USB device provides the device APIs to support the class driver and lite/non-lite application. It includes the USB controller driver only which consist of the common controller driver and xHCI driver.

Modules

- [USB Device Configuration](#)
- [USB Device Controller driver](#)
- [USB Device Spec Chapter 9 driver](#)

Data Structures

- struct [usb_device_endpoint_callback_message_struct_t](#)
Endpoint callback message structure. [More...](#)
- struct [usb_device_endpoint_callback_struct_t](#)
Endpoint callback structure. [More...](#)
- struct [usb_device_endpoint_init_struct_t](#)
Endpoint initialization structure. [More...](#)
- struct [usb_device_endpoint_status_struct_t](#)
Endpoint status structure. [More...](#)

Macros

- #define [USB_CONTROL_ENDPOINT](#) (0U)
Control endpoint index.
- #define [USB_CONTROL_MAX_PACKET_SIZE](#) (64U)
Control endpoint maxPacketSize.
- #define [USB_SETUP_PACKET_SIZE](#) (8U)
The setup packet size of USB control transfer.
- #define [USB_ENDPOINT_NUMBER_MASK](#) (0x0FU)
USB endpoint mask.
- #define [USB_UNINITIALIZED_VAL_32](#) (0xFFFFFFFFFU)
Default invalid value or the endpoint callback length of cancelled transfer.

Typedefs

- typedef [usb_status_t](#)(* [usb_device_endpoint_callback_t](#))([usb_device_handle](#) handle, [usb_device_endpoint_callback_message_struct_t](#) *message, void *callbackParam)
Endpoint callback function typedef.
- typedef [usb_status_t](#)(* [usb_device_callback_t](#))([usb_device_handle](#) handle, uint32_t callbackEvent, void *eventParam)
Device callback function typedef.

Enumerations

- enum `usb_device_status_t` {
 `kUSB_DeviceStatusTestMode` = 1U,
 `kUSB_DeviceStatusSpeed`,
 `kUSB_DeviceStatusOtg`,
 `kUSB_DeviceStatusDevice`,
 `kUSB_DeviceStatusEndpoint`,
 `kUSB_DeviceStatusDeviceState`,
 `kUSB_DeviceStatusAddress`,
 `kUSB_DeviceStatusSynchFrame`,
 `kUSB_DeviceStatusBus`,
 `kUSB_DeviceStatusBusSuspend`,
 `kUSB_DeviceStatusBusSleep`,
 `kUSB_DeviceStatusBusResume`,
 `kUSB_DeviceStatusRemoteWakeup`,
 `kUSB_DeviceStatusBusSleepResume` }
 Defines Get/Set status Types.
- enum `usb_device_state_t` {
 `kUSB_DeviceStateConfigured` = 0U,
 `kUSB_DeviceStateAddress`,
 `kUSB_DeviceStateDefault`,
 `kUSB_DeviceStateAddressing`,
 `kUSB_DeviceStateTestMode` }
 Defines USB 2.0 device state.
- enum `usb_device_endpoint_status_t` {
 `kUSB_DeviceEndpointStateIdle` = 0U,
 `kUSB_DeviceEndpointStateStalled` }
 Defines endpoint state.
- enum `usb_device_event_t` {

```

kUSB_DeviceEventBusReset = 1U,
kUSB_DeviceEventSuspend,
kUSB_DeviceEventResume,
kUSB_DeviceEventSleep,
kUSB_DeviceEventLPMResume,
kUSB_DeviceEventError,
kUSB_DeviceEventDetach,
kUSB_DeviceEventAttach,
kUSB_DeviceEventSetConfiguration,
kUSB_DeviceEventSetInterface,
kUSB_DeviceEventGetDeviceDescriptor,
kUSB_DeviceEventGetConfigurationDescriptor,
kUSB_DeviceEventGetStringDescriptor,
kUSB_DeviceEventGetHidDescriptor,
kUSB_DeviceEventGetHidReportDescriptor,
kUSB_DeviceEventGetHidPhysicalDescriptor,
kUSB_DeviceEventGetBOSDescriptor,
kUSB_DeviceEventGetDeviceQualifierDescriptor,
kUSB_DeviceEventVendorRequest,
kUSB_DeviceEventSetRemoteWakeup,
kUSB_DeviceEventGetConfiguration,
kUSB_DeviceEventGetInterface }

```

Available common EVENT types in device callback.

USB device APIs

- `usb_status_t USB_DeviceInit (uint8_t controllerId, usb_device_callback_t deviceCallback, usb_device_handle *handle)`
Initializes the USB device stack.
- `usb_status_t USB_DeviceRun (usb_device_handle handle)`
Enables the device functionality.
- `usb_status_t USB_DeviceStop (usb_device_handle handle)`
Disables the device functionality.
- `usb_status_t USB_DeviceDeinit (usb_device_handle handle)`
De-initializes the device controller.
- `usb_status_t USB_DeviceSendRequest (usb_device_handle handle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)`
Sends data through a specified endpoint.
- `usb_status_t USB_DeviceRecvRequest (usb_device_handle handle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)`
Receives data through a specified endpoint.
- `usb_status_t USB_DeviceCancel (usb_device_handle handle, uint8_t endpointAddress)`
Cancels the pending transfer in a specified endpoint.
- `usb_status_t USB_DeviceInitEndpoint (usb_device_handle handle, usb_device_endpoint_init_struct_t *epInit, usb_device_endpoint_callback_struct_t *epCallback)`
Initializes a specified endpoint.
- `usb_status_t USB_DeviceDeinitEndpoint (usb_device_handle handle, uint8_t endpointAddress)`

Data Structure Documentation

- Deinitializes a specified endpoint.*
- `usb_status_t USB_DeviceStallEndpoint (usb_device_handle handle, uint8_t endpointAddress)`
Stalls a specified endpoint.
- `usb_status_t USB_DeviceUnstallEndpoint (usb_device_handle handle, uint8_t endpointAddress)`
Unstalls a specified endpoint.
- `usb_status_t USB_DeviceGetStatus (usb_device_handle handle, usb_device_status_t type, void *param)`
Gets the status of the selected item.
- `usb_status_t USB_DeviceSetStatus (usb_device_handle handle, usb_device_status_t type, void *param)`
Sets the status of the selected item.
- `void USB_DeviceTaskFunction (void *deviceHandle)`
Device task function.
- `void USB_DeviceKhciIsrFunction (void *deviceHandle)`
Device KHCI ISR function.
- `void USB_DeviceEhciIsrFunction (void *deviceHandle)`
Device EHCI ISR function.
- `void USB_DeviceLpcIp3511IsrFunction (void *deviceHandle)`
Device LPC USB ISR function.
- `void USB_DeviceGetVersion (uint32_t *version)`
Gets the device stack version function.
- `usb_status_t USB_DeviceUpdateHwTick (usb_device_handle handle, uint64_t tick)`
Update the hardware tick.
- `#define USB_DeviceKhciTaskFunction(deviceHandle) USB_DeviceTaskFunction(deviceHandle)`
Device KHCI task function.
- `#define USB_DeviceEhciTaskFunction(deviceHandle) USB_DeviceTaskFunction(deviceHandle)`
Device EHCI task function.
- `#define USB_DeviceLpcIp3511TaskFunction(deviceHandle) USB_DeviceTaskFunction(deviceHandle)`
Device LPC ip3511 controller task function.

4.2 Data Structure Documentation

4.2.1 struct usb_device_endpoint_callback_message_struct_t

Data Fields

- `uint8_t * buffer`
Transferred buffer.
- `uint32_t length`
Transferred data length.
- `uint8_t isSetup`
Is in a setup phase.

4.2.2 struct usb_device_endpoint_callback_struct_t

Data Fields

- [usb_device_endpoint_callback_t callbackFn](#)
Endpoint callback function.
- void * [callbackParam](#)
Parameter for callback function.

4.2.3 struct usb_device_endpoint_init_struct_t

Data Fields

- uint16_t [maxPacketSize](#)
Endpoint maximum packet size.
- uint8_t [endpointAddress](#)
Endpoint address.
- uint8_t [transferType](#)
Endpoint transfer type.
- uint8_t [zlt](#)
ZLT flag.

4.2.4 struct usb_device_endpoint_status_struct_t

Data Fields

- uint8_t [endpointAddress](#)
Endpoint address.
- uint16_t [endpointStatus](#)
Endpoint status : idle or stalled.

4.3 Macro Definition Documentation

4.3.1 #define USB_SETUP_PACKET_SIZE (8U)

4.3.2 #define USB_DeviceKhciTaskFunction(*deviceHandle*) USB_DeviceTaskFunction(deviceHandle)

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.

Typedef Documentation

Parameters

in	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.3.3 #define USB_DeviceEhciTaskFunction(*deviceHandle*) USB_DeviceTaskFunction(deviceHandle)

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	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.3.4 #define USB_DeviceLpcIp3511TaskFunction(*deviceHandle*) USB_DeviceTaskFunction(deviceHandle)

The function is used to handle the LPC ip3511 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	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.4 Typedef Documentation

4.4.1 typedef usb_status_t(* usb_device_endpoint_callback_t)(usb_device_handle handle, usb_device_endpoint_callback_message_struct_t *message, void *callbackParam)

This callback function is used to notify the upper layer what the transfer result is. This callback pointer is passed when a specified endpoint is initialized by calling API [USB_DeviceInitEndpoint](#).

Parameters

<i>handle</i>	The device handle. It equals to the value returned from USB_DeviceInit .
<i>message</i>	The result of a transfer, which includes transfer buffer, transfer length, and whether is in a setup phase. phase for control pipe.
<i>callbackParam</i>	The parameter for this callback. It is same with usb_device_endpoint_callback_struct_t::callbackParam .

Returns

A USB error code or `kStatus_USB_Success`.

4.4.2 `typedef usb_status_t(* usb_device_callback_t)(usb_device_handle handle, uint32_t callbackEvent, void *eventParam)`

This callback function is used to notify the upper layer that the device status has changed. This callback pointer is passed by calling API [USB_DeviceInit](#).

Parameters

<i>handle</i>	The device handle. It equals the value returned from USB_DeviceInit .
<i>callbackEvent</i>	The callback event type. See enumeration usb_device_event_t .
<i>eventParam</i>	The event parameter for this callback. The parameter type is determined by the callback event.

Returns

A USB error code or `kStatus_USB_Success`.

4.5 Enumeration Type Documentation

4.5.1 `enum usb_device_status_t`

Enumerator

kUSB_DeviceStatusTestMode Test mode.
kUSB_DeviceStatusSpeed Current speed.
kUSB_DeviceStatusOtg OTG status.
kUSB_DeviceStatusDevice Device status.
kUSB_DeviceStatusEndpoint Endpoint state [usb_device_endpoint_status_t](#).
kUSB_DeviceStatusDeviceState Device state.
kUSB_DeviceStatusAddress Device address.
kUSB_DeviceStatusSynchFrame Current frame.
kUSB_DeviceStatusBus Bus status.
kUSB_DeviceStatusBusSuspend Bus suspend.

Enumeration Type Documentation

kUSB_DeviceStatusBusSleep Bus suspend.
kUSB_DeviceStatusBusResume Bus resume.
kUSB_DeviceStatusRemoteWakeup Remote wakeup state.
kUSB_DeviceStatusBusSleepResume Bus resume.

4.5.2 enum usb_device_state_t

Enumerator

kUSB_DeviceStateConfigured Device state, Configured.
kUSB_DeviceStateAddress Device state, Address.
kUSB_DeviceStateDefault Device state, Default.
kUSB_DeviceStateAddressing Device state, Address setting.
kUSB_DeviceStateTestMode Device state, Test mode.

4.5.3 enum usb_device_endpoint_status_t

Enumerator

kUSB_DeviceEndpointStateIdle Endpoint state, idle.
kUSB_DeviceEndpointStateStalled Endpoint state, stalled.

4.5.4 enum usb_device_event_t

Enumerator

kUSB_DeviceEventBusReset USB bus reset signal detected.
kUSB_DeviceEventSuspend USB bus suspend signal detected.
kUSB_DeviceEventResume USB bus resume signal detected. The resume signal is driven by itself or a host
kUSB_DeviceEventSleep USB bus LPM suspend signal detected.
kUSB_DeviceEventLPMResume USB bus LPM resume signal detected. The resume signal is driven by itself or a host
kUSB_DeviceEventError An error is happened in the bus.
kUSB_DeviceEventDetach USB device is disconnected from a host.
kUSB_DeviceEventAttach USB device is connected to a host.
kUSB_DeviceEventSetConfiguration Set configuration.
kUSB_DeviceEventSetInterface Set interface.
kUSB_DeviceEventGetDeviceDescriptor Get device descriptor.
kUSB_DeviceEventGetConfigurationDescriptor Get configuration descriptor.
kUSB_DeviceEventGetStringDescriptor Get string descriptor.

kUSB_DeviceEventGetHidDescriptor Get HID descriptor.
kUSB_DeviceEventGetHidReportDescriptor Get HID report descriptor.
kUSB_DeviceEventGetHidPhysicalDescriptor Get HID physical descriptor.
kUSB_DeviceEventGetBOSDescriptor Get configuration descriptor.
kUSB_DeviceEventGetDeviceQualifierDescriptor Get device qualifier descriptor.
kUSB_DeviceEventVendorRequest Vendor request.
kUSB_DeviceEventSetRemoteWakeup Enable or disable remote wakeup function.
kUSB_DeviceEventGetConfiguration Get current configuration index.
kUSB_DeviceEventGetInterface Get current interface alternate setting value.

4.6 Function Documentation

4.6.1 **usb_status_t USB_DeviceInit (uint8_t controllerId, usb_device_callback_t deviceCallback, usb_device_handle * handle)**

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

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration usb_controller_index_t .
in	<i>deviceCallback</i>	Function pointer of the device callback.
out	<i>handle</i>	It is an out parameter used to return the pointer of the device handle to the caller.

Return values

<i>kStatus_USB_Success</i>	The device is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer.
<i>kStatus_USB_Busy</i>	Cannot allocate a device handle.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller according to the controller id.
<i>kStatus_USB_Invalid-ControllerInterface</i>	The controller driver interfaces is invalid. There is an empty interface entity.
<i>kStatus_USB_Error</i>	The macro USB_DEVICE_CONFIG_ENDPOINTS is more than the IP's endpoint number. Or, the device has been initialized. Or, the mutex or message queue is created failed.

4.6.2 `usb_status_t USB_DeviceRun (usb_device_handle handle)`

The function enables the device functionality, so that the device can be recognized by the host when the device detects that it has been connected to a host.

Parameters

<i>in</i>	<i>handle</i>	The device handle got from USB_DeviceInit .
-----------	---------------	-------------------------------------------------------------

Return values

<i>kStatus_USB_Success</i>	The device is run successfully.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Invalid-Handle</i>	The device handle is a NULL pointer. Or the controller handle is invalid.

4.6.3 usb_status_t USB_DeviceStop (usb_device_handle *handle*)

The function disables the device functionality. After this function called, even if the device is detached to the host, it can't work.

Parameters

<i>in</i>	<i>handle</i>	The device handle received from USB_DeviceInit .
-----------	---------------	------------------------------------------------------------------

Return values

<i>kStatus_USB_Success</i>	The device is stopped successfully.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Invalid-Handle</i>	The device handle is a NULL pointer or the controller handle is invalid.

4.6.4 usb_status_t USB_DeviceDeinit (usb_device_handle *handle*)

The function de-initializes the device controller specified by the handle.

Parameters

<i>in</i>	<i>handle</i>	The device handle got from USB_DeviceInit .
-----------	---------------	-------------------------------------------------------------

Function Documentation

Return values

<i>kStatus_USB_Success</i>	The device is stopped successfully.
<i>kStatus_USB_Invalid-Handle</i>	The device handle is a NULL pointer or the controller handle is invalid.

4.6.5 **usb_status_t USB_DeviceSendRequest (usb_device_handle *handle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

The function is used to send data through a specified endpoint.

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent. The function is not reentrant.
in	<i>length</i>	The data length need to be sent.

Return values

<i>kStatus_USB_Success</i>	The send request is sent successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Busy</i>	Cannot allocate DTDS for current transfer in EHCI driver.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Error</i>	The device is doing reset.

Note

The return value indicates whether the sending request is successful or not. The transfer done is notified by the corresponding callback function. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for one specific endpoint, the application should implement a queue on the application level. The subsequent transfer can begin only when the previous transfer is done (get notification through the endpoint callback).

4.6.6 **usb_status_t USB_DeviceRecvRequest (usb_device_handle *handle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

The function is used to receive data through a specified endpoint. The function is not reentrant.

Function Documentation

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length want to be received.

Return values

<i>kStatus_USB_Success</i>	The receive request is sent successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Busy</i>	Cannot allocate DTDS for current transfer in EHCI driver.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Error</i>	The device is doing reset.

Note

The return value indicates whether the receiving request is successful or not. The transfer done is notified by the corresponding callback function. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for one specific endpoint, the application should implement a queue on the application level. The subsequent transfer can begin only when the previous transfer is done (get notification through the endpoint callback).

4.6.7 **usb_status_t USB_DeviceCancel (usb_device_handle *handle*, uint8_t *endpointAddress*)**

The function is used to cancel the pending transfer in a specified endpoint.

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
----	---------------	-------------------------------------------------------------

in	<i>endpoint-Address</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, and 0U - OUT.
----	-------------------------	-----------------------------------------------------------------------------

Return values

<i>kStatus_USB_Success</i>	The transfer is cancelled.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer or the controller handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.

4.6.8 **usb_status_t USB_DeviceInitEndpoint (usb_device_handle *handle*, usb_device_endpoint_init_struct_t * *epInit*, usb_device_endpoint_callback_struct_t * *epCallback*)**

The function is used to initialize a specified endpoint. The corresponding endpoint callback is also initialized.

Parameters

in	<i>handle</i>	The device handle received from USB_DeviceInit .
in	<i>epInit</i>	Endpoint initialization structure. See the structure usb_device_endpoint_init_struct_t .
in	<i>epCallback</i>	Endpoint callback structure. See the structure usb_device_endpoint_callback_struct_t .

Return values

<i>kStatus_USB_Success</i>	The endpoint is initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The epInit or epCallback is NULL pointer. Or the endpoint number is more than USB_DEVICE_CONFIG_ENDPOINTS.
<i>kStatus_USB_Busy</i>	The endpoint is busy in EHCI driver.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.

Function Documentation

4.6.9 `usb_status_t USB_DeviceDeinitEndpoint (usb_device_handle handle, uint8_t endpointAddress)`

The function is used to deinitializes a specified endpoint.

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>endpoint-Address</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, and 0U - OUT.

Return values

<i>kStatus_USB_Success</i>	The endpoint is de-initialized successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number is more than USB_DEVICE_CONFIG_ENDPOINTS.
<i>kStatus_USB_Busy</i>	The endpoint is busy in EHCI driver.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.

4.6.10 **usb_status_t USB_DeviceStallEndpoint (usb_device_handle *handle*, uint8_t *endpointAddress*)**

The function is used to stall a specified endpoint.

Parameters

in	<i>handle</i>	The device handle received from USB_DeviceInit .
in	<i>endpoint-Address</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, and 0U - OUT.

Return values

<i>kStatus_USB_Success</i>	The endpoint is stalled successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number is more than USB_DEVICE_CONFIG_ENDPOINTS.

Function Documentation

<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
----------------------------------------	-----------------------------

4.6.11 **usb_status_t USB_DeviceUnstallEndpoint (usb_device_handle *handle*, uint8_t *endpointAddress*)**

The function is used to unstall a specified endpoint.

Parameters

in	<i>handle</i>	The device handle received from USB_DeviceInit .
in	<i>endpoint-Address</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, and 0U - OUT.

Return values

<i>kStatus_USB_Success</i>	The endpoint is unstalled successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The endpoint number is more than USB_DEVICE_CONFIG_ENDPOINTS.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.

4.6.12 **usb_status_t USB_DeviceGetStatus (usb_device_handle *handle*, usb_device_status_t *type*, void * *param*)**

The function is used to get the status of the selected item.

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>type</i>	The selected item. See the structure usb_device_status_t .
out	<i>param</i>	The parameter type is determined by the selected item.

Return values

<i>kStatus_USB_Success</i>	Get status successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Invalid-Parameter</i>	The parameter is NULL pointer.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Error</i>	Unsupported type.

4.6.13 **usb_status_t USB_DeviceSetStatus (usb_device_handle *handle*, usb_device_status_t *type*, void * *param*)**

The function is used to set the status of the selected item.

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>type</i>	The selected item. See the structure usb_device_status_t .
in	<i>param</i>	The parameter type is determined by the selected item.

Return values

<i>kStatus_USB_Success</i>	Set status successfully.
<i>kStatus_USB_Invalid-Handle</i>	The handle is a NULL pointer. Or the controller handle is invalid.
<i>kStatus_USB_Controller-NotFound</i>	Cannot find the controller.
<i>kStatus_USB_Error</i>	Unsupported type or the parameter is NULL pointer.

4.6.14 **void USB_DeviceTaskFunction (void * *deviceHandle*)**

The function is used to handle the controller message. This function should not be called in the application directly.

Function Documentation

Parameters

in	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.6.15 void USB_DeviceKhcilsrFunction (void * *deviceHandle*)

The function is the KHCI interrupt service routine.

Parameters

in	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.6.16 void USB_DeviceEhcilsrFunction (void * *deviceHandle*)

The function is the EHCI interrupt service routine.

Parameters

in	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.6.17 void USB_DeviceLpc1p3511IsrFunction (void * *deviceHandle*)

The function is the LPC USB interrupt service routine.

Parameters

in	<i>deviceHandle</i>	The device handle got from USB_DeviceInit .
----	---------------------	-------------------------------------------------------------

4.6.18 void USB_DeviceGetVersion (uint32_t * *version*)

The function is used to get the device stack version.

Parameters

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

4.6.19 `usb_status_t` `USB_DeviceUpdateHwTick` (`usb_device_handle` *handle*, `uint64_t` *tick*)

The function is used to update the hardware tick.

Function Documentation

Parameters

in	<i>handle</i>	The device handle got from USB_DeviceInit .
in	<i>tick</i>	Current hardware tick(uint is ms).

4.7 USB Device Controller driver

4.7.1 Overview

The interface between KHCI/EHCI etc controller Driver and Common Controller driver.

Modules

- [USB Device Controller EHCI driver](#)
- [USB Device Controller KHCI driver](#)
- [USB Device Controller LPC IP3511 driver](#)

Data Structures

- struct [usb_device_callback_message_struct_t](#)
Device notification message structure. [More...](#)
- struct [usb_device_controller_interface_struct_t](#)
USB device controller interface structure. [More...](#)
- struct [usb_device_struct_t](#)
USB device status structure. [More...](#)

Macros

- `#define usb_device_controller_handle usb_device_handle`
Macro to define controller handle.

Typedefs

- typedef [usb_status_t](#)(* [usb_device_controller_init_t](#))(uint8_t controllerId, [usb_device_handle](#) handle, [usb_device_controller_handle](#) *controllerHandle)
USB device controller initialization function typedef.
- typedef [usb_status_t](#)(* [usb_device_controller_deinit_t](#))([usb_device_controller_handle](#) controllerHandle)
USB device controller de-initialization function typedef.
- typedef [usb_status_t](#)(* [usb_device_controller_send_t](#))([usb_device_controller_handle](#) controllerHandle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)
USB device controller send data function typedef.
- typedef [usb_status_t](#)(* [usb_device_controller_recv_t](#))([usb_device_controller_handle](#) controllerHandle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)
USB device controller receive data function typedef.
- typedef [usb_status_t](#)(* [usb_device_controller_cancel_t](#))([usb_device_controller_handle](#) controllerHandle, uint8_t endpointAddress)
USB device controller cancel transfer function in a specified endpoint typedef.
- typedef [usb_status_t](#)(* [usb_device_controller_control_t](#))([usb_device_controller_handle](#) controllerHandle, [usb_device_control_type_t](#) command, void *param)

USB Device Controller driver

USB device controller control function typedef.

Enumerations

- enum `usb_device_notification_t` {
 `kUSB_DeviceNotifyBusReset` = 0x10U,
 `kUSB_DeviceNotifySuspend`,
 `kUSB_DeviceNotifyResume`,
 `kUSB_DeviceNotifyLPMSleep`,
 `kUSB_DeviceNotifyLPMResume`,
 `kUSB_DeviceNotifyError`,
 `kUSB_DeviceNotifyDetach`,
 `kUSB_DeviceNotifyAttach` }
 Available notify types for device notification.
- enum `usb_device_control_type_t` {
 `kUSB_DeviceControlRun` = 0U,
 `kUSB_DeviceControlStop`,
 `kUSB_DeviceControlEndpointInit`,
 `kUSB_DeviceControlEndpointDeinit`,
 `kUSB_DeviceControlEndpointStall`,
 `kUSB_DeviceControlEndpointUnstall`,
 `kUSB_DeviceControlGetDeviceStatus`,
 `kUSB_DeviceControlGetEndpointStatus`,
 `kUSB_DeviceControlSetDeviceAddress`,
 `kUSB_DeviceControlGetSynchFrame`,
 `kUSB_DeviceControlResume`,
 `kUSB_DeviceControlSleepResume`,
 `kUSB_DeviceControlSuspend`,
 `kUSB_DeviceControlSleep`,
 `kUSB_DeviceControlSetDefaultStatus`,
 `kUSB_DeviceControlGetSpeed`,
 `kUSB_DeviceControlGetOtgStatus`,
 `kUSB_DeviceControlSetOtgStatus`,
 `kUSB_DeviceControlSetTestMode`,
 `kUSB_DeviceControlGetRemoteWakeUp` }
 Control type for controller.

4.7.2 Data Structure Documentation

4.7.2.1 struct `usb_device_callback_message_struct_t`

Data Fields

- `uint8_t * buffer`

- *Transferred buffer.*
uint32_t **length**
- *Transferred data length.*
uint8_t **code**
- *Notification code.*
uint8_t **isSetup**
- *Is in a setup phase.*

4.7.2.2 struct usb_device_controller_interface_struct_t

Data Fields

- **usb_device_controller_init_t** deviceInit
Controller initialization.
- **usb_device_controller_deinit_t** deviceDeinit
Controller de-initialization.
- **usb_device_controller_send_t** deviceSend
Controller send data.
- **usb_device_controller_recv_t** deviceRecv
Controller receive data.
- **usb_device_controller_cancel_t** deviceCancel
Controller cancel transfer.
- **usb_device_controller_control_t** deviceControl
Controller control.

4.7.2.3 struct usb_device_struct_t

Data Fields

- volatile uint64_t **hwTick**
Current hw tick(ms)
- **usb_device_controller_handle** controllerHandle
Controller handle.
- const
usb_device_controller_interface_struct_t * controllerInterface
Controller interface handle.
- **usb_device_callback_t** deviceCallback
Device callback function pointer.
- **usb_device_endpoint_callback_struct_t** epCallback [USB_DEVICE_CONFIG_ENDPOINTS<< 1U]
Endpoint callback function structure.
- uint8_t **deviceAddress**
Current device address.
- uint8_t **controllerId**
Controller ID.
- uint8_t **state**
Current device state.
- uint8_t **remotewakeup**
Remote wakeup is enabled or not.

USB Device Controller driver

- `uint8_t isResetting`
Is doing device reset or not.

4.7.3 Enumeration Type Documentation

4.7.3.1 `enum usb_device_notification_t`

Enumerator

kUSB_DeviceNotifyBusReset Reset signal detected.
kUSB_DeviceNotifySuspend Suspend signal detected.
kUSB_DeviceNotifyResume Resume signal detected.
kUSB_DeviceNotifyLPMSleep LPM signal detected.
kUSB_DeviceNotifyLPMResume Resume signal detected.
kUSB_DeviceNotifyError Errors happened in bus.
kUSB_DeviceNotifyDetach Device disconnected from a host.
kUSB_DeviceNotifyAttach Device connected to a host.

4.7.3.2 `enum usb_device_control_type_t`

Enumerator

kUSB_DeviceControlRun Enable the device functionality.
kUSB_DeviceControlStop Disable the device functionality.
kUSB_DeviceControlEndpointInit Initialize a specified endpoint.
kUSB_DeviceControlEndpointDeinit De-initialize a specified endpoint.
kUSB_DeviceControlEndpointStall Stall a specified endpoint.
kUSB_DeviceControlEndpointUnstall Unstall a specified endpoint.
kUSB_DeviceControlGetDeviceStatus Get device status.
kUSB_DeviceControlGetEndpointStatus Get endpoint status.
kUSB_DeviceControlSetDeviceAddress Set device address.
kUSB_DeviceControlGetSynchFrame Get current frame.
kUSB_DeviceControlResume Drive controller to generate a resume signal in USB bus.
kUSB_DeviceControlSleepResume Drive controller to generate a LPM resume signal in USB bus.
kUSB_DeviceControlSuspend Drive controller to enetr into suspend mode.
kUSB_DeviceControlSleep Drive controller to enetr into sleep mode.
kUSB_DeviceControlSetDefaultStatus Set controller to default status.
kUSB_DeviceControlGetSpeed Get current speed.
kUSB_DeviceControlGetOtgStatus Get OTG status.
kUSB_DeviceControlSetOtgStatus Set OTG status.
kUSB_DeviceControlSetTestMode Drive xCHI into test mode.
kUSB_DeviceControlGetRemoteWakeUp Get flag of LPM Remote Wake-up Enabled by USB host.

4.7.4 USB Device Controller KHCI driver

4.7.4.1 Overview

Data Structures

- struct `usb_device_khci_endpoint_state_struct_t`
Endpoint state structure. [More...](#)
- struct `usb_device_khci_state_struct_t`
KHCI state structure. [More...](#)

Macros

- #define `USB_DEVICE_MAX_FS_ISO_MAX_PACKET_SIZE` (1023U)
The maximum value of ISO maximum packet size for FS in USB specification 2.0.
- #define `USB_DEVICE_MAX_FS_NONE_ISO_MAX_PACKET_SIZE` (64U)
The maximum value of non-ISO maximum packet size for FS in USB specification 2.0.
- #define `USB_KHCI_BDT_SET_ADDRESS`(bdt_base, ep, direction, odd, address)
Set BDT buffer address.
- #define `USB_KHCI_BDT_SET_CONTROL`(bdt_base, ep, direction, odd, control)
Set BDT control fields.
- #define `USB_KHCI_BDT_GET_ADDRESS`(bdt_base, ep, direction, odd)
Get BDT buffer address.
- #define `USB_KHCI_BDT_GET_CONTROL`(bdt_base, ep, direction, odd)
Get BDT control fields.

USB device KHCI functions

- `usb_status_t USB_DeviceKhciInit` (uint8_t controllerId, `usb_device_handle` handle, `usb_device_controller_handle` *khciHandle)
Initializes the USB device KHCI instance.
- `usb_status_t USB_DeviceKhciDeinit` (`usb_device_controller_handle` khciHandle)
Deinitializes the USB device KHCI instance.
- `usb_status_t USB_DeviceKhciSend` (`usb_device_controller_handle` khciHandle, uint8_t endpoint-Address, uint8_t *buffer, uint32_t length)
Sends data through a specified endpoint.
- `usb_status_t USB_DeviceKhciRecv` (`usb_device_controller_handle` khciHandle, uint8_t endpoint-Address, uint8_t *buffer, uint32_t length)
Receives data through a specified endpoint.
- `usb_status_t USB_DeviceKhciCancel` (`usb_device_controller_handle` khciHandle, uint8_t ep)
Cancels the pending transfer in a specified endpoint.
- `usb_status_t USB_DeviceKhciControl` (`usb_device_controller_handle` khciHandle, `usb_device_control_type_t` type, void *param)
Controls the status of the selected item.

4.7.4.2 Data Structure Documentation

4.7.4.2.1 struct usb_device_khci_endpoint_state_struct_t

Data Fields

- uint8_t * [transferBuffer](#)
Address of buffer containing the data to be transmitted.
- uint32_t [transferLength](#)
Length of data to transmit.
- uint32_t [transferDone](#)
The data length has been transferred.
- uint32_t [state](#)
The state of the endpoint.
- uint32_t [maxPacketSize](#): 10U
The maximum packet size of the endpoint.
- uint32_t [stalled](#): 1U
The endpoint is stalled or not.
- uint32_t [data0](#): 1U
The data toggle of the transaction.
- uint32_t [bdtOdd](#): 1U
The BDT toggle of the endpoint.
- uint32_t [dmaAlign](#): 1U
Whether the transferBuffer is DMA aligned or not.
- uint32_t [transferring](#): 1U
The endpoint is transferring.
- uint32_t [zlt](#): 1U
zlt flag

4.7.4.2.1.1 Field Documentation

4.7.4.2.1.1.1 uint32_t usb_device_khci_endpoint_state_struct_t::transferLength

4.7.4.2.2 struct usb_device_khci_state_struct_t

Data Fields

- [usb_device_struct_t](#) * [deviceHandle](#)
Device handle used to identify the device object belongs to.
- uint8_t * [bdt](#)
BDT buffer address.
- USB_Type * [registerBase](#)
The base address of the register.
- uint8_t [setupPacketBuffer](#) [USB_SETUP_PACKET_SIZE *2]
The setup request buffer.
- uint8_t * [dmaAlignBuffer](#)
This buffer is used to fix the transferBuffer or transferLength does not align to 4-bytes when the function USB_DeviceKhciRecv is called.
- [usb_device_khci_endpoint_state_struct_t](#) [endpointState](#) [USB_DEVICE_CONFIG_ENDPOINTS *2]
Endpoint state structures.

- `uint8_t isDmaAlignBufferInusing`
The dmaAlignBuffer is used or not.
- `uint8_t isResetting`
Is doing device reset or not.
- `uint8_t controllerId`
Controller ID.
- `uint8_t setupBufferIndex`
A valid setup buffer flag.

4.7.4.2.2.1 Field Documentation

4.7.4.2.2.1.1 `uint8_t* usb_device_khci_state_struct_t::dmaAlignBuffer`

The macro `USB_DEVICE_CONFIG_KHCI_DMA_ALIGN` is used to enable or disable this feature. If the feature is enabled, when the `transferBuffer` or `transferLength` does not align to 4-bytes, the `transferLength` is not more than `USB_DEVICE_CONFIG_KHCI_DMA_ALIGN_BUFFER_LENGTH`, and the flag `isDmaAlignBufferInusing` is zero, the `dmaAlignBuffer` is used to receive data and the flag `isDmaAlignBufferInusing` is set to 1. When the transfer is done, the received data, kept in `dmaAlignBuffer`, is copied to the `transferBuffer`, and the flag `isDmaAlignBufferInusing` is cleared.

4.7.4.3 Function Documentation

4.7.4.3.1 `usb_status_t USB_DeviceKhciInit (uint8_t controllerId, usb_device_handle handle, usb_device_controller_handle * khciHandle)`

This function initializes the USB device KHCI module specified by the `controllerId`.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration type <code>usb_controller_index_t</code> .
in	<i>handle</i>	Pointer of the device handle used to identify the device object belongs to.
out	<i>khciHandle</i>	An out parameter used to return the pointer of the device KHCI handle to the caller.

Returns

A USB error code or `kStatus_USB_Success`.

4.7.4.3.2 `usb_status_t USB_DeviceKhciDeinit (usb_device_controller_handle khciHandle)`

This function deinitializes the USB device KHCI module.

USB Device Controller driver

Parameters

in	<i>khciHandle</i>	Pointer of the device KHCI handle.
----	-------------------	------------------------------------

Returns

A USB error code or kStatus_USB_Success.

4.7.4.3.3 **usb_status_t USB_DeviceKhciSend (usb_device_controller_handle *khciHandle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

This function sends data through a specified endpoint.

Parameters

in	<i>khciHandle</i>	Pointer of the device KHCI handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length need to be sent.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value indicates whether the sending request is successful or not. The transfer completion is notified by the corresponding callback function. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is obtained through the endpoint callback).

4.7.4.3.4 **usb_status_t USB_DeviceKhciRecv (usb_device_controller_handle *khciHandle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

This function receives data through a specified endpoint.

Parameters

in	<i>khciHandle</i>	Pointer of the device KHCI handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length to be received.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value indicates whether the receiving request is successful or not. The transfer completion is notified by the corresponding callback function. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is obtained through the endpoint callback).

4.7.4.3.5 usb_status_t USB_DeviceKhciCancel (usb_device_controller_handle *khciHandle*, uint8_t *ep*)

The function is used to cancel the pending transfer in a specified endpoint.

Parameters

in	<i>khciHandle</i>	Pointer of the device KHCI handle.
in	<i>ep</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, abd 0U - OUT.

Returns

A USB error code or kStatus_USB_Success.

4.7.4.3.6 usb_status_t USB_DeviceKhciControl (usb_device_controller_handle *khciHandle*, usb_device_control_type_t *type*, void * *param*)

The function is used to control the status of the selected item.

USB Device Controller driver

Parameters

in	<i>khciHandle</i>	Pointer of the device KHCI handle.
in	<i>type</i>	The selected item. See enumeration type <code>usb_device_control_type_t</code> .
in, out	<i>param</i>	The parameter type is determined by the selected item.

Returns

A USB error code or `kStatus_USB_Success`.

4.7.5 USB Device Controller EHCI driver

4.7.5.1 Overview

Data Structures

- struct [usb_device_ehci_state_struct_t](#)
EHCI state structure. [More...](#)

Macros

- #define [USB_DEVICE_MAX_HS_ISO_MAX_PACKET_SIZE](#) (1024U)
The maximum value of ISO type maximum packet size for HS in USB specification 2.0.
- #define [USB_DEVICE_MAX_HS_INTERRUPT_MAX_PACKET_SIZE](#) (1024U)
The maximum value of interrupt type maximum packet size for HS in USB specification 2.0.
- #define [USB_DEVICE_MAX_HS_BULK_MAX_PACKET_SIZE](#) (512U)
The maximum value of bulk type maximum packet size for HS in USB specification 2.0.
- #define [USB_DEVICE_MAX_HS_CONTROL_MAX_PACKET_SIZE](#) (64U)
The maximum value of control type maximum packet size for HS in USB specification 2.0.

USB device EHCI functions

- [usb_status_t](#) [USB_DeviceEhciInit](#) ([uint8_t](#) controllerId, [usb_device_handle](#) handle, [usb_device_controller_handle](#) *ehciHandle)
Initializes the USB device EHCI instance.
- [usb_status_t](#) [USB_DeviceEhciDeinit](#) ([usb_device_controller_handle](#) ehciHandle)
Deinitializes the USB device EHCI instance.
- [usb_status_t](#) [USB_DeviceEhciSend](#) ([usb_device_controller_handle](#) ehciHandle, [uint8_t](#) endpoint-Address, [uint8_t](#) *buffer, [uint32_t](#) length)
Sends data through a specified endpoint.
- [usb_status_t](#) [USB_DeviceEhciRecv](#) ([usb_device_controller_handle](#) ehciHandle, [uint8_t](#) endpoint-Address, [uint8_t](#) *buffer, [uint32_t](#) length)
Receive data through a specified endpoint.
- [usb_status_t](#) [USB_DeviceEhciCancel](#) ([usb_device_controller_handle](#) ehciHandle, [uint8_t](#) ep)
Cancels the pending transfer in a specified endpoint.
- [usb_status_t](#) [USB_DeviceEhciControl](#) ([usb_device_controller_handle](#) ehciHandle, [usb_device_control_type_t](#) type, void *param)
Controls the status of the selected item.

4.7.5.2 Data Structure Documentation

4.7.5.2.1 struct [usb_device_ehci_state_struct_t](#)

Data Fields

- [usb_device_struct_t](#) * [deviceHandle](#)
Device handle used to identify the device object is belonged to.

USB Device Controller driver

- USBHS_Type * [registerBase](#)
The base address of the register.
- USBPHY_Type * [registerPhyBase](#)
The base address of the PHY register.
- usb_device_ehci_qh_struct_t * [qh](#)
The QH structure base address.
- usb_device_ehci_dtd_struct_t * [dtd](#)
The DTD structure base address.
- usb_device_ehci_dtd_struct_t * [dtdFree](#)
The idle DTD list head.
- usb_device_ehci_dtd_struct_t * [dtdHard](#) [USB_DEVICE_CONFIG_ENDPOINTS *2]
The transferring DTD list head for each endpoint.
- usb_device_ehci_dtd_struct_t * [dtdTail](#) [USB_DEVICE_CONFIG_ENDPOINTS *2]
The transferring DTD list tail for each endpoint.
- int8_t [dtdCount](#)
The idle DTD node count.
- uint8_t [endpointCount](#)
The endpoint number of EHCI.
- uint8_t [isResetting](#)
Whether a PORT reset is occurring or not.
- uint8_t [controllerId](#)
Controller ID.
- uint8_t [speed](#)
Current speed of EHCI.
- uint8_t [isSuspending](#)
Is suspending of the PORT.

4.7.5.3 Function Documentation

4.7.5.3.1 `usb_status_t USB_DeviceEhciInit (uint8_t controllerId, usb_device_handle handle, usb_device_controller_handle * ehciHandle)`

This function initializes the USB device EHCI module specified by the controllerId.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration type <code>usb_controller_index_t</code> .
in	<i>handle</i>	Pointer of the device handle used to identify the device object is belonged to.
out	<i>ehciHandle</i>	An out parameter used to return the pointer of the device EHCI handle to the caller.

Returns

A USB error code or `kStatus_USB_Success`.

4.7.5.3.2 usb_status_t USB_DeviceEhciDeinit (usb_device_controller_handle *ehciHandle*)

This function deinitializes the USB device EHCI module.

USB Device Controller driver

Parameters

in	<i>ehciHandle</i>	Pointer of the device EHCI handle.
----	-------------------	------------------------------------

Returns

A USB error code or kStatus_USB_Success.

4.7.5.3.3 **usb_status_t USB_DeviceEhciSend (usb_device_controller_handle *ehciHandle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

This function sends data through a specified endpoint.

Parameters

in	<i>ehciHandle</i>	Pointer of the device EHCI handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length to be sent.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value means whether the sending request is successful or not. The transfer completion is indicated by the corresponding callback function. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is received through the endpoint callback).

4.7.5.3.4 **usb_status_t USB_DeviceEhciRecv (usb_device_controller_handle *ehciHandle*, uint8_t *endpointAddress*, uint8_t * *buffer*, uint32_t *length*)**

This function Receives data through a specified endpoint.

Parameters

in	<i>ehciHandle</i>	Pointer of the device EHCI handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length want to be received.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value just means if the receiving request is successful or not; the transfer done is notified by the corresponding callback function. Currently, only one transfer request can be supported for one specific endpoint. If there is a specific requirement to support multiple transfer requests for one specific endpoint, the application should implement a queue in the application level. The subsequent transfer could begin only when the previous transfer is done (get notification through the endpoint callback).

4.7.5.3.5 usb_status_t USB_DeviceEhciCancel (usb_device_controller_handle *ehciHandle*, uint8_t *ep*)

The function is used to cancel the pending transfer in a specified endpoint.

Parameters

in	<i>ehciHandle</i>	Pointer of the device EHCI handle.
in	<i>ep</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, 0U - OUT.

Returns

A USB error code or kStatus_USB_Success.

4.7.5.3.6 usb_status_t USB_DeviceEhciControl (usb_device_controller_handle *ehciHandle*, usb_device_control_type_t *type*, void * *param*)

The function is used to control the status of the selected item.

USB Device Controller driver

Parameters

in	<i>ehciHandle</i>	Pointer of the device EHCI handle.
in	<i>type</i>	The selected item. See enumeration type <code>usb_device_control_type_t</code> .
in, out	<i>param</i>	The parameter type is determined by the selected item.

Returns

A USB error code or `kStatus_USB_Success`.

4.7.6 USB Device Controller LPC IP3511 driver

4.7.6.1 Overview

Data Structures

- struct `usb_device_lpc3511ip_endpoint_state_struct_t`
Endpoint state structure. [More...](#)
- struct `usb_device_lpc3511ip_state_struct_t`
LPC USB controller (IP3511) state structure. [More...](#)

Macros

- #define `USB_DEVICE_IP3511_ENDPOINT_RESERVED_BUFFER_SIZE` (5 * 1024)
The reserved buffer size, the buffer is for the memory copy if the application transfer buffer is ((not 64 bytes alignment) || (not in the same 64K ram) || (HS && OUT && not multiple of 4))
- #define `USB_DEVICE_IP3511_BITS_FOR_RESERVED_BUFFER` ((`USB_DEVICE_IP3511_ENDPOINT_RESERVED_BUFFER_SIZE` + 63) / 64)
Use one bit to represent one reserved 64 bytes to allocate the buffer by uint of 64 bytes.
- #define `USB_DEVICE_IP3511_RESERVED_BUFFER_FOR_COPY` (`USB_DEVICE_CONFIG_LPCIP3511FS` + `USB_DEVICE_CONFIG_LPCIP3511HS`)
How many IPs support the reserved buffer.
- #define `USB_DEVICE_IP3511_DOUBLE_BUFFER_ENABLE` (1u)
Prime all the double endpoint buffer at the same time, if the transfer length is larger than max packet size.

USB device controller (IP3511) functions

- `usb_status_t USB_DeviceLpc3511IpInit` (uint8_t controllerId, `usb_device_handle` handle, `usb_device_controller_handle` *controllerHandle)
Initializes the USB device controller instance.
- `usb_status_t USB_DeviceLpc3511IpDeinit` (`usb_device_controller_handle` controllerHandle)
Deinitializes the USB device controller instance.
- `usb_status_t USB_DeviceLpc3511IpSend` (`usb_device_controller_handle` controllerHandle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)
Sends data through a specified endpoint.
- `usb_status_t USB_DeviceLpc3511IpRecv` (`usb_device_controller_handle` controllerHandle, uint8_t endpointAddress, uint8_t *buffer, uint32_t length)
Receives data through a specified endpoint.
- `usb_status_t USB_DeviceLpc3511IpCancel` (`usb_device_controller_handle` controllerHandle, uint8_t ep)
Cancels the pending transfer in a specified endpoint.
- `usb_status_t USB_DeviceLpc3511IpControl` (`usb_device_controller_handle` controllerHandle, `usb_device_control_type_t` type, void *param)
Controls the status of the selected item.

4.7.6.2 Data Structure Documentation

4.7.6.2.1 struct usb_device_lpc3511ip_endpoint_state_struct_t

Data Fields

- uint8_t * [transferBuffer](#)
Address of buffer containing the data to be transmitted.
- uint32_t [transferLength](#)
Length of data to transmit.
- uint32_t [transferDone](#)
The data length has been transferred.
- uint32_t [transferPrimedLength](#)
it may larger than transferLength, because the primed length may larger than the transaction length.
- uint8_t * [epPacketBuffer](#)
The max packet buffer for copying.
- uint32_t [state](#)
The state of the endpoint.
- uint32_t [maxPacketSize](#): 11U
The maximum packet size of the endpoint.
- uint32_t [stalled](#): 1U
The endpoint is stalled or not.
- uint32_t [transferring](#): 1U
The endpoint is transferring.
- uint32_t [zlt](#): 1U
zlt flag
- uint32_t [epPacketCopied](#): 1U
whether use the copy buffer
- uint32_t [epControlDefault](#): 5u
The EP command/status 26~30 bits.
- uint32_t [doubleBufferBusy](#): 2U
How many buffers are primed, for control endpoint it is not used.
- uint32_t [producerOdd](#): 1U
When priming one transaction, prime to this endpoint buffer.
- uint32_t [consumerOdd](#): 1U
When transaction is done, read result from this endpoint buffer.

4.7.6.2.1.1 Field Documentation

4.7.6.2.1.1.1 uint32_t usb_device_lpc3511ip_endpoint_state_struct_t::transferLength

4.7.6.2.1.1.2 uint32_t usb_device_lpc3511ip_endpoint_state_struct_t::transferPrimedLength

4.7.6.2.2 struct usb_device_lpc3511ip_state_struct_t

Data Fields

- uint8_t * [controlData](#)
< control data buffer, must align with 64
- uint8_t * [setupData](#)
4 bytes for zero length transaction, must align with 64

- `usb_device_handle deviceHandle`
(4 bytes) Device handle used to identify the device object belongs to
- `USB_LPC3511IP_Type * registerBase`
(4 bytes) ip base address
- `uint8_t controllerId`
Controller ID.
- `uint8_t isResetting`
Is doing device reset or not.
- `uint8_t deviceSpeed`
some controller support the HS

4.7.6.2.2.1 Field Documentation

4.7.6.2.2.1.1 `uint8_t* usb_device_lpc3511ip_state_struct_t::controlData`

8 bytes' setup data, must align with 64

4.7.6.3 Macro Definition Documentation

4.7.6.3.1 `#define USB_DEVICE_IP3511_BITS_FOR_RESERVED_BUFFER ((USB_DEVICE_IP3511_ENDPOINT_RESERVED_BUFFER_SIZE + 63) / 64)`

4.7.6.4 Function Documentation

4.7.6.4.1 `usb_status_t USB_DeviceLpc3511Iplnit (uint8_t controllerId, usb_device_handle handle, usb_device_controller_handle * controllerHandle)`

This function initializes the USB device controller module specified by the controllerId.

Parameters

in	<i>controllerId</i>	The controller ID of the USB IP. See the enumeration type <code>usb_controller_index_t</code> .
in	<i>handle</i>	Pointer of the device handle used to identify the device object belongs to.
out	<i>controller-Handle</i>	An out parameter used to return the pointer of the device controller handle to the caller.

USB Device Controller driver

Returns

A USB error code or kStatus_USB_Success.

4.7.6.4.2 **usb_status_t USB_DeviceLpc3511pDeinit (usb_device_controller_handle *controllerHandle*)**

This function deinitializes the USB device controller module.

Parameters

in	<i>controller-Handle</i>	Pointer of the device controller handle.
----	--------------------------	------------------------------------------

Returns

A USB error code or kStatus_USB_Success.

4.7.6.4.3 **usb_status_t USB_DeviceLpc3511pSend (usb_device_controller_handle controllerHandle, uint8_t endpointAddress, uint8_t * buffer, uint32_t length)**

This function sends data through a specified endpoint.

Parameters

in	<i>controller-Handle</i>	Pointer of the device controller handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to hold the data need to be sent.
in	<i>length</i>	The data length need to be sent.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value indicates whether the sending request is successful or not. The transfer completion is notified by the corresponding callback function. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is obtained through the endpoint callback).

4.7.6.4.4 **usb_status_t USB_DeviceLpc3511pRecv (usb_device_controller_handle controllerHandle, uint8_t endpointAddress, uint8_t * buffer, uint32_t length)**

This function receives data through a specified endpoint.

USB Device Controller driver

Parameters

in	<i>controller-Handle</i>	Pointer of the device controller handle.
in	<i>endpoint-Address</i>	Endpoint index.
in	<i>buffer</i>	The memory address to save the received data.
in	<i>length</i>	The data length to be received.

Returns

A USB error code or kStatus_USB_Success.

Note

The return value indicates whether the receiving request is successful or not. The transfer completion is notified by the corresponding callback function. Currently, only one transfer request can be supported for a specific endpoint. If there is a specific requirement to support multiple transfer requests for a specific endpoint, the application should implement a queue in the application level. The subsequent transfer can begin only when the previous transfer is done (a notification is obtained through the endpoint callback).

4.7.6.4.5 **usb_status_t USB_DeviceLpc3511pCancel (usb_device_controller_handle controllerHandle, uint8_t ep)**

The function is used to cancel the pending transfer in a specified endpoint.

Parameters

in	<i>controller-Handle</i>	ointer of the device controller handle.
in	<i>ep</i>	Endpoint address, bit7 is the direction of endpoint, 1U - IN, abd 0U - OUT.

Returns

A USB error code or kStatus_USB_Success.

4.7.6.4.6 **usb_status_t USB_DeviceLpc3511pControl (usb_device_controller_handle controllerHandle, usb_device_control_type_t type, void * param)**

The function is used to control the status of the selected item.

Parameters

in	<i>controller-Handle</i>	Pointer of the device controller handle.
in	<i>type</i>	The selected item. Please refer to enumeration type <code>usb_device_control_type_t</code> .
in,out	<i>param</i>	The parameter type is determined by the selected item.

Returns

A USB error code or `kStatus_USB_Success`.

4.8 USB Device Spec Chapter 9 driver

4.8.1 Overview

Macros

- #define **USB_DEVICE_STATUS_SIZE** (0x02U)
Defines USB device status size when the host request to get device status.
- #define **USB_INTERFACE_STATUS_SIZE** (0x02U)
Defines USB device interface status size when the host request to get interface status.
- #define **USB_ENDPOINT_STATUS_SIZE** (0x02U)
Defines USB device endpoint status size when the host request to get endpoint status.
- #define **USB_CONFIGURE_SIZE** (0X01U)
Defines USB device configuration size when the host request to get current configuration.
- #define **USB_INTERFACE_SIZE** (0X01U)
Defines USB device interface alternate setting size when the host request to get interface alternate setting.
- #define **USB_GET_STATUS_DEVICE_MASK** (0x03U)
Defines USB device status mask.
- #define **USB_GET_STATUS_INTERFACE_MASK** (0x03U)
Defines USB device interface status mask.
- #define **USB_GET_STATUS_ENDPOINT_MASK** (0x03U)
Defines USB device endpoint status mask.

Enumerations

- enum **usb_device_control_read_write_sequence_t** {
 kUSB_DeviceControlPipeSetupStage = 0U,
 kUSB_DeviceControlPipeDataStage,
 kUSB_DeviceControlPipeStatusStage }
Control read and write sequence.

Functions

- **usb_status_t** **USB_DeviceControlPipeInit** (**usb_device_handle** handle, void *param)
Initializes the control pipes.

4.8.2 Enumeration Type Documentation

4.8.2.1 enum usb_device_control_read_write_sequence_t

Enumerator

kUSB_DeviceControlPipeSetupStage Setup stage.
kUSB_DeviceControlPipeDataStage Data stage.
kUSB_DeviceControlPipeStatusStage status stage

4.8.3 Function Documentation

4.8.3.1 `usb_status_t USB_DeviceControlPipeInit (usb_device_handle handle, void * param)`

The function is used to initialize the control pipes. This function should be called when event `kUSB_DeviceEventBusReset` is received.

USB Device Spec Chapter 9 driver

Parameters

in	<i>handle</i>	The device handle.
in	<i>param</i>	The event parameter.

Returns

A USB error code or kStatus_USB_Success.

4.9 USB Device Configuration

Chapter 5

USB OS Adapter

5.1 Overview

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

Note

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

Macros

- #define **USB_BIG_ENDIAN** (0U)
Define big endian.
- #define **USB_LITTLE_ENDIAN** (1U)
Define little endian.
- #define **ENDIANNESS USB_LITTLE_ENDIAN**
Define current endian.

Typedefs

- typedef void * **usb_osa_event_handle**
Define USB OSA event handle.
- typedef void * **usb_osa_sem_handle**
Define USB OSA semaphore handle.
- typedef void * **usb_osa_mutex_handle**
Define USB OSA mutex handle.
- typedef void * **usb_osa_msgq_handle**
Define USB OSA message queue handle.

Enumerations

- enum **usb_osa_status_t** {
 kStatus_USB_OSA_Success = 0x00U,
 kStatus_USB_OSA_Error,
 kStatus_USB_OSA_TimeOut }
USB OSA error code.
- enum **usb_osa_event_mode_t** {
 kUSB_OsaEventManualClear = 0U,
 kUSB_OsaEventAutoClear = 1U }
The event flags are cleared automatically or manually.

USB OSA Memory Management

- void * [USB_OsaMemoryAllocate](#) (uint32_t length)
Reserves the requested amount of memory in bytes.
- void [USB_OsaMemoryFree](#) (void *p)
Frees the memory previously reserved.

USB OSA Event

- [usb_osa_status_t USB_OsaEventCreate](#) ([usb_osa_event_handle](#) *handle, uint32_t flag)
Creates an event object with all flags cleared.
- [usb_osa_status_t USB_OsaEventDestroy](#) ([usb_osa_event_handle](#) handle)
Destroys a created event object.
- [usb_osa_status_t USB_OsaEventSet](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask)
Sets an event flag.
- [usb_osa_status_t USB_OsaEventWait](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask, uint32_t flag, uint32_t timeout, uint32_t *bitSet)
Waits for an event flag.
- [usb_osa_status_t USB_OsaEventCheck](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask, uint32_t *bitSet)
Checks an event flag.
- [usb_osa_status_t USB_OsaEventClear](#) ([usb_osa_event_handle](#) handle, uint32_t bitMask)
Clears an event flag.

USB OSA Semaphore

- [usb_osa_status_t USB_OsaSemCreate](#) ([usb_osa_sem_handle](#) *handle, uint32_t count)
Creates a semaphore with a given value.
- [usb_osa_status_t USB_OsaSemDestroy](#) ([usb_osa_sem_handle](#) handle)
Destroys a semaphore object.
- [usb_osa_status_t USB_OsaSemPost](#) ([usb_osa_sem_handle](#) handle)
Posts a semaphore.
- [usb_osa_status_t USB_OsaSemWait](#) ([usb_osa_sem_handle](#) handle, uint32_t timeout)
Waits on a semaphore.

USB OSA Mutex

- [usb_osa_status_t USB_OsaMutexCreate](#) ([usb_osa_mutex_handle](#) *handle)
Creates a mutex.
- [usb_osa_status_t USB_OsaMutexDestroy](#) ([usb_osa_mutex_handle](#) handle)
Destroys a mutex.
- [usb_osa_status_t USB_OsaMutexLock](#) ([usb_osa_mutex_handle](#) handle)
Waits for a mutex and locks it.
- [usb_osa_status_t USB_OsaMutexUnlock](#) ([usb_osa_mutex_handle](#) handle)
Unlocks a mutex.

USB OSA Message Queue

- [usb_osa_status_t USB_OsaMsgqCreate](#) ([usb_osa_msgq_handle](#) *handle, uint32_t count, uint32_t size)

- Creates a message queue.*
- `usb_osa_status_t USB_OsaMsgqDestroy (usb_osa_msgq_handle handle)`
- Destroys a message queue.*
- `usb_osa_status_t USB_OsaMsgqSend (usb_osa_msgq_handle handle, void *msg)`
- Sends a message.*
- `usb_osa_status_t USB_OsaMsgqRecv (usb_osa_msgq_handle handle, void *msg, uint32_t timeout)`
- Receives a message.*
- `usb_osa_status_t USB_OsaMsgqCheck (usb_osa_msgq_handle handle, void *msg)`
- Checks a message queue and receives a message if the queue is not empty.*

5.2 Enumeration Type Documentation

5.2.1 enum usb_osa_status_t

Enumerator

kStatus_USB_OSA_Success Success.
kStatus_USB_OSA_Error Failed.
kStatus_USB_OSA_TimeOut Timeout occurs while waiting.

5.2.2 enum usb_osa_event_mode_t

Enumerator

kUSB_OsaEventManualClear The flags of the event is cleared manually.
kUSB_OsaEventAutoClear The flags of the event is cleared automatically.

5.3 Function Documentation

5.3.1 void* USB_OsaMemoryAllocate (uint32_t length)

The function is used to reserve the requested amount of memory in bytes and initializes it to 0.

Parameters

<i>length</i>	Amount of bytes to reserve.
---------------	-----------------------------

Returns

Pointer to the reserved memory. NULL if memory can't be allocated.

5.3.2 void USB_OsaMemoryFree (void * p)

The function is used to free the memory block previously reserved.

Function Documentation

Parameters

<i>p</i>	Pointer to the start of the memory block previously reserved.
----------	---------------------------------------------------------------

5.3.3 `usb_osa_status_t USB_OsaEventCreate (usb_osa_event_handle * handle, uint32_t flag)`

This function creates an event object and sets its clear mode. If the clear mode is `kUSB_OsaEvent-AutoClear`, when a task gets the event flags, these flags are cleared automatically. If the clear mode is `kUSB_OsaEventManualClear`, the flags must be cleared manually.

Parameters

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

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_event_handle eventHandle;  
usb_osa_status_t      usbOsaStatus;  
usbOsaStatus = USB_OsaEventCreate(&eventHandle,  
    kUSB_OsaEventManualClear);
```

5.3.4 `usb_osa_status_t USB_OsaEventDestroy (usb_osa_event_handle handle)`

Parameters

<i>handle</i>	Pointer to the event object.
---------------	------------------------------

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventDestroy(eventHandle);
```

5.3.5 `usb_osa_status_t` `USB_OsaEventSet (usb_osa_event_handle handle, uint32_t bitMask)`

Sets specified flags for an event object.

Function Documentation

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to be set.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventSet(eventHandle, 0x01U);
```

5.3.6 usb_osa_status_t USB_OsaEventWait (usb_osa_event_handle *handle*, uint32_t *bitMask*, uint32_t *flag*, uint32_t *timeout*, uint32_t * *bitSet*)

This function waits for a combination of flags to be set in an event object. An applications can wait for any/all bits to be set. This function can get the flags that wake up the waiting task.

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to wait.
<i>flag</i>	Wait all flags or any flag to be set. 0U - wait any flag, others, wait all flags.
<i>timeout</i>	The maximum number of milliseconds to wait for the event. If the wait condition is not met, passing 0U waits indefinitely when the environment is an RTOS and returns the kStatus_OSA_Timeout immediately. Pass any value for the bare metal.
<i>bitSet</i>	Flags that wake up the waiting task are obtained by this parameter.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventWait(eventHandle, 0x01U, 0U, 0U, &bitSet);
```

5.3.7 `usb_osa_status_t` **USB_OsaEventCheck** (`usb_osa_event_handle` *handle*, `uint32_t` *bitMask*, `uint32_t` * *bitSet*)

This function checks for a combination of flags to be set in an event object.

Function Documentation

Parameters

<i>handle</i>	Pointer to the event object.
<i>bitMask</i>	Event flags to check.
<i>bitSet</i>	Flags have been set.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
uint32_t            bitSet;  
...  
usbOsaStatus = USB_OsaEventCheck(eventHandle, 0x01U, &bitSet);
```

5.3.8 usb_osa_status_t USB_OsaEventClear (usb_osa_event_handle *handle*, uint32_t *bitMask*)

This function clears flags of an event object.

Parameters

<i>handle</i>	Pointer to the event object
<i>bitMask</i>	Event flags to be cleared.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_status_t    usbOsaStatus;  
...  
usbOsaStatus = USB_OsaEventClear(eventHandle, 0x01U);
```

5.3.9 usb_osa_status_t USB_OsaSemCreate (usb_osa_sem_handle * *handle*, uint32_t *count*)

This function creates a semaphore and sets the default count.

Parameters

<i>handle</i>	It is an out parameter, which is used to return pointer of the semaphore object.
<i>count</i>	Initializes a value of the semaphore.

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaSemCreate(&semHandle, 1U);
```

5.3.10 usb_osa_status_t USB_OsaSemDestroy (usb_osa_sem_handle *handle*)

This function destroys a semaphore object.

Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

Returns

An USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemDestroy(semHandle);
```

5.3.11 usb_osa_status_t USB_OsaSemPost (usb_osa_sem_handle *handle*)

This function wakes up a task waiting on the semaphore. If a task is not pending, increases the semaphore's value.

Function Documentation

Parameters

<i>handle</i>	Pointer to the semaphore.
---------------	---------------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemPost(semHandle);
```

5.3.12 usb_osa_status_t USB_OsaSemWait (usb_osa_sem_handle *handle*, uint32_t *timeout*)

This function checks the semaphore's value. If it is positive, it decreases the semaphore's value and return kStatus_OSA_Success.

Parameters

<i>handle</i>	Pointer to the semaphore.
<i>timeout</i>	The maximum number of milliseconds to wait for the semaphore. If the wait condition is not met, passing 0U waits indefinitely when environment is RTOS. And return kStatus_OSA_Timeout immediately for bare metal no matter what value has been passed.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_sem_handle    semHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaSemWait(semHandle, 0U);
```

5.3.13 usb_osa_status_t USB_OsaMutexCreate (usb_osa_mutex_handle * *handle*)

This function creates a mutex and sets it to an unlocked status.

Parameters

<i>handle</i>	It is out parameter, which is used to return the pointer of the mutex object.
---------------	-------------------------------------------------------------------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
usbOsaStatus = USB_OsaMutexCreate(&mutexHandle);
```

5.3.14 usb_osa_status_t USB_OsaMutexDestroy (usb_osa_mutex_handle *handle*)

This function destroys a mutex and sets it to an unlocked status.

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;
usb_osa_status_t      usbOsaStatus;
...
usbOsaStatus = USB_OsaMutexDestroy(mutexHandle);
```

5.3.15 usb_osa_status_t USB_OsaMutexLock (usb_osa_mutex_handle *handle*)

This function checks the mutex status. If it is unlocked, it locks it and returns the kStatus_OSA_Success. Otherwise, it waits forever to lock in RTOS and returns the kStatus_OSA_Success immediately for bare metal.

Function Documentation

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexLock(mutexHandle);
```

5.3.16 usb_osa_status_t USB_OsaMutexUnlock (usb_osa_mutex_handle *handle*)

This function unlocks a mutex.

Parameters

<i>handle</i>	Pointer to the mutex.
---------------	-----------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_mutex_handle mutexHandle;  
usb_osa_status_t      usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMutexUnlock(mutexHandle);
```

5.3.17 usb_osa_status_t USB_OsaMsgqCreate (usb_osa_msgq_handle * *handle*, uint32_t *count*, uint32_t *size*)

This function creates a message queue.

Parameters

<i>handle</i>	It is an out parameter, which is used to return a pointer of the message queue object.
<i>count</i>	The count of elements in the queue.
<i>size</i>	Size of every elements in words.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
usbOsaStatus = USB_OsaMsgqCreate(msgqHandle, 8U, 4U);
```

5.3.18 usb_osa_status_t USB_OsaMsgqDestroy (usb_osa_msgq_handle *handle*)

This function destroys a message queue.

Parameters

<i>handle</i>	Pointer to a message queue.
---------------	-----------------------------

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle msgqHandle;
usb_osa_status_t    usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqDestroy(msgqHandle);
```

5.3.19 usb_osa_status_t USB_OsaMsgqSend (usb_osa_msgq_handle *handle*, void * *msg*)

This function sends a message to the tail of the message queue.

Function Documentation

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to a message to be put into the queue.

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqSend(msgqHandle, &message);
```

5.3.20 `usb_osa_status_t USB_OsaMsgqRecv (usb_osa_msgq_handle handle, void * msg, uint32_t timeout)`

This function receives a message from the head of the message queue.

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.
<i>timeout</i>	The maximum number of milliseconds to wait for a message. If the wait condition is not met, passing 0U waits indefinitely when an environment is RTOS and returns the <code>kStatus_OSA_Timeout</code> immediately for bare metal.

Returns

A USB OSA error code or `kStatus_OSA_Success`.

Example:

```
usb_osa_msgq_handle    msgqHandle;
message_struct_t       message;
usb_osa_status_t       usbOsaStatus;
...
usbOsaStatus = USB_OsaMsgqRecv(msgqHandle, &message, 0U);
```

5.3.21 `usb_osa_status_t USB_OsaMsgqCheck (usb_osa_msgq_handle handle, void * msg)`

This function checks a message queue and receives a message if the queue is not empty.

Parameters

<i>handle</i>	Pointer to a message queue.
<i>msg</i>	The pointer to save a received message.

Returns

A USB OSA error code or kStatus_OSA_Success.

Example:

```
usb_osa_msgq_handle    msgqHandle;  
message_struct_t       message;  
usb_osa_status_t       usbOsaStatus;  
...  
usbOsaStatus = USB_OsaMsgqCheck(msgqHandle, &message);
```


How to Reach Us:

Home Page:

nxp.com

Web Support:

nxp.com/support

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

NXP reserves the right to make changes without further notice to any products herein. NXP makes no warranty, representation, or guarantee regarding the suitability of its products for any particular purpose, nor does NXP assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages.

“Typical” parameters that may be provided in NXP data sheets and/or specifications can and do vary in different applications, and actual performance may vary over time. All operating parameters, including “typicals,” must be validated for each customer application by customer’s technical experts. Freescale does not convey any license under its patent rights nor the rights of others. NXP sells products pursuant to standard terms and conditions of sale, which can be found at the following address: nxp.com/SalesTermsandConditions.

NXP, the NXP logo, Freescale, the Freescale logo, Kinetis, and Processor Expert are trademarks of NXP. Tower is a trademark of NXP. All other product or service names are the property of their respective owners. Arm, Arm Powered logo, and Cortex are registered trademarks of ARM Limited (or its subsidiaries) in the EU and/or elsewhere. All rights reserved.

© 2018 NXP B.V.

Document Number: MCUXSDKUSBDAPIRM

Rev. 0

May 2018

