µC/USB Device™

Universal Serial Bus Device Stack

Reference Manual

Micriμm Press
Weston, FL 33326

µC/USB Device User's Manual

µC/USB-Device Reference Manual

API - Core

- USBD_OS_EP_LockRelease
- USBD_OS_DlyMs
- USBD_OS_CoreEventGet
- USBD_OS_CoreEventPut
- USBD_OS_DbgEventRdy
- USBD_OS_DbgEventWait

Device Functions

- USBD_Init
- USBD_DevAdd
- USBD_DevStart
- USBD_DevStop
- USBD_DevStateGet
- USBD_DevSetMS_VendorCode
- USBD_DevSelfPwrSet
- USBD_DevFrameNbrGet

USBD_Init

Description

Initialize USB device stack. This function is called by the application exactly once. This function initializes all the internal variables and modules used by the USB device stack.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_INIT_FAIL

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_Init()` must be called:

    a. Only once from a product’s application.

    b. After product’s OS has been initialized

    c. Before product’s application calls any USB device stack function(s).

2. Initialize USB device stack. This function is called by the application exactly once. This function initializes all the internal variables and modules used by the USB device stack.

USBD_DevAdd

Description

Adds a device to the stack.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_DevAdd (USBD_DEV_CFG      *p_dev_cfg,
                         USBD_BUS_FNCTS    *p_bus_fnct,
                         USBD_DRV_API      *p_drv_api,
                         USBD_DRV_CFG      *p_drv_cfg,
                         USBD_DRV_BSP_API  *p_bsp_api,
                         USBD_ERR          *p_err);
```

Arguments

p_dev_cfg

Pointer to specific USB device configuration

p_bus_fnct

Pointer to application specific structure that contains callback functions called on bus state changes.

p_drv_api

Pointer to specific USB device driver API.

p_drv_cfg

Pointer to specific USB device driver configuration.

p_bsp_api

Pointer to specific USB device board-specific API.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_DEV_ALLOC
USBD_ERR_EP_NONE_AVAIL
```

Returned Value

Device number, if no error(s).

`USBD_DEV_NBR_NONE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_DevStart

Description

Starts device stack. This function connects the device to the USB host.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_DevStart (CPU_INT08U   dev_nbr,
                     USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE

Returned Value

None.

Callers

Application.

Notes / Warnings

1. Device can be started only if state is either `USBD_DEV_STATE_NONE` or `USB_DEV_STATE_INIT`.

USBD_DevStop

Description

Stops device. This function disconnects the device from the USB host.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_DevStop (CPU_INT08U   dev_nbr,
                    USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_DevStateGet

Description

Gets current device state.

Files

usbd_core.h/usbd_core.c

Prototype

```
USBD_DEV_STATE  USBD_DevStateGet (CPU_INT08U   dev_nbr,
                                  USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR

Returned Value

Current device state, if no error(s).

USBD_DEV_STATE_NONE, otherwise.

Callers

Application,

Classes.

Notes / Warnings

None.

USBD_DevSetMS_VendorCode

Description

Sets the device Microsoft vendor code used by Microsoft OS descriptors.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_DevSetMS_VendorCode (CPU_INT08U   dev_nbr,
                                CPU_INT08U   vendor_code,
                                USBD_ERR    *p_err)
```

Arguments

dev_nbr

Device number.

vendor_code

Microsoft OS vendor code.

p_err

Pointer to variable that will receive the return error code from this function:

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR

Returned Value

None.

Callers

Application.

Notes / Warnings

1. The vendor code used MUST be different from any vendor bRequest value.

2. Microsoft OS descriptors must be enabled by setting `USBD_CFG_MS_OS_DESC_EN` to `DEF_ENABLED` before using this function.

USBD_DevSelfPwrSet

Description

Sets the powered state (self- or bus-powered) when the device is in the addressed state, before a configuration is set.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_DevSelfPwrSet (CPU_INT08U    dev_nbr,
                          CPU_BOOLEAN   self_pwr,
                          USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

self_pwr

The power source of the device:

DEF_TRUE if device is self-powered.
DEF_FALSE if device is bus-powered.

p_err

Pointer to variable that will receive the return error code from this function:

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_DevFrameNbrGet

Description

Gets the current frame number from the USB device driver.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT16U  USBD_DevFrameNbrGet (CPU_INT08U   dev_nbr,
                                 USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

p_err

Pointer to variable that will receive the return error code from this function:

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE

Returned Value

The current frame number.

Callers

Application.

Notes / Warnings

1. The frame number will always be in the range 0-2047 (11 bits).

2. Frame number returned to the caller contains the frame and microframe numbers. It is encoded using a 16-bit format: bit 0 to 10 contains the frame numer, bit 11 to 13 contains the microframe number. The caller must use the macros `USBD_FRAME_NBR_GET()` or `USBD_MICROFRAME_NBR_GET()` to get the frame or microframe number only.

Configuration Functions

- USBD_CfgAdd
- USBD_CfgOtherSpeed

USBD_CfgAdd

Description

Adds a configuration to the device.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_CfgAdd (       CPU_INT08U     dev_nbr,
                                CPU_INT08U     attrib,
                                CPU_INT16U     max_pwr,
                                USBD_DEV_SPD   spd,
                          const CPU_CHAR      *p_name,
                                USBD_ERR      *p_err);
```

Arguments

dev_nbr

Device number.

attrib

Configuration attributes.

USBD_DEV_ATTRIB_REMOTE_WAKEUP
USBD_DEV_ATTRIB_SELF_POWERED

max_pwr

Bus power required for this device (see Note #1).

spd

Configuration speed.

USBD_DEV_SPD_FULL

USBD_DEV_SPD_HIGH

p_name

Pointer to string describing the configuration (See Note #2).

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_ALLOC
USBD_ERR_CFG_INVALID_MAX_PWR

Returned Value

Configuration number, If no error(s).

USBD_CFG_NBR_NONE, otherwise.

Callers

Application.

Notes / Warnings

1. USB spec 2.0, section 7.2.1.3/4 defines power constrains for bus-powered devices:

    a. “A low-power function is one that draws up to one unit load from the USB cable when operational”

    b. “A function is defined as being high-power if, when fully powered, it draws over one but no more than five unit loads from the USB cable.”

2. A unit load is defined as 100mA, thus max_pwr argument should be between 0 mA and 500mA.

3. String support is optional, in this case '`p_name`' can be a `NULL` string pointer.

4. Configuration can only be added when the device is in either the `USBD_DEV_STATE_NONE` or `USB_DEV_STATE_INIT` states.

USBD_CfgOtherSpeed

Description

Associates a configuration with its other-speed counterpart.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_CfgOtherSpeed (CPU_INT08U   dev_nbr,
                          CPU_INT08U   cfg_nbr,
                          CPU_INT08U   cfg_other,
                          USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration number.

cfg_other

Other-speed configuration number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR

Returned Value

None,

Callers

Application.

Notes / Warnings

1. Configurations from high- and full-speed can be associated with each other to provide comparable functionality regardless of speed.

2. Configuration can only be added when the device is in either the `USBD_DEV_STATE_NONE` or `USB_DEV_STATE_INIT` states.

Interface Functions

- USBD_IF_Add
- USBD_IF_AltAdd
- USBD_IF_Grp

USBD_IF_Add

Description

Send data on CDC data class interface.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_INT08U  USBD_IF_Add (       CPU_INT08U        dev_nbr,
                                CPU_INT08U        cfg_nbr,
                                USBD_CLASS_DRV   *p_class_drv,
                                void             *p_class_arg,
                                CPU_INT08U        class_code,
                                CPU_INT08U        class_sub_code,
                                CPU_INT08U        class_protocol_code,
                          const CPU_CHAR         *p_name,
                                USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration index to add the interface.

p_class_drv

Pointer to interface driver.

p_class_arg

Pointer to interface driver argument.

class_code

Class code assigned by the USB-IF.

`class_sub_code`

Subclass code assigned by the USB-IF.

`class_protocol_code`

Protocol code assigned by the USB-IF.

`p_name`

Pointer to string describing the Interface.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
```

Returned Value

Interface number, If no error(s).

`USBD_IF_NBR_NONE`, otherwise.

Callers

Classes.

Notes / Warnings

None.

USBD_IF_AltAdd

Description

Adds an alternate setting to a specific interface.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_IF_AltAdd (       CPU_INT08U   dev_nbr,
                                   CPU_INT08U   cfg_nbr,
                                   CPU_INT08U   if_nbr,
                             const CPU_CHAR    *p_name,
                                   USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration number.

if_nbr

Interface number.

p_name

Pointer to alternate setting name.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_INVALID_NBR
USBD_ERR_IF_ALT_ALLOC
```

Returned Value

Interface alternate setting number, if no errors.

`USBD_IF_ALT_NBR_NONE`, otherwise.

Callers

Classes.

Notes / Warnings

None.

USBD_IF_Grp

Description

Creates an interface group.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_IF_Grp (       CPU_INT08U   dev_nbr,
                                CPU_INT08U   cfg_nbr,
                                CPU_INT08U   class_code,
                                CPU_INT08U   class_sub_code,
                                CPU_INT08U   class_protocol_code,
                                CPU_INT08U   if_start,
                                CPU_INT08U   if_cnt,
                          const CPU_CHAR    *p_name,
                                USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration index to add the interface.

p_class_drv

Pointer to interface driver.

p_class_arg

Pointer to interface driver argument.

class_code

Class code assigned by the USB-IF.

`class_sub_code`

Subclass code assigned by the USB-IF.

`class_protocol_code`

Protocol code assigned by the USB-IF.

`if_start`

Interface number of the first interface that is associated with this group

`if_cnt`

Number of consecutive interfaces that are associated with this group.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_INVALID_NBR
USBD_ERR_IF_GRP_NBR_IN_USE
USBD_ERR_IF_GRP_ALLOC
```

Returned Value

Interface group number, if no errors.

`USBD_IF_GRP_NBR_NONE`, otherwise.

Callers

Classes.

Notes / Warnings

None.

String Functions

- USBD_StrAdd
- USBD_StrIxGet

USBD_StrAdd

Description

Adds a string to the USB device.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_StrAdd (       CPU_INT08U   dev_nbr,
                    const  CPU_CHAR    *p_str,
                           USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

p_str

Pointer to string to add.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. USB spec 2.0 chapter 9.5 states "Where appropriate, descriptors contain references to string descriptors that provide displayable information describing a descriptor in human-readable form. The inclusion of string descriptors is optional. However, the reference fields within descriptors are mandatory. If a device does not support string descriptors, string reference fields must be reset to zero to indicate no string descriptor is available". Since string descriptors are optional, 'p_str' could be a NULL pointer.

USBD_StrIxGet

Description

Gets string index corresponding to a given string.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_StrIxGet (       CPU_INT08U   dev_nbr,
                           const  CPU_CHAR    *p_str);
```

Arguments

dev_nbr

Device number.

p_str

Pointer to string to find.

Returned Value

String index.

Callers

Classes.

Notes / Warnings

None.

Endpoints Functions

- USBD_CtrlTx
- USBD_CtrlRx
- USBD_BulkAdd
- USBD_BulkRx
- USBD_BulkRxAsync
- USBD_BulkTx
- USBD_BulkTxAsync
- USBD_IntrAdd
- USBD_IntrRx
- USBD_IntrRxAsync
- USBD_IntrTx
- USBD_IntrTxAsync
- USBD_IsocAdd
- USBD_IsocRxAsync
- USBD_IsocTxAsync
- USBD_EP_RxZLP
- USBD_EP_TxZLP
- USBD_EP_Abort

- USBD_EP_Stall
- USBD_EP_IsStalled
- USBD_EP_MaxPktSizeGet
- USBD_EP_MaxPhyNbrGet
- USBD_EP_MaxNbrOpenGet

USBD_CtrlTx

Description

Sends data on control IN endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_CtrlTx (CPU_INT08U    dev_nbr,
                         void         *p_buf,
                         CPU_INT32U    buf_len,
                         CPU_INT16U    timeout_ms,
                         CPU_BOOLEAN   end,
                         USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

p_buf

Pointer to buffer of data that will be sent

buf_len

Number of octets to transmit.

timeout_ms

Timeout in milliseconds.

end

End-of-transfer flag (see Note #1).

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets transmitted，if no errors.

0，otherwise.

Callers

Classes.

Notes / Warnings

1. If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the host.

2. This function can be only called from USB device class drivers during class specific setup request callbacks.

USBD_CtrlRx

Description

Receive data on control OUT endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_CtrlRx (CPU_INT08U    dev_nbr,
                         void         *p_buf,
                         CPU_INT32U    buf_len,
                         CPU_INT16U    timeout_ms,
                         USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

p_buf

Pointer to buffer of data that will be sent

buf_len

Number of octets to transmit.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets received If no error(s).

0，otherwise.

Callers

Classes.

Application.

Notes / Warnings

This function can be only called from USB device class drivers during class specific setup request callbacks.

USBD_BulkAdd

Description

Adds a bulk endpoint to alternate setting interface.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_BulkAdd (CPU_INT08U    dev_nbr,
                          CPU_INT08U    cfg_nbr,
                          CPU_INT08U    if_nbr,
                          CPU_INT08U    if_alt_nbr,
                          CPU_BOOLEAN   dir_in,
                          CPU_INT16U    max_pkt_len,
                          USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration number.

if_nbr

Interface number.

if_alt_nbr

Interface alternate setting number.

dir_in

Endpoint direction.

Value	Direction
DEF_YES	IN
DEF_NO	OUT

max_pkt_len

Endpoint maximum packet length (see Note #1).

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC

Returned Value

Endpoint address, if no error(s).

USBD_EP_ADDR_NONE, otherwise.

Callers

Classes.

Notes / Warnings

1. If the max_pkt_len argument is '0', the stack will allocate the first available bulk endpoint regardless its maximum packet size.

USBD_BulkRx

Description

Receives data on bulk OUT endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_BulkRx (CPU_INT08U   dev_nbr,
                         CPU_INT08U   ep_addr,
                         void        *p_buf,
                         CPU_INT32U   buf_len,
                         CPU_INT16U   timeout_ms,
                         USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to destination buffer to receive data

buf_len

Number of octets to receive.

timeout_ms

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEVINVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets received，If no error(s).

0，otherwise.

Callers

Classes.

Notes / Warnings

1. This function blocks until:
    a. All data is received, or
    b. An error occurred.
    c. Transfer does not complete in the period specified by `timeout_ms`.

USBD_BulkRxAsync

Description

Receives data on bulk OUT endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_BulkRxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to destination buffer to receive data

buf_len

Number of octets to receive.

async_fnct

Function that will be invoked upon completion of receive operation

`p_async_arg`

Pointer to argument that will be passed as parameter of `async_fnct`.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. The callback specified by `async_fnct` has the following prototype.

```
void  USBD_AsyncFnct (CPU_INT08U   dev_nbr,
                      CPU_INT08U   ep_addr,
                      void        *p_buf,
                      CPU_INT32U   buf_len,
                      CPU_INT32U   xfer_len,
                      void        *p_arg,
                      USBD_ERR     err);
```

Argument(s):

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to destination buffer to receive data.

`buf_len`

Buffer length.

`xfer_len`

Number of byte received.

`p_arg`

Pointer to function argument.

`err`

Error status.

```
USBD_ERR_NONE
USBD_ERR_EP_ABORT
```

USBD_BulkTx

Description

Sends data on bulk IN endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_BulkTx (CPU_INT08U    dev_nbr,
                         CPU_INT08U    ep_addr,
                         void         *p_buf,
                         CPU_INT32U    buf_len,
                         CPU_INT16U    timeout_ms,
                         CPU_BOOLEAN   end,
                         USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to buffer of data that will be transmitted.

buf_len

Number of octets to transmit.

timeout_ms

Timeout in milliseconds.

`end`

End-of-transfer flag (see Note #2).

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets transmitted，If no error(s).

0，otherwise.

Callers

Classes.

Notes / Warnings

1. This function blocks until:

    a. All data is transmitted, or

    b. An error occurred.

    c. Transfer does not complete in the period specified by `timeout_ms`.

2. If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the host.

USBD_BulkTxAsync

Description

Sends data on bulk IN endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_BulkTxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        CPU_BOOLEAN       end,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to buffer of data that will be transmitted

buf_len

Number of octets to transmit.

async_fnct

Function that will be invoked upon completion of transmit operation.

`p_async_arg`

Pointer to argument that will be passed as parameter of `async_fnct` (see Note #2).

`end`

End-of-transfer flag (see Note #1).

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the host.

2. The callback specified by `async_fnct` has the following prototype.

```
void  USB_AsyncFnct (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
```

```
                    void        *p_buf,
                    CPU_INT32U   buf_len,
                    CPU_INT32U   xfer_len,
                    void        *p_arg,
                    USBD_ERR     err);
```

Argument(s):

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to buffer of data that will be transmitted.

`buf_len`

Buffer length.

`xfer_len`

Number of byte transmitted.

`p_arg`

Pointer to function argument.

`err`

Error status.

`USBD_ERR_NONE`
`USBD_ERR_EP_ABORT`

USBD_IntrAdd

Description

Adds an interrupt endpoint to alternate setting interface.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_IntrAdd (CPU_INT08U    dev_nbr,
                          CPU_INT08U    cfg_nbr,
                          CPU_INT08U    if_nbr,
                          CPU_INT08U    if_alt_nbr,
                          CPU_BOOLEAN   dir_in,
                          CPU_INT16U    max_pkt_len,
                          CPU_INT16U    interval,
                          USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration number.

if_nbr

Interface number.

if_alt_nbr

Interface alternate setting number.

dir_in

Endpoint direction.

Value	Direction
DEF_YES	IN
DEF_NO	OUT

max_pkt_len

Endpoint maximum packet length (see Note #1).

interval

Endpoint interval in frames/microframes.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

Endpoint address, if no error(s).

USBD_EP_ADDR_NONE, otherwise.

Callers

Classes.

Notes / Warnings

1. If the `max_pkt_len` argument is '0', the stack will allocate the first available interrupt endpoint regardless its maximum packet size.

USBD_IntrRx

Description

Receives data on interrupt OUT endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_IntrRx (CPU_INT08U   dev_nbr,
                         CPU_INT08U   ep_addr,
                         void        *p_buf,
                         CPU_INT32U   buf_len,
                         CPU_INT16U   timeout_ms,
                         USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to destination buffer to receive data

buf_len

Number of octets to receive.

timeout_ms

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEVINVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets received, If no error(s).

0, otherwise.

Callers

Classes.

Notes / Warnings

1. This function blocks until:
    a. All data is received, or
    b. An error occurred.
    c. Transfer does not complete in the period specified by `timeout_ms`.

USBD_IntrRxAsync

Description

Receives data on interrupt OUT endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_IntrRxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to destination buffer to receive data

buf_len

Number of octets to receive.

async_fnct

Function that will be invoked upon completion of receive operation

`p_async_arg`

Pointer to argument that will be passed as parameter of async_fnct.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. The callback specified by `async_fnct` has the following prototype.

```
void  USB_AsyncFnct (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     void        *p_buf,
                     CPU_INT32U   buf_len,
                     CPU_INT32U   xfer_len,
                     void        *p_arg,
                     USBD_ERR     err);
```

Argument(s):

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to destination buffer to receive data.

`buf_len`

Buffer length.

`xfer_len`

Number of byte received.

`p_arg`

Pointer to function argument.

`err`

Error status.

`USBD_ERR_NONE`
`USBD_ERR_EP_ABORT`

USBD_IntrTx

Description

Sends data on interrupt IN endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT32U  USBD_IntrTx (CPU_INT08U    dev_nbr,
                         CPU_INT08U    ep_addr,
                         void         *p_buf,
                         CPU_INT32U    buf_len,
                         CPU_INT16U    timeout_ms,
                         CPU_BOOLEAN   end,
                         USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to buffer of data that will be transmitted.

buf_len

Number of octets to transmit.

timeout_ms

Timeout in milliseconds.

`end`

End-of-transfer flag (see Note #2).

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets transmitted, If no error(s).

0, otherwise.

Callers

Classes.

Notes / Warnings

1. This function blocks until:

    a. All data is transmitted, or

    b. An error occurred.

    c. Transfer does not complete in the period specified by `timeout_ms`.

2. If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the host.

USBD_IntrTxAsync

Description

Sends data on interrupt IN endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_IntrTxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        CPU_BOOLEAN       end,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to buffer of data that will be transmitted

buf_len

Number of octets to transmit.

async_fnct

Function that will be invoked upon completion of transmit operation.

`p_async_arg`

Pointer to argument that will be passed as parameter of `async_fnct`. (see Note #2)

`end`

End-of-transfer flag (see Note #1).

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the host.

2. The callback specified by `async_fnct` has the following prototype.

```
void  USB_AsyncFnct (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
```

```
                    void        *p_buf,
                    CPU_INT32U   buf_len,
                    CPU_INT32U   xfer_len,
                    void        *p_arg,
                    USBD_ERR     err);
```

Argument(s)

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to buffer of data that will be transmitted.

`buf_len`

Buffer length.

`xfer_len`

Number of byte transmitted.

`p_arg`

Pointer to function argument.

`err`

Error status.

`USBD_ERR_NONE`
`USBD_ERR_EP_ABORT`

USBD_IsocAdd

Description

Add an isochronous endpoint to alternate setting interface.

Files

usbd_core.h/usbd_core.c

Prototype

```
CPU_INT08U  USBD_IsocAdd (CPU_INT08U    dev_nbr,
                          CPU_INT08U    cfg_nbr,
                          CPU_INT08U    if_nbr,
                          CPU_INT08U    if_alt_nbr,
                          CPU_BOOLEAN   dir_in,
                          CPU_INT08U    attrib,
                          CPU_INT16U    max_pkt_len,
                          CPU_INT08U    transaction_frame,
                          CPU_INT16U    interval,
                          USBD_ERR     *p_err);
```

Arguments

dev_nbr

Device number.

cfg_nbr

Configuration number.

if_nbr

Interface number.

if_alt_nbr

Interface alternate setting number.

dir_in

Endpoint direction.

Value	Direction
DEF_YES	IN
DEF_NO	OUT

attrib

Isochronous endpoint synchronization and usage type attributes.

max_pkt_len

Endpoint maximum packet length (see Note #1).

transaction_frame

Endpoint transactions per (micro)frame (see Note #2).

interval

Endpoint interval in frames/microframes.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

Endpoint address, if no error(s).

USBD_EP_ADDR_NONE, otherwise.

Callers

Classes.

Notes / Warnings

1. If the `max_pkt_len` argument is '0', the stack will allocate the first available interrupt endpoint regardless its maximum packet size.

2. For full-speed endpoints, `transaction_frame` must be set to 1 since there is no support for high-bandwidth endpoints.

USBD_IsocRxAsync

Description

Receives data on isochronous OUT endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_IsocRxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to destination buffer to receive data

buf_len

Number of octets to receive.

async_fnct

Function that will be invoked upon completion of receive operation

`p_async_arg`

Pointer to argument that will be passed as parameter of async_fnct.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_QUEUING
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. The callback specified by `async_fnct` has the following prototype.

```
void  USB_AsyncFnct (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     void        *p_buf,
                     CPU_INT32U   buf_len,
                     CPU_INT32U   xfer_len,
                     void        *p_arg,
                     USBD_ERR     err);
```

Argument(s):

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to destination buffer to receive data.

`buf_len`

Buffer length.

`xfer_len`

Number of byte received.

`p_arg`

Pointer to function argument.

`err`

Error status.

```
USBD_ERR_NONE
USBD_ERR_EP_ABORT
```

USBD_IsocTxAsync

Description

Sends data on isochronous IN endpoint asynchronously.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_IsocTxAsync (CPU_INT08U        dev_nbr,
                        CPU_INT08U        ep_addr,
                        void             *p_buf,
                        CPU_INT32U        buf_len,
                        USBD_ASYNC_FNCT   async_fnct,
                        void             *p_async_arg,
                        USBD_ERR         *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_buf

Pointer to buffer of data that will be transmitted

buf_len

Number of octets to transmit.

async_fnct

Function that will be invoked upon completion of transmit operation.

`p_async_arg`

Pointer to argument that will be passed as parameter of `async_fnct`. (see Note #1)

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_QUEUING
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

1. The callback specified by `async_fnct` has the following prototype.

```
void  USB_AsyncFnct (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     void        *p_buf,
                     CPU_INT32U   buf_len,
                     CPU_INT32U   xfer_len,
                     void        *p_arg,
                     USBD_ERR     err);
```

Argument(s)

`dev_nbr`

Device number.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to buffer of data that will be transmitted.

`buf_len`

Buffer length.

`xfer_len`

Number of byte transmitted.

`p_arg`

Pointer to function argument.

`err`

Error status.

```
USBD_ERR_NONE
USBD_ERR_EP_ABORT
```

USBD_EP_RxZLP

Description

Receives zero-length packet from the host.

Files

usbd_core.h/usbd_ep.c

Prototype

```
void  USBD_EP_RxZLP (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     CPU_INT16U   timeout_ms,
                     USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_OS_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT

```
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

None.

USBD_EP_TxZLP

Description

Transmits zero-length packet from the host.

Files

usbd_core.h/usbd_ep.c

Prototype

```
void  USBD_EP_TxZLP (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     CPU_INT16U   timeout_ms,
                     USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_OS_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT

USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Classes.

Notes / Warnings

None.

USBD_EP_Abort

Description

Abort I/O transfer on endpoint.

Files

usbd_core.h/usbd_ep.c

Prototype

```
void  USBD_EP_Abort (CPU_INT08U   dev_nbr,
                     CPU_INT08U   ep_addr,
                     USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_ABORT
USBD_ERR_EP_OS_FAIL

Returned Value

None.

Callers

Classes.

Notes / Warnings

None.

USBD_EP_Stall

Description

Modify stall state condition on non-control endpoints.

Files

usbd_core.h/usbd_ep.c

Prototype

```
void  USBD_EP_Stall (CPU_INT08U    dev_nbr,
                     CPU_INT08U    ep_addr,
                     CPU_BOOLEAN   state,
                     USBD_ERR     *p_err)
```

Arguments

dev_nbr

Device number.

ep_addr

Line control change notification callback (see note #1).

state

Endpoint stall state.

Value	Stall state
DEF_SET	Set
DEF_CLR	Clear

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_ARG
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_STALL
USBD_ERR_EP_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

Classes.

Notes / Warnings

None.

USBD_EP_IsStalled

Description

Gets stall status of non-control endpoint

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_BOOLEAN  USBD_EP_IsStalled (CPU_INT08U   dev_nbr,
                                CPU_INT08U   ep_addr,
                                USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Pointer to the structure where the current line coding will be stored.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_DEV_INVALID_ARG
USBD_ERR_EP_INVALID_ADDR

Returned Value

DEF_TRUE, if endpoint is stalled.

DEF_FALSE, otherwise.

Callers

Classes,

Application.

Notes / Warnings

None.

USBD_EP_MaxPktSizeGet

Description

Retrieves endpoint’s maximum packet size

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT16U  USBD_EP_MaxPktSizeGet (CPU_INT08U   dev_nbr,
                                   CPU_INT08U   ep_addr,
                                   USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_addr

Endpoint address.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
```

Returned Value

Maximum packet size, If no error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_EP_MaxPhyNbrGet

Description

Get the maximum physical endpoint number.

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT08U  USBD_EP_MaxPhyNbrGet (CPU_INT08U  dev_nbr)
```

Arguments

dev_nbr

Device number.

Returned Value

Maximum physical endpoint number, If no error(s).

USBD_EP_PHY_NONE, otherwise.

Callers

USB device controllers drivers.

Application.

Notes / Warnings

None.

USBD_EP_MaxNbrOpenGet

Description

Retrieve maximum number of opened endpoints

Files

usbd_core.h/usbd_ep.c

Prototype

```
CPU_INT08U  USBD_EP_MaxNbrOpenGet (CPU_INT08U  dev_nbr);
```

Arguments

dev_nbr

Device number.

Returned Value

Maximum number of opened endpoints, If no errors.

0, otherwise.

Callers

Classes,

Application.

Notes / Warnings

None.

Trace Functions

- USBD_Trace

USBD_Trace

Description

Outputs debug information from the core. Users must implement this function if trace functionality is enabled (`USBD_CFG_DBG_TRACE` is defined to `DEF_ENABLED`).

Files

`usbd_core.h`

Prototype

```
void  USBD_Trace (const CPU_CHAR  *p_str);
```

Arguments

`p_drv`

Pointer to the string containing debug information.

Returned Value

None.

Callers

USB core debug task handler.

Notes / Warnings

None.

Application Callback Functions

- App_USBD_EventReset
- App_USBD_EventSuspend
- App_USBD_EventResume
- App_USBD_EventCfgSet
- App_USBD_EventCfgClr
- App_USBD_EventConn
- App_USBD_EventDisconn

App_USBD_EventReset

Description

Bus reset event callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventReset (CPU_INT08U  dev_nbr);
```

Arguments

`dev_nbr`

Device number.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventSuspend

Description

Bus suspend event callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventSuspend (CPU_INT08U  dev_nbr);
```

Arguments

`dev_nbr`

Device number.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventResume

Description

Bus resume event callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventResume (CPU_INT08U  dev_nbr);
```

Arguments

`dev_nbr`

Device number.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventCfgSet

Description

Set configuration callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventCfgSet (CPU_INT08U  dev_nbr,
                                    CPU_INT08U  cfg_val);
```

Arguments

`dev_nbr`

Device number.

`cfg_val`

Active device configuration number selected by the host.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventCfgClr

Description

Clear configuration callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventCfgClr (CPU_INT08U  dev_nbr,
                                    CPU_INT08U  cfg_val);
```

Arguments

`dev_nbr`

Device number.

`cfg_val`

Current device configuration number that will be closed.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventConn

Description

Device connection event callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventConn (CPU_INT08U  dev_nbr);
```

Arguments

`dev_nbr`

Device number.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

App_USBD_EventDisconn

Description

Device disconnection event callback function.

Files

Application (template in `app_usbd.c`)

Prototype

```
static  void  App_USBD_EventDisconn (CPU_INT08U  dev_nbr);
```

Arguments

`dev_nbr`

Device number.

Returned Value

None.

Callers

Core.

Notes / Warnings

This function is registered to the core by having a pointer to it in the `USBD_BUS_FNCTS` structure passed as a parameter when calling `USBD_DevAdd`.

Device Driver Callback Functions

- USBD_EP_RxCmpl
- USBD_EP_TxCmpl
- USBD_EventConn
- USBD_EventDisconn
- USBD_EventHS
- USBD_EventReset
- USBD_EventResume
- USBD_EventSetup
- USBD_EventSuspend

USBD_EP_RxCmpl

Description

Notifies the stack that an OUT transfer is completed.

Files

`usbd_core.h/usbd_ep.c`

Prototype

```
void  USBD_EP_RxCmpl (USBD_DRV    *p_drv,
                      CPU_INT08U   ep_log_nbr);
```

Arguments

`p_drv`

Pointer to device driver structure.

`ep_log_nbr`

Endpoint logical number.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EP_TxCmpl

Description

Notifies the stack that an IN transfer is completed.

Files

usbd_core.h/usbd_ep.c

Prototype

```
void  USBD_EP_TxCmpl (USBD_DRV    *p_drv,
                      CPU_INT08U   ep_log_nbr);
```

Arguments

p_drv

Pointer to device driver structure.

ep_log_nbr

Endpoint logical number.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventConn

Description

Notifies the stack the device is connected to the host.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_EventConn (USBD_DRV  *p_drv);
```

Arguments

p_drv

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventDisconn

Description

Notifies the stack the device is disconnected from the host.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_EventDisconn (USBD_DRV  *p_drv);
```

Arguments

p_drv

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventHS

Description

This function notifies the stack that a host is high speed capable.

Files

`usbd_core.h/usbd_core.c`

Prototype

```
void  USBD_EventHS (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventReset

Description

Notifies the stack a reset event in the bus.

Files

`usbd_core.h/usbd_core.c`

Prototype

```
void  USBD_EventReset (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventResume

Description

Notifies the stack a resume event in the bus.

Files

`usbd_core.h/usbd_core.c`

Prototype

```
void  USBD_EventResume (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventSetup

Description

Notifies the stack that a setup transfer has been received.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_EventSetup (USBD_DRV  *p_drv,
                       void      *p_buf);
```

Arguments

p_drv

Pointer to device driver structure.

p_buf

Pointer to buffer that contains the setup packet.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

USBD_EventSuspend

Description

Notifies the stack a suspend event in the bus.

Files

usbd_core.h/usbd_core.c

Prototype

```
void  USBD_EventSuspend (USBD_DRV  *p_drv);
```

Arguments

p_drv

Pointer to device driver structure.

Returned Value

None.

Callers

USB device controller driver ISR

Notes / Warnings

None.

Core OS Functions

- USBD_OS_Init
- USBD_CoreTaskHandler
- USBD_DbgTaskHandler
- USBD_OS_EP_SignalCreate
- USBD_OS_EP_SignalDel
- USBD_OS_EP_SignalPend
- USBD_OS_EP_SignalAbort
- USBD_OS_EP_SignalPost
- USBD_OS_EP_LockCreate
- USBD_OS_EP_LockDel
- USBD_OS_EP_LockAcquire
- USBD_OS_EP_LockRelease
- USBD_OS_DlyMs
- USBD_OS_CoreEventGet
- USBD_OS_CoreEventPut
- USBD_OS_DbgEventRdy
- USBD_OS_DbgEventWait

USBD_OS_Init

Description

Initialize USB RTOS layer internal objects.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Implementation guidelines

1. The followings RTOS resources are required by the stack and should be allocated in when this function is called.

   a. One task for core and asynchronous events.

   b. One queue that can hold up to `USBD_CORE_EVENT_NBR_TOTAL` events.

   c. `USBD_CFG_MAX_NBR_DEV` x `USBD_CFG_MAX_NBR_EP_OPEN` semaphores and mutex for endpoints operations.

   d. If tracing is enabled, a semaphore and a task to manage debug events allocation and debug events processing respectively.

2. If any error happen, `USBD_ERR_OS_INIT_FAIL` should be assigned to `p_err` and the function should return immediately. Otherwise, `USBD_ERR_NONE` should be assigned to `p_err`.

USBD_CoreTaskHandler

Description

Process all core events and operations.

Files

usbd_internal.h/usbd_core.c

Prototype

```
void  USBD_CoreTaskHandler (void);
```

Arguments

None.

Returned Value

None.

Callers

USB RTOS layer.

Implementation guidelines

1. Typically, the RTOS layer should create a shell task for core events. The primary purpose of the shell task is to run `USBD_CoreTaskHandler()`.

USBD_DbgTaskHandler

Description

Process all pending debug events generated by the core.

Files

`usbd_internal.h/usbd_core.c`

Prototype

```
void  USBD_DbgTaskHandler (void);
```

Arguments

None.

Returned Value

None.

Callers

USB RTOS layer.

Implementation guidelines

1. Typically, the RTOS layer code should create a shell task to process debug events generated by the core. The primary purpose of the shell task is to run `USBD_DbgTaskHandler()`.

2. This function is only present in the code if trace option is enabled in the stack.

USBD_OS_EP_SignalCreate

Description

Creates a signal/semaphore for endpoints operations.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_SignalCreate (CPU_INT08U   dev_nbr,
                               CPU_INT08U   ep_ix,
                               USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Endpoints open functions.

Implementation guidelines

1. The purpose of this function is to allocate a signal or a semaphore for the specified endpoint.

2. Typically, the RTOS layer code should create a two-dimensional array to store the signals/semaphores handlers. The `dev_nbr` and `ep_ix` are used to index this array.

3. `dev_nbr` ranges between 0 and `USBD_CFG_MAX_NBR_DEV`.

4. `ep_ix` ranges between 0 and `USBD_CFG_MAX_NBR_EP_OPEN`.

5. In case the creation fails, `USBD_ERR_OS_SIGNAL_CREATE` should be assigned to `p_err`. Otherwise, `USBD_ERR_NONE` should be assigned to `p_err`.

USBD_OS_EP_SignalDel

Description

Deletes a signal/semaphore.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_SignalDel (CPU_INT08U  dev_nbr,
                            CPU_INT08U  ep_ix);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

Returned Value

None.

Callers

Endpoints close functions.

Implementation guidelines

1. A call to this function should delete the signal / semaphore associated to the specified endpoint.

USBD_OS_EP_SignalPend

Description

Waits for a signal/semaphore to become available.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_SignalPend (CPU_INT08U   dev_nbr,
                             CPU_INT08U   ep_ix,
                             CPU_INT16U   timeout_ms,
                             USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Endpoints Rx/Tx functions.

Implementation guidelines

1. A call to this function should pend on the signal / semaphore associated to the specified endpoint.

2. Following table describes the error codes that should be assigned to `p_err` depending on the operation result.

Operation result	Error code
No error.	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error code for pend operations

USBD_OS_EP_SignalAbort

Description

Aborts any wait operation on signal/semaphore.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_SignalAbort (CPU_INT08U   dev_nbr,
                              CPU_INT08U   ep_ix,
                              USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Endpoints abort functions.

Implementation guidelines

1. This function should abort all pend operations performed on the signal / semaphore associated to the specified endpoint.

2. If any error happen, `USBD_ERR_OS_FAIL` should be assigned to `p_err`. Otherwise, `USBD_ERR_NONE` should be assigned to `p_err`.

USBD_OS_EP_SignalPost

Description

Makes a signal/semaphore available.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void   USBD_OS_EP_SignalPost  (CPU_INT08U   dev_nbr,
                               CPU_INT08U   ep_ix,
                               USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Endpoints transfer complete functions.

Implementation guidelines

1. A call to this function should post the signal / semaphore associated to the specified endpoint.

2. In case the post fails, `USBD_ERR_OS_FAIL` should be assigned to `p_err`. Otherwise, `USBD_ERR_NONE` should be assigned to `p_err`.

USBD_OS_EP_LockCreate

Description

Create an OS resource to use as an endpoint lock.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_LockCreate (CPU_INT08U   dev_nbr,
                             CPU_INT08U   ep_ix,
                             USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Endpoints open functions.

Implementation guidelines

1. The purpose of this function is to allocate a lock (mutex or semaphore) for the specified endpoint.

2. Typically, the RTOS layer code should create a two-dimensional array to store the lock handlers. The `dev_nbr` and `ep_ix` are used to index this array.

3. `dev_nbr` ranges between 0 and `USBD_CFG_MAX_NBR_DEV`.

4. `ep_ix` ranges between 0 and `USBD_CFG_MAX_NBR_EP_OPEN`.

5. In case the creation fails, `USBD_ERR_OS_SIGNAL_CREATE` should be assigned to `p_err`. Otherwise, `USBD_ERR_NONE` should be assigned to `p_err`.

USBD_OS_EP_LockDel

Description

Delete the OS resource used as an endpoint lock.

Files

`usbd_internal.h/usbd_os.c`

Prototype

```
void  USBD_OS_EP_LockDel (CPU_INT08U  dev_nbr,
                          CPU_INT08U  ep_ix);
```

Arguments

`dev_nbr`

Device number.

`ep_ix`

Endpoint index.

Returned Value

None.

Callers

Endpoints close functions.

Implementation guidelines

1. A call to this function should delete the lock associated to the specified endpoint.

USBD_OS_EP_LockAcquire

Description

Wait for an endpoint to become available and acquire its lock.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_EP_LockAcquire (CPU_INT08U   dev_nbr,
                              CPU_INT08U   ep_ix,
                              CPU_INT16U   timeout_ms,
                              USBD_ERR    *p_err);
```

Arguments

dev_nbr

Device number.

ep_ix

Endpoint index.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Implementation guidelines

1. A call to this function should pend on the lock associated to the specified endpoint.

2. Following table describes the error codes that should be assigned to `p_err` depending on the operation result.

Operation result	Error code
No error.	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error code for pend operations

USBD_OS_EP_LockRelease

Description

Release an endpoint lock.

Files

`usbd_internal.h/usbd_os.c`

Prototype

```
void  USBD_OS_EP_LockRelease (CPU_INT08U  dev_nbr,
                              CPU_INT08U  ep_ix);
```

Arguments

`dev_nbr`

Device number.

`ep_ix`

Endpoint index.

Returned Value

None.

Implementation guidelines

1. A call to this function should post the lock associated to the specified endpoint.

USBD_OS_DlyMs

Description

Delay a task for a certain time.

Files

`usbd_internal.h/usbd_os.c`

Prototype

```
void  USBD_OS_DlyMs (CPU_INT32U  ms);
```

Arguments

`ms`

Delay in milliseconds.

Returned Value

None.

Implementation guidelines

1. The RTOS layer code should delay the calling task by the specified number of milliseconds.

USBD_OS_CoreEventGet

Description

Wait until a core event is ready.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  *USBD_OS_CoreEventGet (CPU_INT32U   timeout_ms,
                             USBD_ERR    *p_err);
```

Arguments

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

Pointer to core event, if no errors.

Null pointer, otherwise.

Callers

USBD_CoreTaskHandler.

Implementation guidelines

1. A call to this function should block until an event is added to queue and return it.

2. Table - Error code for pend operations in the *USBD_OS_EP_SignalPend* page describes the error codes that should be assigned to `p_err` depending on the operation result.

USBD_OS_CoreEventPut

Description

Queues a core event.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_CoreEventPut (void  *p_event);
```

Arguments

p_event

Pointer to core event.

Returned Value

None.

Callers

Endpoints and bus event handlers.

Implementation guidelines

1. A call to this function should add the passed event to the core events queue.

USBD_OS_DbgEventRdy

Description

Signals debug event handler task.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_DbgEventRdy (void);
```

Arguments

None.

Returned Value

None.

Callers

Debug functions.

Implementation guidelines

1. A call to this function should post the signal / semaphore that resume the debug task.

USBD_OS_DbgEventWait

Description

Signals debug event handler task.

Files

usbd_internal.h/usbd_os.c

Prototype

```
void  USBD_OS_DbgEventWait (void);
```

Arguments

None.

Returned Value

None.

Callers

USBD_DbgTaskHandler

Implementation guidelines

1. A call to this function should pend on the signal / semaphore that resume the debug task.

API - Device Controller Driver

Device Driver Functions

- USBD_DrvInit
- USBD_DrvStart
- USBD_DrvStop
- USBD_DrvAddrSet
- USBD_DrvAddrEn
- USBD_DrvCfgSet
- USBD_DrvCfgClr
- USBD_DrvFrameNbrGet
- USBD_DrvEP_Open
- USBD_DrvEP_Close
- USBD_DrvEP_RxStart
- USBD_DrvEP_Rx
- USBD_DrvEP_RxZLP
- USBD_DrvEP_Tx
- USBD_DrvEP_TxStart
- USBD_DrvEP_TxZLP
- USBD_DrvEP_Abort
- USBD_DrvEP_Stall

- USBD_DrvISR_Handler

USBD_DrvInit

Description

Driver initialization/`Init()` function. This function is called by `USBD_DevStart()` exactly once for each specific device added by the application. If multiple instances of the same device are present on the development board, then this function is called for each instance of the device. However, applications should not try to add the same specific device more than once. If a device fails to initialize, it is recommend debugging to find and correct the cause of failure.

Note: This function relies heavily on the implementation of several device board support package (BSP) functions. See Device Driver BSP Functions for more information on device BSP functions.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvInit (USBD_DRV  *p_drv
                            USBD_ERR  *p_err);
```

Note that since every device driver function is accessed only by function pointer via the device driver's API structure, they do not need to be globally available and should therefore be declared as '`static`'.

Arguments

`p_drv`

Pointer to USB device driver structure.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. The `Init()` function generally performs the following operations, however, depending on the device being initialized, functionality may need to be added or removed:

    a. Configure clock gating to the USB device, configure all necessary I/O pins, and configure the host interrupt controller. This is generally performed via the device's BSP function pointer, `Init()`, implemented in `usbd_bsp_<driver_name>.c` (see `USBD_BSP_Init`).

    b. Reset USB controller or USB controller registers.

    c. Disable and clear pending interrupts (should already be cleared).

    d. Set the device address to zero.

    e. For DMA devices: Allocate memory for all necessary descriptors. This is performed via calls to µC/LIB's memory module. If memory allocation fails, set `p_err` to `USBD_ERR_ALLOC` and return.

    f. Set `p_err` to `USBD_ERR_NONE` if initialization proceeded as expected. Otherwise, set `p_err` to an appropriate device error code.

USBD_DrvStart

Description

Device driver `Start()` function. This function is called once each time a device is started.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvStart (USBD_DRV  *p_drv
                             USBD_ERR  *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. The `Start()` function performs the following operations:

    a. Typically, activates the pull-up on the D+ pin to simulate attachment to host. Some MCUs/MPUs have an internal pull-up that is activated by a device controller register; for others, this may be a general purpose I/O pin. This is generally performed via the device's BSP function pointer, `Conn()`, implemented in `usbd_bsp_<driver_name>.c` (see `USBD_BSP_Conn`). The device's `BSP Conn()` is also responsible for enabling the host interrupt controller.

    b. Clear all interrupt flags.

    c. Locally enable interrupts on the hardware device. The host interrupt controller should have already been configured within the device driver `Init()` function.

    d. Enable the controller.

    e. Set `p_err` equal to `USBD_ERR_NONE` if no errors have occurred. Otherwise, set `p_err` to an appropriate device error code.

USBD_DrvStop

Description

The next function within the device API structure is the device `Stop()` function. This function is called once each time a device is stopped.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvStop (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the `Stop()` function performs the following operations:

    a. Disable the controller.

    b. Clear and locally disable interrupts on the hardware device.

    c. Disconnect from the USB host (e.g, reset the pull-up on the D+ pin). This is generally performed via the device's BSP function pointer, `Disconn()`, implemented in `usbd_bsp_<driver_name>.c` (see `USBD_BSP_Disconn`).

USBD_DrvAddrSet

Description

The next API function to implement is the device address set/`AddrSet()` function. The device address set function is called while processing a `SET_ADDRESS` setup request.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_BOOLEAN  USBD_DrvAddrSet (USBD_DRV    *p_drv,
                                      CPU_INT08U   dev_addr);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`dev_addr`

Device address assigned by the host.

Returned Value

`DEF_OK`, if NO error(s).

`DEF_FAIL`, otherwise.

Callers

USB device core layer.

Notes / Warnings

1. For device controllers that have hardware assistance to enable the device address after the status stage has completed, the assignment of the device address can also be combined with enabling the device address mode.

2. For device controllers that change the device address immediately, without waiting the status phase to complete, see `USBD_DrvAddrEn`.

USBD_DrvAddrEn

Description

The next function in the device API structure is the device address enable/`AddrEn()` function.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_BOOLEAN  USBD_DrvAddrEn (USBD_DRV    *p_drv
                                     CPU_INT08U   dev_addr);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`dev_addr`

Device address assigned by the host.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. For device controllers that have hardware assistance to enable the device address after the status stage has completed, no operation needs to be performed.

2. For device controllers that change the device address immediately, without waiting the status phase to complete, the device address must be set and enabled.

USBD_DrvCfgSet

Description

Bring device into configured state.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_BOOLEAN  USBD_DrvCfgSet (USBD_DRV    *p_drv,
                                     CPU_INT08U   cfg_val);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`cfg_val`

Configuration value.

Returned Value

`DEF_OK`,

if NO error(s).

`DEF_FAIL`,

otherwise.

Callers

USB device core layer.

Notes / Warnings

Typically, the set configuration function sets the device as configured. For some controllers, this may not be necessary.

USBD_DrvCfgClr

Description

Bring device into de-configured state.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvCfgClr (USBD_DRV    *p_drv,
                              CPU_INT08U   cfg_val);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`cfg_val`

Configuration value.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the clear configuration function sets the device as not being configured. For some controllers, this may not be necessary.

2. This function in invoked after a bus reset or before the status stage of some `SET_CONFIGURATION` requests.

USBD_DrvFrameNbrGet

Description

Retrieve current frame number.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_INT16U  USBD_DrvFrameNbrGet (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

Returned Value

Frame number.

Callers

USB device core layer.

Notes / Warnings

1. The value returned by this function is a 16-bit value in which the current frame number and microframe number (if the controller is high-speed) should be encoded. The frame number should be encoded using 11 bits (bit 0 to 10). A microframe number will be encoded using 3 bits (bit 11 to 13).

USBD_DrvEP_Open

Description

Open and configure a device endpoint, given its characteristics (e.g., endpoint type, endpoint address, maximum packet size, etc).

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvEP_Open (USBD_DEV    *p_drv,
                               CPU_INT08U   ep_addr,
                               CPU_INT08U   ep_type,
                               CPU_INT16U   max_pkt_size,
                               CPU_INT08U   transaction_frame,
                               USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`ep_type`

Endpoint type:

```
USB_EP_TYPE_CTRL,
USB_EP_TYPE_ISOC,
USB_EP_TYPE_BULK,
USB_EP_TYPE_INTR.
```

`max_pkt_size`

Maximum packet size.

`transaction_frame`

Endpoint transactions per frame.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the endpoint open function performs the following operations:

    a. Configure endpoint information in the device controller. This may include not only assigning the type and maximum packet size, but also making certain that the endpoint is successfully configured (or *realized* or *mapped*). For some device controllers, this may not be necessary.

    b. `max_pkt_size` is the maximum packet size the endpoint can send or receive.

USBD_DrvEP_Close

Description

Close a device endpoint, and un-initialize/clear endpoint configuration in hardware.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvEP_Close (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the endpoint close function clears the endpoint information in the device controller. For some controllers, this may not be necessary.

USBD_DrvEP_RxStart

Description

Configure endpoint with buffer to receive data.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_INT32U  USBD_DrvEP_RxStart (USBD_DRV    *p_drv,
                                        CPU_INT08U   ep_addr,
                                  CPU_INT08U  *p_buf,
                                  CPU_INT32U   buf_len,
                                  USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to data buffer.

`buf_len`

Length of the buffer.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

Maximum number of octets that will be received, if NO error(s).

0, otherwise.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the function to configure the endpoint receive transaction performs the following operations:

    a. Determine maximum transaction length, given the specified length of the buffer (`buf_len`).

    b. Setup receive transaction.

USBD_DrvEP_Rx

Description

Receive the specified amount of data from device endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_INT32U  USBD_DrvEP_Rx (USBD_DRV    *p_drv,
                                   CPU_INT08U   ep_addr,
                                   CPU_INT08U  *p_buf,
                                   CPU_INT32U   buf_len,
                                   USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to data buffer.

`buf_len`

Length of the buffer.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

Number of octets received, if NO error(s)

0, otherwise

Callers

USB device core layer.

Notes / Warnings

1. Typically, the receive from endpoint function performs the following operations:

    a. Check if packet has been received and is ready to be read.

    b. Determine packet length.

    c. Copy the data received into the buffer referenced by `p_buf`. If the USB device controller is working in DMA mode, no CPU copy will be required as it was already done by the DMA engine of the USB device controller.

    d. If an error occurred during the transfer (i.e. overflow or buffer error), the function must set the value of `p_err` to `USBD_ERR_RX` and treat the transfer data as invalid.

    e. Clear endpoint buffer to allow next packet to be received. For some controllers, this may not be necessary.

USBD_DrvEP_RxZLP

Description

Receive zero-length packet from endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvEP_RxZLP (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr,
                                USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

None.

USBD_DrvEP_Tx

Description

Configure endpoint with buffer to transmit data.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_INT32U  USBD_DrvEP_Tx (USBD_DRV    *p_drv,
                                   CPU_INT08U   ep_addr,
                                   CPU_INT08U  *p_buf,
                                   CPU_INT32U   buf_len,
                                   USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to data buffer.

`buf_len`

Length of the buffer.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

Number of octets transmitted, if NO error(s).

0, otherwise.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the function to configure the endpoint receive transaction performs the following operations:

    a. Check if data can be transmitted.

    b. Write data to device endpoint.

    c. Configure the packet length in USB device controller. This is often necessary when the packet is shorter than the maximum packet size. Depending on the USB controller, this operation may need to be performed prior to writing the data to the device endpoint.

USBD_DrvEP_TxStart

Description

Transmit the specified amount of data to device endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvEP_TxStart (USBD_DRV    *p_drv,
                                  CPU_INT08U   ep_addr,
                                  CPU_INT08U  *p_buf,
                                  CPU_INT32U   buf_len,
                                  USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_buf`

Pointer to data buffer.

`buf_len`

Length of the buffer.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

Number of octets transmitted, if NO error(s).

0, otherwise.

Callers

USB device core layer.

Notes / Warnings

1. Typically, the function to configure the endpoint transmit transaction performs the following operations:

    a. Trigger packet transmission.

USBD_DrvEP_TxZLP

Description

Transmit zero-length packet to endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvEP_TxZLP (USBD_DRV    *p_drv,
                                CPU_INT08U   ep_addr,
                                USBD_ERR    *p_err);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

USB device core layer.

Notes / Warnings

None.

USBD_DrvEP_Abort

Description

Abort any pending transfer on endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_BOOLEAN  USBD_DrvEP_Abort (USBD_DRV    *p_drv,
                                       CPU_INT08U   ep_addr);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint Address.

Returned Value

`DEF_OK`, if NO error(s).

`DEF_FAIL`, otherwise.

Callers

USB device core layer.

Notes / Warnings

None.

USBD_DrvEP_Stall

Description

Set or clear stall condition on endpoint.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  CPU_BOOLEAN  USBD_DrvEP_Stall (USBD_DRV     *p_drv,
                                       CPU_INT08U    ep_addr,
                                       CPU_BOOLEAN   state);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

`ep_addr`

Endpoint address.

`state`

Endpoint stall state.

Returned Value

`DEF_OK`, if NO error(s).

`DEF_FAIL`, otherwise.

Callers

USB device core layer.

Notes / Warnings

None.

USBD_DrvISR_Handler

Description

USB device Interrupt Service Routine (ISR) handler.

Files

Every device driver's `usbd_drv.c`

Prototype

```
static  void  USBD_DrvISR_Handler (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

Returned Value

None.

Callers

Processor level kernel-aware interrupt handler.

Notes / Warnings

None.

Device Driver BSP Functions

- USBD_BSP_Init
- USBD_BSP_Conn
- USBD_BSP_Disconn

USBD_BSP_Init

Description

Initialize board-specific USB controller dependencies.

Files

Every device driver's `usbd_bsp_<driver_name>.c`

Prototype

```
static  void  USBD_BSP_Init (USBD_DRV  *p_drv);
```

Arguments

`p_drv`

Pointer to USB device driver structure.

Returned Value

None.

Callers

USB device driver.

Notes / Warnings

None.

USBD_BSP_Conn

Description

Enable USB controller connection dependencies.

Files

Every device driver's `usbd_bsp_<driver_name>.c`

Prototype

```
static  void  USBD_BSP_Conn (void);
```

Arguments

None.

Returned Value

None.

Callers

USB device driver.

Notes / Warnings

None.

USBD_BSP_Disconn

Description

Disable USB controller connection dependencies.

Files

Every device driver's `usbd_bsp_<driver_name>.c`

Prototype

```
static  void  USBD_BSP_Disconn (void);
```

Arguments

None.

Returned Value

None.

Callers

USB device driver.

Notes / Warnings

None.

API - Audio Class

- USBD_Audio_OS_DlyMs
- USBD_Audio_OS_RecordTask
- USBD_Audio_OS_PlaybackTask

- Audio Peripheral Driver Functions
  - USBD_Audio_DrvInit
  - USBD_Audio_DrvCtrlOT_CopyProtSet
  - USBD_Audio_DrvCtrlFU_MuteManage
  - USBD_Audio_DrvCtrlFU_VolManage
  - USBD_Audio_DrvCtrlFU_BassManage
  - USBD_Audio_DrvCtrlFU_MidManage
  - USBD_Audio_DrvCtrlFU_TrebleManage
  - USBD_Audio_DrvCtrlFU_GraphicEqualizerManage
  - USBD_Audio_DrvCtrlFU_AutoGainManage
  - USBD_Audio_DrvCtrlFU_DlyManage
  - USBD_Audio_DrvCtrlFU_BassBoostManage
  - USBD_Audio_DrvCtrlFU_LoudnessManage
  - USBD_Audio_DrvCtrlMU_CtrlManage
  - USBD_Audio_DrvCtrlSU_InPinManage
  - USBD_Audio_DrvAS_SamplingFreqManage

- USBD_Audio_DrvAS_PitchManage
- USBD_Audio_DrvStreamStart
- USBD_Audio_DrvStreamStop
- USBD_Audio_DrvStreamRecordRx
- USBD_Audio_DrvStreamPlaybackTx

Audio Class Functions

- USBD_Audio_Init
- USBD_Audio_Add
- USBD_Audio_CfgAdd
- USBD_Audio_IsConn
- USBD_Audio_IT_Add
- USBD_Audio_OT_Add
- USBD_Audio_FU_Add
- USBD_Audio_MU_Add
- USBD_Audio_SU_Add
- USBD_Audio_IT_Assoc
- USBD_Audio_OT_Assoc
- USBD_Audio_FU_Assoc
- USBD_Audio_MU_Assoc
- USBD_Audio_SU_Assoc
- USBD_Audio_MU_MixingCtrlSet
- USBD_Audio_AS_IF_Cfg
- USBD_Audio_AS_IF_Add
- USBD_Audio_AS_IF_StatGet

- USBD_Audio_RecordBufGet
- USBD_Audio_RecordRxCmpl
- USBD_Audio_PlaybackTxCmpl
- USBD_Audio_PlaybackBufFree
- Deprecated Functions
  - USBD_Audio_RecordBufFree

USBD_Audio_Init

Description

Initialize internal structures, local global variables, Audio Processing module and OS layer used by the audio class.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_Init (CPU_INT16U   msg_qty
                       USBD_ERR    *p_err);
```

Arguments

msg_qty

Maximum quantity of messages for playback and record tasks' queues.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_OS_INIT_FAIL

Returned Value

None.

Callers

Application.

Notes / Warnings

1. The initialization function *must* be called only once by the application, and before calling any other audio class API.

USBD_Audio_Add

Description

Create a new instance of the audio class.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_Add (USBD_AUDIO_DRV_API  *p_audio_drv_api,
                            USBD_ERR            *p_err);
```

Arguments

p_audio_drv_api

Pointer to Audio Peripheral Driver API.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_ALLOC

Returned Value

Class instance number, if NO error(s).

USBD_CLASS_NBR_NONE, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Audio_CfgAdd

Description

Add an audio class instance into the specified configuration. The audio class instance was previously created by the function `USBD_Audio_Add`.

Files

`usbd_audio.h / usbd_audio.c`

Prototype

```
void  USBD_Audio_CfgAdd (CPU_INT08U   class_nbr,
                         CPU_INT08U   dev_nbr,
                         CPU_INT08U   cfg_nbr,
                         USBD_ERR    *p_err);
```

Arguments

`class_nbr`

Class instance number.

`dev_nbr`

Device number.

`cfg_nbr`

Configuration index to add audio class instance to.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
USBD_ERR_NULL_PTR
```

```
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. If `USBD_Audio_CfgAdd()` is called several times from the application, it allows to create multiple instances and multiple configurations. For instance, the following architecture could be created:

```
Audio Class 0 (device number 0)
    |-- Configuration 0
    |-- Configuration 1
Audio Class 1 (device number 1)
    |-- Configuration 0
    |-- Configuration 1
```

2. The Configuration Descriptor corresponding to an audio device has the following format:

```
Configuration Descriptor
|-- Interface Descriptor (AudioControl)
[standard]
    |-- Header Descriptor
[class-specific]
    |-- Unit Descriptor(s)
[class-specific]
    |-- Terminal Descriptor(s)
[class-specific]
|-- Endpoint Descriptor (Interrupt IN) - optional
[standard]
    |-- Interface Descriptor (AudioStreaming)
[standard]
    |-- AS Interface Descriptor
[class-specific]
        |-- AS Format Type Descriptor
[class-specific]
    |-- AS Format-Specific Descriptor(s)
[class-specific]
    |-- Endpoint Descriptor (Isochronous IN or OUT Data)
[standard]
        |-- AS Isochronous Audio Data Endpoint Descriptor
[class-specific]
        |-- Endpoint Descriptor (Isochronous OUT or IN Synch endpoint)
[standard]
    |-- Interface Descriptor (AudioStreaming)
[standard]
        |-- ...
    |-- Interface Descriptor (AudioStreaming)
```

```
[standard]
        |-- ...
```

The function `USBD_Audio_CfgAdd()` only adds the AudioControl interface descriptor. Other standard and class-specific descriptors are added by functions:

`USBD_Audio_IT_Add`

`USBD_Audio_OT_Add`

`USBD_Audio_FU_Add`

`USBD_Audio_MU_Add`

`USBD_Audio_SU_Add`

`USBD_Audio_AS_IF_Add`

USBD_Audio_IsConn

Description

Get the audio class connection state.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_BOOLEAN  USBD_Audio_IsConn (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Returned Value

DEF_YES, if audio class is connected.

DEF_NO, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Audio_IT_Add

Description

Add an Input Terminal to the specified class instance (i.e. audio function).

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_IT_Add (       CPU_INT08U          class_nbr,
                               const  USBD_AUDIO_IT_CFG  *p_it_cfg,
                                      USBD_ERR           *p_err);
```

Arguments

class_nbr

Class instance number.

p_it_cfg

Pointer to the Input Terminal configuration structure.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_AUDIO_IT_ALLOC

Returned Value

Terminal ID assigned by audio class, if NO error(s).

0, otherwise

Callers

Application.

Notes / Warnings

1. Audio 1.0 specification indicates that ID #0 is reserved for undefined ID. Thus it indicates an error.

USBD_Audio_OT_Add

Description

Add an Output Terminal to the specified class instance (i.e. audio function).

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_OT_Add (       CPU_INT08U                 class_nbr,
                               const  USBD_AUDIO_OT_CFG         *p_ot_cfg,
                               const  USBD_AUDIO_DRV_AC_OT_API  *p_ot_api,
                                      USBD_ERR                  *p_err);
```

Arguments

class_nbr

Class instance number.

p_ot_cfg

Pointer to the Output Terminal configuration structure.

p_ot_api

Pointer to the audio codec API associated to this Output Terminal.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_AUDIO_OT_ALLOC

Returned Value

Terminal ID assigned by audio class, if NO error(s).

0, otherwise

Callers

Application.

Notes / Warnings

1. Audio 1.0 specification indicates that ID #0 is reserved for undefined ID. Thus it indicates an error.

2. `p_ot_api` can be `DEF_NULL` if the Output Terminal does not support the Copy Protection control.

USBD_Audio_FU_Add

Description

Add a Feature Unit to the specified class instance (i.e. audio function).

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_FU_Add (       CPU_INT08U                 class_nbr,
                               const  USBD_AUDIO_FU_CFG         *p_fu_cfg,
                               const  USBD_AUDIO_DRV_AC_FU_API  *p_fu_api,
                                      USBD_ERR                  *p_err);
```

Arguments

class_nbr

Class instance number.

p_fu_cfg

Pointer to the Feature Unit configuration structure.

p_fu_api

Pointer to the audio codec API associated to this Feature Unit.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_AUDIO_FU_ALLOC

Returned Value

Unit ID assigned by audio class, if NO error(s).

0, otherwise

Callers

Application.

Notes / Warnings

1. Audio 1.0 specification indicates that ID #0 is reserved for undefined ID. Thus it indicates an error.

2. `p_fu_api` can NOT be `DEF_NULL`. A Feature Unit must at least support the mute and volume controls.

USBD_Audio_MU_Add

Description

Add a Mixer Unit to the specified class instance (i.e. audio function).

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_MU_Add (       CPU_INT08U                 class_nbr,
                               const  USBD_AUDIO_MU_CFG         *p_mu_cfg,
                               const  USBD_AUDIO_DRV_AC_MU_API  *p_mu_api,
                                      USBD_ERR                  *p_err);
```

Arguments

class_nbr

Class instance number.

p_mu_cfg

Pointer to the Mixer Unit configuration structure.

p_mu_api

Pointer to the audio codec API associated to this Mixer Unit.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_AUDIO_MU_ALLOC
USBD_ERR_ALLOC

Returned Value

Unit ID assigned by audio class, if NO error(s).

0, otherwise

Callers

Application.

Notes / Warnings

1. This function is enabled by setting the configuration constant `USBD_AUDIO_CFG_MAX_NBR_MU` to a value different from zero.

2. Audio 1.0 specification indicates that ID #0 is reserved for undefined ID. Thus it indicates an error.

3. `p_mu_api` can be `DEF_NULL` if the Mixer Unit does NOT support any programmable controls.

USBD_Audio_SU_Add

Description

Add a Selector Unit to the specified class instance (i.e. audio function).

Files

usbd_audio.h / usbd_audio.c

Prototype

```
CPU_INT08U  USBD_Audio_SU_Add (       CPU_INT08U                 class_nbr,
                               const  USBD_AUDIO_SU_CFG         *p_su_cfg,
                               const  USBD_AUDIO_DRV_AC_SU_API  *p_su_api,
                                      USBD_ERR                  *p_err);
```

Arguments

class_nbr

Class instance number.

p_su_cfg

Pointer to the Selector Unit configuration structure.

p_su_api

Pointer to the audio codec API associated to this Selector Unit.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_AUDIO_SU_ALLOC

Returned Value

Unit ID assigned by audio class, if NO error(s).

0, otherwise

Callers

Application.

Notes / Warnings

1. This function is enabled by setting the configuration constant `USBD_AUDIO_CFG_MAX_NBR_SU` to a value different from zero.

2. Audio 1.0 specification indicates that ID #0 is reserved for undefined ID. Thus it indicates an error.

3. `p_su_api` can NOT be `DEF_NULL`.

USBD_Audio_IT_Assoc

Description

Associate the Input Terminal to an Output Terminal

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_IT_Assoc (CPU_INT08U   class_nbr,
                           CPU_INT08U   it_id,
                           CPU_INT08U   assoc_terminal_id,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

it_id

Input Terminal ID.

assoc_terminal_id

Output Terminal ID associated.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR

Returned Value

None.

Callers

Application.

Notes / Warnings

1. The Input Terminal descriptor has the field 'bAssocTerminal'. It associates an Output Terminal to this Input Terminal, effectively implementing a bi-directional Terminal pair.

2. If there is no associated Output Terminal, you can set the parameter `assoc_terminal_id` to `USBD_AUDIO_TERMINAL_NO_ASSOCIATION`

USBD_Audio_OT_Assoc

Description

Specify the entity ID connected to this Output Terminal and associate the Output Terminal to an Input Terminal.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_OT_Assoc (CPU_INT08U   class_nbr,
                           CPU_INT08U   ot_id,
                           CPU_INT08U   entity_id_src,
                           CPU_INT08U   assoc_terminal_id,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

ot_id

Output Terminal ID.

entity_id_src

ID of the Unit or Terminal to which this Terminal is connected.

assoc_terminal_id

Input Terminal ID associated.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. The Output Terminal descriptor has the field 'bAssocTerminal'. It associates an Input Terminal to this Output Terminal, effectively implementing a bi-directional Terminal pair.

2. If there is no associated Input Terminal, you can set the parameter `assoc_terminal_id` to `USBD_AUDIO_TERMINAL_NO_ASSOCIATION`

USBD_Audio_FU_Assoc

Description

Specify the entity ID connected to this Feature Unit.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_FU_Assoc (CPU_INT08U   class_nbr,
                           CPU_INT08U   fu_id,
                           CPU_INT08U   src_entity_id,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

fu_id

Feature Unit ID.

src_entity_id

ID of the Unit or Terminal to which this Unit is connected.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_INVALID_ARG

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_Audio_MU_Assoc

Description

Specify the entities ID connected to this Mixer Unit.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_MU_Assoc (CPU_INT08U   class_nbr,
                           CPU_INT08U   mu_id,
                           CPU_INT08U  *p_src_entity_id,
                           CPU_INT08U   nbr_input_pins,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

mu_id

Mixer Unit ID.

p_src_entity_id

Pointer to table containing IDs of Units and/or Terminals to which Input Pins of this Mixer Unit are connected.

nbr_input_pins

Number of input pins.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
```

Returned Value

None.

Callers

Application.

Notes / Warnings

The listing Listing - USBD_Audio_MU_Assoc() Example Usage in the *USBD_Audio_MU_Assoc* page presents an example of usage:

```
CPU_INT08U  MU11_SrcIdTbl[] = {                                 (1)
    0u,
    0u,
    0u
};

MU11_SrcIdTbl[0u] = FU7_ID;                                     (2)
MU11_SrcIdTbl[1u] = IT2_ID;
MU11_SrcIdTbl[2u] = FU8_ID;

USBD_Audio_MU_Assoc(audio_nbr,                                  (3)
                    MU11_ID,
                   &MU11_SrcIdTbl[0u],
                    USBD_MU11_Cfg.NbrInPins,
                   &err);
if (err != USBD_ERR_NONE) {
    /* $$$$ Handle the error. */
}
```

Listing - USBD_Audio_MU_Assoc() Example Usage

(1) Declare a table that will contain the ID of terminals and/or units connected to the input pins of the Mixer Unit. The table size correspond to the number of inputs pins of the Mixer Unit.

(2) Initialize the table with all the IDs obtained when calling the functions `USBD_Audio_XX_Add()`. In this example, two Input terminals and one Feature unit are

connected to the Mixer Unit.

(3) Pass the table address and the number of input pins of the Mixer Unit to the function.

USBD_Audio_SU_Assoc

Description

Specify the entities ID connected to this Selector Unit.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_SU_Assoc (CPU_INT08U   class_nbr,
                           CPU_INT08U   su_id,
                           CPU_INT08U  *p_src_entity_id,
                           CPU_INT08U   nbr_input_pins,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

su_id

Selector Unit ID.

p_src_entity_id

Pointer to table containing IDs of Units and/or Terminals to which Input Pins of this Selector Unit are connected.

nbr_input_pins

Number of input pins.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
```

Returned Value

None.

Callers

Application.

Notes / Warnings

The listing Listing - USBD_Audio_SU_Assoc() Example Usage in the *USBD_Audio_SU_Assoc* page presents an example of usage:

```
CPU_INT08U  SU12_SrcIdTbl[] = {                                 (1)
    0u,
    0u
};

SU12_SrcIdTbl[0u] = FU6_ID;                                     (2)
SU12_SrcIdTbl[1u] = FU9_ID;

USBD_Audio_SU_Assoc(audio_nbr,                                  (3)
                    SU12_ID,
                   &SU12_SrcIdTbl[0u],
                    USBD_SU12_Cfg.NbrInPins,
                   &err);
if (err != USBD_ERR_NONE) {
    /* $$$$ Handle the error. */
}
```

Listing - USBD_Audio_SU_Assoc() Example Usage

(1) Declare a table that will contain the ID of terminals and/or units connected to the input pins of the Selector Unit. The table size correspond to the number of inputs pins of the Selector Unit.

(2) Initialize the table with all the IDs obtained when calling the functions `USBD_Audio_XX_Add()`. In this example, two Feature units are connected to the Selector Unit.

(3) Pass the table address and the number of input pins of the Selector Unit to the function.

USBD_Audio_MU_MixingCtrlSet

Description

Set one programmable mixing control.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_MU_MixingCtrlSet (CPU_INT08U   class_nbr,
                                   CPU_INT08U   mu_id,
                                   CPU_INT08U   log_in_ch_nbr,
                                   CPU_INT08U   log_out_ch_nbr,
                                   USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

mu_id

Mixer Unit ID.

log_in_ch_nbr

Logical input channel number.

log_out_ch_nbr

Logical output channel number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

```
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
```

Returned Value

None.

Callers

Application.

Notes / Warnings

The Mixer Unit descriptor has a field called `bmControls`. The field `bmControls` is interpreted by the host as a two-dimensional bit array representing a bitmap of which Mixer Unit controls are programmable or not. Called several times. the function `USBD_Audio_MU_MixingCtrlSet()` allows to set the programmable mixing controls within the bitmap. There are two different situations:

1. Mixer Unit has only non-programmable controls.

2. Mixer Unit has non-programmable and programmable controls.

In the situation #1, the function `USBD_Audio_MU_MixingCtrlSet()` is not necessary. In the situation #2, the function is required for each programmable mixing control. To fully understand how to use the function, the way the Mixer Unit Descriptor reports its programmable controls in `bmControls` must be understood.

A Mixer Unit with 2 stereo Input pins and 1 stereo Output pin is considered. In Figure - Mixer Unit Example in the *USBD_Audio_MU_MixingCtrlSet* page, u refers to logical input channels and v to logical output channel.

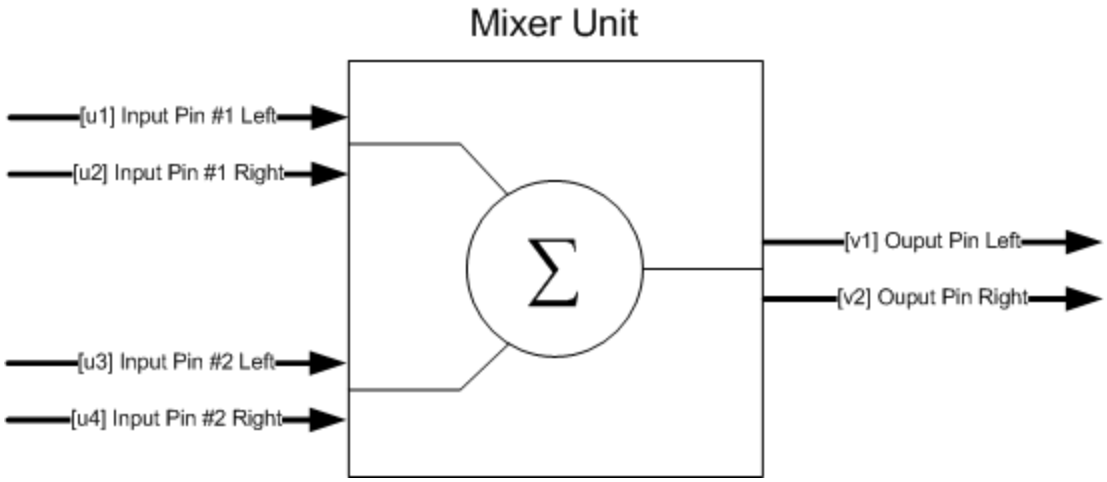

Figure - Mixer Unit Example

The following excerpt from section "4.3.2.3 Mixer Unit Descriptor" of audio 1.0 specification explains how the programmable controls are reported in `bmControls`:

"*This bitmap must be interpreted as a two-dimensional bit array that has a row for each logical input channel and a column for each logical output channel. If a bit at position [u, v] is set, this means that the Mixer Unit contains a programmable mixing Control that connects input channel u to output channel v. If bit [u, v] is clear, this indicates that the connection between input channel u and output channel v is non-programmable. Its fixed value can be retrieved through the appropriate request. The valid range for u is from one to n. The valid range for v is from one to m. The bmControls field stores the bit array row after row where the MSb of the first byte corresponds to the connection between input channel 1 and output channel 1. If (n x m) is not an integer multiple of 8, the bit array is padded with zeros until an integer number of bytes is occupied.*"

n represents the number of logical input channels and m the number of logical output channels.

Listing - Mixer Unit with Non-Programmable and Programmable Controls in the *USBD_Audio_MU_MixingCtrlSet* page shows the situation #2 where Mixer Unit has non-programmable and programmable controls. We consider that left logical input channels can be mixed in left logical output channel and right logical input channels can be mixed in right logical output channel. Table - Representation of Mixer Unit Programmable Controls in the *USBD_Audio_MU_MixingCtrlSet* page gives a representation of the programmable controls for our example:

Row	u \ v	1	2
row 1	1	'1'	'0'
row 2	2	'0'	'1'
row 3	3	'1'	'0'
row 4	4	'0'	'1'

Table - Representation of Mixer Unit Programmable Controls

As stated by the excerpt, the 2D bit array is stored row after row. Table - Representation of Mixer Unit Programmable Controls in the *USBD_Audio_MU_MixingCtrlSet* page would produce the following `bmControls` field:

Bit #	7	6	5	4	3	2	1	0
Row	row 1		row 2		row 3		row 4	
[u, v]	[1, 1]	[1, 2]	[2, 1]	[2, 2]	[3, 1]	[3, 2]	[4, 1]	[4, 2]
bmControls value	'1'	'0'	'0'	'1'	'1'	'0'	'0'	'1'

In this example, `bmControls` has a size of 1 byte requiring no zeros padding.

The resulting `USBD_Audio_MU_MixingCtrlSet()` calling sequence would be (for clarity, the error handling is not shown in this code snippet):

```
USBD_ERR    err;
CPU_INT08U  audio_nbr;

audio_nbr = USBD_Audio_Add(APP_CFG_USBD_AUDIO_NBR_ENTITY,
                           &USBD_Audio_DrvCommonAPI_Template,
                            DEF_NULL,
                           &err);

MU11_ID = USBD_Audio_MU_Add(audio_nbr,
                            &USBD_MU11_Cfg,
                            &USBD_Audio_DrvMU_API_Template,
                            &err);

USBD_Audio_MU_MixingCtrlSet(audio_nbr, MU11_ID, 1u, 1u, &err);
USBD_Audio_MU_MixingCtrlSet(audio_nbr, MU11_ID, 2u, 2u, &err);
USBD_Audio_MU_MixingCtrlSet(audio_nbr, MU11_ID, 3u, 1u, &err);
USBD_Audio_MU_MixingCtrlSet(audio_nbr, MU11_ID, 4u, 2u, &err);
```

Listing - Mixer Unit with Non-Programmable and Programmable Controls

USBD_Audio_AS_IF_Cfg

Description

Configure AudioStreaming interface settings by:

1. Allocating an AudioStreaming Settings structure.
2. Initializing this structure with the stream information.
3. Allocating buffers that will be used by the given AudioStreaming interface.

Files

`usbd_audio.h / usbd_audio.c`

Prototype

```
USBD_AUDIO_AS_IF_HANDLE  USBD_Audio_AS_IF_Cfg (const  USBD_AUDIO_STREAM_CFG          *p_stream_cfg,
                                               const  USBD_AUDIO_AS_IF_CFG          *p_as_if_cfg,
                                               const  USBD_AUDIO_DRV_AS_API         *p_as_api,
                                                      MEM_SEG                       *p_seg,
                                                      CPU_INT08U                     terminal_ID,
                                                      USBD_AUDIO_PLAYBACK_CORR_FNCT  corr_callback,
                                                      USBD_ERR                      *p_err);
```

Arguments

`p_stream_cfg`

Pointer to general stream configuration.

`p_as_if_cfg`

Pointer to the AudioStreaming interface configuration structure.

`p_as_api`

Pointer to AudioStreaming interface API.

`p_seg`

Pointer to memory segment used for buffers allocation. If null pointer, buffers are allocated from the heap. Otherwise, buffers are allocated from a memory region created by the application.

`terminal_ID`

Terminal ID associated to this AudioStreaming interface.

`corr_callback`

Application callback for user-defined correction algorithm.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
```

Returned Value

Handle to AudioStreaming interface, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

The listing Listing - USBD_Audio_AS_IF_Cfg() Example Usage in the *USBD_Audio_AS_IF_Cfg* page presents an example of usage using a memory segment to allocate the stream buffers. Listing Listing - Audio Class Initialization Example in the *Audio Class Instance Configuration* page shows the same function using a general heap for the stream buffers.

```
#define  APP_CFG_USBD_AUDIO_PLAYBACK_NBR_BUF              20u

#define  APP_CFG_USBD_AUDIO_BUF_EXTRA_BYTE          (USBD_AUDIO_STEREO *
USBD_AUDIO_FMT_TYPE_I_SUBFRAME_SIZE_2)

#define  APP_CFG_USBD_AUDIO_DEMO_MEM_SEG_SIZE      (APP_CFG_USBD_AUDIO_PLAYBACK_NBR_BUF            \
                                                    * (USBD_AUDIO_FMT_TYPE_I_SAMFREQ_48KHZ / 1000u) \
                                                    * USBD_AUDIO_STEREO                             \
                                                    * USBD_AUDIO_FMT_TYPE_I_SUBFRAME_SIZE_2         \
                                                    * APP_CFG_USBD_AUDIO_BUF_EXTRA_BYTE)

static  CPU_INT08U  App_USBD_Audio_MemSegData[APP_CFG_USBD_AUDIO_DEMO_MEM_SEG_SIZE];
static  MEM_SEG     App_USBD_Audio_MemSegInfo;

USBD_AUDIO_AS_IF_HANDLE  speaker_playback_as_if_handle;


Mem_SegCreate(         "Audio Data Memory Segment",                                  (1)
                       &App_USBD_Audio_MemSegInfo,
              (CPU_ADDR)App_USBD_Audio_MemSegData,
                        APP_CFG_USBD_AUDIO_DEMO_MEM_SEG_SIZE,
                        LIB_MEM_PADDING_ALIGN_NONE,
                       &err_lib);
if (err_lib!= LIB_MEM_ERR_NONE) {
    /* $$$$ Handle the error. */
}

speaker_playback_as_if_handle =  USBD_Audio_AS_IF_Cfg(&USBD_SpeakerStreamCfg,
                                                      &USBD_AS_IF1_SpeakerCfg,
                                                      &USBD_Audio_DrvAS_API_Simulation,
                                                      &App_USBD_Audio_MemSegInfo,       (2)
                                                       Speaker_IT_USB_OUT_ID,
                                                       DEF_NULL,
                                                      &err);
if (err != USBD_ERR_NONE) {
    /* $$$$ Handle the error. */
}
```

Listing - USBD_Audio_AS_IF_Cfg() Example Usage

(1) Create the memory segment. The memory segment will be created on a statically allocated buffer whose size will fit the necessary number of playback buffers. The base address of the memory segment data can also point to a controller dedicated memory. In that case, you may define the memory segment size to the the available dedicated memory size. Other stream buffers may be allocated from the same memory. In this example, the constant `LIB_MEM_PADDING_ALIGN_NONE` is passed to the function. This argument relates to the buffer padding done internally by µC/LIB when getting a buffer from the memory segment. The padding is done only if the argument is different from `LIB_MEM_PADDING_ALIGN_NONE`. Padding may be useful to avoid cache incoherence when a udio buffers are allocated from a cacheable memory region accessible to a DMA engine. Refer to page Memory Segments for more details about this function.

(2) Pass the memory segment structure to the function `USBD_Audio_AS_IF_Cfg()`. The audio class will create a memory pool into the given memory segment from which the audio class will get audio buffers while performing streaming.

USBD_Audio_AS_IF_Add

Description

Add the AudioStreaming interface to the given configuration.

Files

usbd_audio.h / usbd_audio.c

Prototype

```
void  USBD_Audio_AS_IF_Add (       CPU_INT08U                class_nbr,
                                   CPU_INT08U                cfg_nbr,
                                   USBD_AUDIO_AS_IF_HANDLE   as_if_handle,
                            const  USBD_AUDIO_AS_IF_CFG     *p_as_if_cfg,
                            const  CPU_CHAR                 *p_as_cfg_name,
                                   USBD_ERR                 *p_err);
```

Arguments

class_nbr

Class instance number.

cfg_nbr

Configuration number.

as_if_handle

Handle to AudioStreaming interface.

p_as_if_cfg

Pointer to AudioStreaming interface configuration.

p_as_cfg_name

Pointer to AudioStreaming interface name.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_AUDIO_AS_IF_ALLOC
```

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_Audio_AS_IF_StatGet

Description

Get the statistics structure associated to the given AudioStreaming interface.

Files

`usbd_audio.h / usbd_audio.c`

Prototype

```
USBD_AUDIO_STAT  *USBD_Audio_AS_IF_StatGet (USBD_AUDIO_AS_HANDLE  as_handle);
```

Arguments

`as_handle`

AudioStreaming interface handle.

Returned Value

Pointer to the audio stream statistics, if NO error(s).

Null Pointer, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Audio_RecordBufGet

Description

Get a buffer from an AudioStreaming interface pool.

Files

`usbd_audio_processing.h / usbd_audio_processing.c`

Prototype

```
void  *USBD_Audio_RecordBufGet (USBD_AUDIO_AS_HANDLE   as_handle,
                                CPU_INT16U            *p_buf_len);
```

Arguments

`as_handle`

AudioStreaming interface handle.

`p_buf_len`

Pointer to allocated buffer length (in bytes).

Returned Value

Pointer to record buffer, if NO error(s).

Null pointer, otherwise.

Callers

Audio Peripheral Driver.

Notes / Warnings

1. Most of the time, this function will be called from an ISR context except when a record is started on the codec side with the function `USBD_Audio_DrvStreamStart`.

USBD_Audio_RecordRxCmpl

Description

Signal to the Record task that a buffer is ready.

Files

`usbd_audio_processing.h / usbd_audio_processing.c`

Prototype

```
void  USBD_Audio_RecordRxCmpl (USBD_AUDIO_AS_HANDLE  as_handle);
```

Arguments

`as_handle`

AudioStreaming interface handle.

Returned Value

None.

Callers

Audio Peripheral Driver.

Notes / Warnings

1. Most of the time, this function will be called from an ISR context.

USBD_Audio_PlaybackTxCmpl

Description

Signal the end of the audio transfer to the playback task.

Files

`usbd_audio_processing.h / usbd_audio_processing.c`

Prototype

```
void  USBD_Audio_PlaybackTxCmpl (USBD_AUDIO_AS_HANDLE  as_handle);
```

Arguments

`as_handle`

AudioStreaming interface handle.

Returned Value

None.

Callers

Audio Peripheral Driver.

Notes / Warnings

1. Most of the time, this function will be called from an ISR context and when a playback is started on the codec side with the function `USBD_Audio_DrvStreamStart()` to signal how many buffers can be queued in the codec driver to start streaming.

USBD_Audio_PlaybackBufFree

Description

Signal the playback task that there is a buffer to free.

Files

usbd_audio_processing.h / usbd_audio_processing.c

Prototype

```
void  USBD_Audio_PlaybackBufFree (USBD_AUDIO_AS_HANDLE   as_handle,
                                  void                  *p_buf);
```

Arguments

as_handle

AudioStreaming interface handle.

p_buf

Pointer to buffer to free.

Returned Value

None.

Callers

Audio Peripheral Driver.

Notes / Warnings

1. Most of the time, this function will be called from an ISR context.

Deprecated Functions

The functions described in this section are deprecated and will be removed in a future release of µC/USB-Device. They should not be used in any new design.

- USBD_Audio_RecordBufFree

USBD_Audio_RecordBufFree

This function is DEPRECATED and will be removed in a future version of this product. The new record path implementation does not require the Audio Peripheral driver to free explicitly a consumed buffer.

Description

Free aborted record buffer.

Files

usbd_audio_processing.h / usbd_audio_processing.c

Prototype

```
void  USBD_Audio_RecordBufFree (USBD_AUDIO_AS_HANDLE   as_handle,
                                void                  *p_buf);
```

Arguments

as_handle

AudioStreaming interface handle.

p_buf

Pointer to buffer to free.

Returned Value

None.

Callers

Audio Peripheral Driver.

Notes / Warnings

1. Most of the time, this function will be called from an ISR context.

Audio Class OS Functions

- USBD_Audio_OS_Init
- USBD_Audio_OS_AS_IF_LockCreate
- USBD_Audio_OS_AS_IF_LockAcquire
- USBD_Audio_OS_AS_IF_LockRelease
- USBD_Audio_OS_RingBufQLockCreate
- USBD_Audio_OS_RingBufQLockAcquire
- USBD_Audio_OS_RingBufQLockRelease
- USBD_Audio_OS_RecordReqPost
- USBD_Audio_OS_RecordReqPend
- USBD_Audio_OS_PlaybackReqPost
- USBD_Audio_OS_PlaybackReqPend
- USBD_Audio_OS_DlyMs
- USBD_Audio_OS_RecordTask
- USBD_Audio_OS_PlaybackTask

USBD_Audio_OS_Init

Description

Initialize the audio class OS layer.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_Init (CPU_INT16U   msg_qty,
                          USBD_ERR    *p_err);
```

Arguments

msg_qty

Maximum quantity of messages for playback and record tasks' queues.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_INIT_FAIL

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. This function is called only once by the audio class. It usually performs the following operations:

   a. Create one queue for the playback task and one for the record task that can contain up to `msg_qty`.

   b. Create a task used for playback and another for record. If the task creation fails, set `p_err` to `USBD_ERR_OS_INIT_FAIL` and return.

   c. Set `p_err` to `USBD_ERR_NONE` if the initialization proceeded as expected.

USBD_Audio_OS_AS_IF_LockCreate

Description

Create an OS resource to use as an AudioStreaming interface lock.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_AS_IF_LockCreate (CPU_INT08U   as_if_nbr,
                                      USBD_ERR    *p_err);
```

Arguments

as_if_nbr

AudioStreaming interface index.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_SIGNAL_CREATE

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. In case of errors, `USBD_ERR_OS_SIGNAL_CREATE` should be returned. Otherwise, `USBD_ERR_NONE` should be used.

USBD_Audio_OS_AS_IF_LockAcquire

Description

Wait for an AudioStreaming interface to become available and acquire its lock.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_AS_IF_LockAcquire (CPU_INT08U   as_if_nbr,
                                       CPU_INT16U   timeout_ms,
                                       USBD_ERR    *p_err);
```

Arguments

as_if_nbr

AudioStreaming interface index.

timeout_ms

Lock wait timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Audio Class.

Implementation Guidelines

Typical implementation will consist in pending on a mutex or binary semaphore.

`p_err` argument should be assigned as described in following table.

Operation result	Error code to assign
No error	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error Code Assignment for Pending on a Lock

USBD_Audio_OS_AS_IF_LockRelease

Description

Release an AudioStreaming interface lock.

Files

`usbd_audio_os.h / usbd_audio_os.c`

Prototype

```
void  USBD_Audio_OS_AS_IF_LockRelease (CPU_INT08U  as_if_nbr);
```

Arguments

`as_if_nbr`

AudioStreaming interface index.

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

None.

USBD_Audio_OS_RingBufQLockCreate

Description

Create an OS resource to use as a stream ring buffer queue lock.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_RingBufQLockCreate (CPU_INT08U   as_if_settings_ix,
                                        USBD_ERR    *p_err);
```

Arguments

as_if_settings_ix

AudioStreaming interface settings index.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_SIGNAL_CREATE

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. In case of errors, `USBD_ERR_OS_SIGNAL_CREATE` should be returned. Otherwise, `USBD_ERR_NONE` should be used.

USBD_Audio_OS_RingBufQLockAcquire

Description

Wait for a stream ring buffer queue to become available and acquire its lock.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_RingBufQLockAcquire (CPU_INT08U   as_if_nbr,
                                         CPU_INT16U   timeout_ms,
                                         USBD_ERR    *p_err);
```

Arguments

as_if_settings_ix

AudioStreaming interface settings index.

timeout_ms

Lock wait timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. Typical implementation will consist in pending on a mutex or binary semaphore.

   `p_err` argument should be assigned as described in following table.

Operation result	Error code to assign
No error	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error Code Assignment for Pending on a Lock

USBD_Audio_OS_RingBufQLockRelease

Description

Release a stream ring buffer queue lock.

Files

`usbd_audio_os.h / usbd_audio_os.c`

Prototype

```
void  USBD_Audio_OS_RingBufQLockRelease (CPU_INT08U  as_if_settings_ix);
```

Arguments

`as_if_settings_ix`

AudioStreaming interface settings index.

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

None.

USBD_Audio_OS_RecordReqPost

Description

Post a request into the record task's queue.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  USBD_Audio_OS_RecordReqPost (void      *p_msg,
                                   USBD_ERR  *p_err);
```

Arguments

p_msg

Pointer to the record message.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. A call to this function should post a request into the queue associated to the record task. The request will be an AudioStreaming interface handle.

2. In case of errors, `USBD_ERR_OS_FAIL` should be returned. Otherwise, `USBD_ERR_NONE` should be used.

USBD_Audio_OS_RecordReqPend

Description

Pend on a request from the record task's queue.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  *USBD_Audio_OS_RecordReqPend (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL

Returned Value

Pointer to record request, if NO error(s).

Null pointer, otherwise

Callers

Audio Class.

Implementation guidelines

1. Typical implementation will consist in pending on a queue.

   `p_err` argument should be assigned as described in following table.

Operation result	Error code to assign
No error	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error Code Assignment for Pending on a Record Request

USBD_Audio_OS_PlaybackReqPost

Description

Post a request to the playback's task queue.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void   USBD_Audio_OS_PlaybackReqPost (void      *p_msg,
                                      USBD_ERR  *p_err);
```

Arguments

p_msg

Pointer to the message which is a playback event.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. A call to this function should post an event into the queue associated to the playback task.

2. In case of errors, `USBD_ERR_OS_FAIL` should be returned. Otherwise, `USBD_ERR_NONE` should be used.

USBD_Audio_OS_PlaybackReqPend

Description

Pend on a request from the playback's task queue.

Files

usbd_audio_os.h / usbd_audio_os.c

Prototype

```
void  *USBD_Audio_OS_PlaybackReqPend (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL

Returned Value

Pointer to playback request, if NO error(s).

Null pointer, otherwise.

Callers

Audio Class.

Implementation Guidelines

Typical implementation will consist in pending on a queue.

p_err argument should be assigned as described in following table.

Operation result	Error code to assign
No error	USBD_ERR_NONE
Pend timeout	USBD_ERR_OS_TIMEOUT
Pend aborted	USBD_ERR_OS_ABORT
Pend failed for any other reason	USBD_ERR_OS_FAIL

Table - Error Code Assignment for Pending on a Playback Request

USBD_Audio_OS_DlyMs

Description

Delay a task for a certain time.

Files

`usbd_audio_os.h / usbd_audio_os.c`

Prototype

```
void  USBD_Audio_OS_DlyMs (CPU_INT32U ms);
```

Arguments

`ms`

Delay in milliseconds.

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

1. The RTOS layer code should delay the calling task by the specified number of milliseconds.

OS Tick Rate

Whenever possible, `USBD_Audio_OS_DlyMs()` should provide a delay with a 1 ms granularity. That is the OS tick rate should be set to produce at least 1 tick per millisecond. It will help for the audio class tasks scheduling as audio class works on a 1 ms frame basis. Moreover, a retry mechanism is

implemented in the playback and record tasks in case a transfer cannot be queued on a given endpoint. The playback or record task waits 1 ms between each attempt before re-transmitting the transfer.

USBD_Audio_OS_RecordTask

Description

Process record data streams.

Files

`usbd_audio_os.h / usbd_audio_os.c`

Prototype

```
static  void  USBD_Audio_OS_RecordTask (void  *p_arg);
```

Arguments

`p_arg`

Pointer to task initialization argument.

Returned Value

None.

Callers

This is a task.

Implementation guidelines

1. The task should call `USBD_Audio_RecordTaskHandler()` function responsible for the record data streams management and defined in the Audio Processing module.

USBD_Audio_OS_PlaybackTask

Description

Processes playback data streams.

Files

`usbd_audio_os.h / usbd_audio_os.c`

Prototype

```
static  void  USBD_Audio_OS_PlaybackTask (void  *p_arg);
```

Arguments

`p_arg`

Pointer to task initialization argument.

Returned Value

None.

Callers

This is a task.

Implementation guidelines

1. The task should call `USBD_Audio_PlaybackTaskHandler()` function responsible for the playback data streams management and defined in the Audio Processing module.

Audio Peripheral Driver Functions

- USBD_Audio_DrvInit
- USBD_Audio_DrvCtrlOT_CopyProtSet
- USBD_Audio_DrvCtrlFU_MuteManage
- USBD_Audio_DrvCtrlFU_VolManage
- USBD_Audio_DrvCtrlFU_BassManage
- USBD_Audio_DrvCtrlFU_MidManage
- USBD_Audio_DrvCtrlFU_TrebleManage
- USBD_Audio_DrvCtrlFU_GraphicEqualizerManage
- USBD_Audio_DrvCtrlFU_AutoGainManage
- USBD_Audio_DrvCtrlFU_DlyManage
- USBD_Audio_DrvCtrlFU_BassBoostManage
- USBD_Audio_DrvCtrlFU_LoudnessManage
- USBD_Audio_DrvCtrlMU_CtrlManage
- USBD_Audio_DrvCtrlSU_InPinManage
- USBD_Audio_DrvAS_SamplingFreqManage
- USBD_Audio_DrvAS_PitchManage
- USBD_Audio_DrvStreamStart
- USBD_Audio_DrvStreamStop

- USBD_Audio_DrvStreamRecordRx
- USBD_Audio_DrvStreamPlaybackTx

USBD_Audio_DrvInit

Description

Initialize audio codec chip and different microcontroller peripherals needed to communicate with it.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  void  USBD_Audio_DrvInit (USBD_AUDIO_DRV  *p_audio_drv,
                                  USBD_ERR        *p_err);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_AUDIO_CODEC_INIT_FAILED
USBD_ERR_ALLOC

Returned Value

None.

Callers

Audio Class.

Implementation guidelines

Typical operations performed by the function `USBD_Audio_DrvInit()` consist in:

- Allocating an optional internal data structure using µC/LIB memory functions (refer to Memory Allocation Functions). This structure can then be saved in the `DataPtr` field of `USBD_AUDIO_DRV` structure.
- Setting all clocks required by peripherals (for instance I^2C, I^2S controllers, etc.) communicating with the audio chip.
- Setting any I/O required to interface peripherals to the audio chip.
- Setting any interrupt needed by peripherals. It may also encompass installing ISR (Interrupt Service Routine) for those peripherals (for instance, an ISR for I2C communication, an ISR for I2S communication if no DMA is used).
- Initializing the DMA controller if it is used to transfer audio data.
- Initializing the audio chip by configuring its registers accessed by I2C or SPI for instance.
- Initializing any software resources needed for the driver functioning. For instance:
  - Allocating DMA descriptors if it is required by your DMA controller implementation from the heap or a dedicated memory using µC/LIB memory functions (refer to Memory Allocation Functions).
  - Allocating the silence buffer using µC/LIB memory functions if a playback stream is required and initializing it with zeros using `Mem_Clr()`.
  - Allocating a dummy buffer using µC/LIB memory functions if a record stream is required.
- Set `p_err` to `USBD_ERR_AUDIO_CODEC_INIT_FAILED` if any error occurs during the initialization sequence. Otherwise set it to `USBD_ERR_NONE`.

`USBD_Audio_DrvInit()` will initialize the function pointer contained in the structure

USBD_AUDIO_DRV_COMMON_API.

USBD_Audio_DrvCtrlOT_CopyProtSet

Description

Set the Copy Protection Level for a particular Output Terminal.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlOT_CopyProtSet (USBD_AUDIO_DRV  *p_audio_drv,
                                                       CPU_INT08U       terminal_id,
                                                       CPU_INT08U       copy_prot_level);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id

Output Terminal ID.

copy_prot_level

Copy protection level to set.

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

None.

USBD_Audio_DrvCtrlFU_MuteManage

Description

Get or set mute state for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_MuteManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                      CPU_INT08U       unit_id,
                                                      CPU_INT08U       log_ch_nbr,
                                                      CPU_BOOLEAN      set_en,
                                                      CPU_BOOLEAN     *p_mute);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

set_en

Flag indicating to get or set the mute.

p_mute

Pointer to the mute state to get or set.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. `log_ch_nbr` allows you to get the mute state for a specific channel. When `log_ch_nbr` is 0, you get the mute state for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_VolManage

Description

Get or set volume for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_VolManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                     CPU_INT08U       req,
                                                     CPU_INT08U       unit_id,
                                                     CPU_INT08U       log_ch_nbr,
                                                     CPU_INT16U      *p_vol);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Volume request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`p_vol`

Pointer to the volume value to set or get.

Endianness

`p_vol` uses a little endian memory organization. Hence, you should use µC/LIB macros `MEM_VAL_GET_INT16U_LITTLE()` and `MEM_VAL_SET_INT16U_LITTLE()` when reading or writing the volume from/to it. This will ensure data is accessed correctly regarding your CPU endianness.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. `log_ch_nbr` allows you to get or set the volume for a specific channel. When `log_ch_nbr` is 0, you get or set the volume for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

This listing shows an example usage of this function.

```
#define  FU_SPEAKER_ID              6u
#define  FU_MIC_ID                  7u

#define  CODEC_MASTER_VOL_CTRL_MIN_dB               0xC200u
#define  CODEC_MASTER_VOL_CTRL_MAX_dB               0x0000u
#define  CODEC_MASTER_VOL_CTRL_RES_dB               0x0040u
#define  CODEC_MASTER_VOL_CTRL_SILENCE                 252u
#define  CODEC_DECIMATOR_VOL_CTRL_MIN_dB            0xBF80u
#define  CODEC_DECIMATOR_VOL_CTRL_MAX_dB            0x1800u
#define  CODEC_DECIMATOR_VOL_CTRL_RES_dB            0x0080u
#define  CODEC_DECIMATOR_VOL_CTRL_SILENCE              128u


static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_VolManage (CPU_INT08U   req,
                                                     CPU_INT08U   unit_id,
                                                     CPU_INT08U   log_ch_nbr,
                                                     CPU_INT16U  *p_vol)
{
    CPU_INT16S  temp;
    CPU_INT08U  vol_to_set;


    (void)&log_ch_nbr;

    switch (req) {                          (1)
        case USBD_AUDIO_REQ_GET_CUR:                    (2)
             if (unit_id == FU_SPEAKER_ID) {
                 *p_vol = FU_VolSpeaker;
             } else {
                 *p_vol = FU_VolMic;
             }
             break;


        case USBD_AUDIO_REQ_GET_MIN:                    (3)
             if (unit_id == FU_SPEAKER_ID) {
                 *p_vol = CODEC_MASTER_VOL_CTRL_MIN_dB;       /* -78 dB.
*/
             } else {
                 *p_vol = CODEC_DECIMATOR_VOL_CTRL_MIN_dB;    /* -63.5 dB.
*/
             }
             break;


        case USBD_AUDIO_REQ_GET_MAX:                    (4)
             if (unit_id == FU_SPEAKER_ID) {
                 *p_vol = CODEC_MASTER_VOL_CTRL_MAX_dB;       /* 0 dB.
*/
             } else {
                 *p_vol = CODEC_DECIMATOR_VOL_CTRL_MAX_dB;    /* +24 dB.
*/
             }
             break;


        case USBD_AUDIO_REQ_GET_RES:                    (5)
             if (unit_id == FU_SPEAKER_ID) {
                 *p_vol = CODEC_MASTER_VOL_CTRL_RES_dB;       /* Resolution of 0.25 dB.
*/
             } else {
                 *p_vol = CODEC_DECIMATOR_VOL_CTRL_RES_dB;    /* Resolution of 0.5 dB.
*/
             }
             break;
```

```
        case USBD_AUDIO_REQ_SET_CUR:
             if (unit_id == FU_SPEAKER_ID) {
                 FU_VolSpeaker = *p_vol;                    (6)
                 if (*p_vol == USBD_AUDIO_FU_CTRL_VOL_SILENCE) {
                     vol_to_set = CODEC_MASTER_VOL_CTRL_SILENCE;
                 }
                                                              /* Negative dB.
*/
                                                                        (7)
                 temp       = (CPU_INT16S)*p_vol;
                 vol_to_set = (CPU_INT08U)(-temp / CODEC_MASTER_VOL_CTRL_RES_dB);
                                                                        (8)
                 /* $$$$ Set volume level for record stream in audio chip. */


             } else
                 FU_VolMic = *p_vol;
                 if (*p_vol == USBD_AUDIO_FU_CTRL_VOL_SILENCE) {
                     vol_to_set = CODEC_DECIMATOR_VOL_CTRL_SILENCE;
                 }
                                                               /* Negative dB.
*/
                 if (*p_vol > CODEC_DECIMATOR_VOL_CTRL_MIN_dB) {              (7)
                     temp       = (CPU_INT16S)*p_vol;
                     vol_to_set = (CPU_INT08U)(-temp / CODEC_DECIMATOR_VOL_CTRL_RES_dB);
                 } else {                                      /* Positive dB.
*/
                     vol_to_set = (CPU_INT08U)(*p_vol / CODEC_DECIMATOR_VOL_CTRL_RES_dB);
                 }
                                                                        (8)
                 /* $$$$ Set volume level for record stream in audio chip. */
             }
             break;

        default:
             return (DEF_FAIL);
    }

    return (DEF_OK);
}
```

Listing - USBD_Audio_DrvCtrlFU_VolManage() Example Usage

(1) The request type may be processed in a switch statement. Volume control supports all `GET_XXX` requests and `SET_CUR`.

(2) By using the Feature Unit ID, unique within the audio function, you can return the current volume level stored for example in a local global variable updated by the `SET_CUR` request case.

(3) By using the Feature Unit ID, you can return the minimum volume level possible. This value will be a constant. In this example, the volume attenuation for the speaker ranges

from 0 to -78 dB and corresponds to the range supported by the audio codec. Thus, the minimum value is -78 dB. Refer to the warning below about volume levels range translated from audio codec range to Audio 1.0 specification range and vice versa.

(4) By using the Feature Unit ID, you can return the maximum volume level possible. This value will be a constant. In this example, the volume attenuation for the speaker ranges from 0 to -78 dB and corresponds to the range supported by the audio codec. Thus, the minimum value is 0 dB. Refer to the warning below about volume levels range translated from audio codec range to Audio 1.0 specification range and vice versa.

(5) By using the Feature Unit ID, you can return the resolution supported by the audio codec to change the volume level. This value will be a constant. In this example, the codec uses steps of 0.25 dB. Refer to the warning below about volume levels range translated from audio codec range to Audio 1.0 specification range and vice versa.

(6) For the request `SET_CUR`, you can update the volume level stored for instance locally in a global variable. This global variable will be read by `GET_CUR` request case.

(7) The volume value pointed by `p_vol` is translated into a value understood by the audio codec. Indeed, the volume value sent by the host corresponds to the volume range defined by Audio 1.0 specification. This range does not necessarily match the volume range supported by the audio codec. Refer to the warning below about volume levels range translated from Audio 1.0 specification range to audio codec range to and vice versa.

(8) You should use the proper functions to set the volume level in the audio codec. It could be for instance sending some I^2C commands to write in some audio codec registers. Return `DEF_FAIL` if any error occurs during the volume setting.

Volume Values Range Conversion (Audio Chip <-> Audio 1.0 spec)

When manipulating volume levels, be aware that you must handle the conversion from the audio codec range to the Audio 1.0 specification range and vice versa. Audio 1.0 indicates that the volume can range from +127.9961 dB (0x7FFF) down to -127.9961 dB (0x8001) in steps of 1/256 dB (0x0001) with an extended value of 0x8000 for -dB.

For `GET_MIN`, `GET_MAX`, and `GET_RES` request, respectively, the minimum, maximum and step volume must be translated from the audio codec range to the Audio 1.0 range.

For SET_CUR request, the volume level must be translated from the Audio 1.0 range to the audio codec range.

USBD_Audio_DrvCtrlFU_BassManage

Description

Get or set bass for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_BassManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                      CPU_INT08U       req,
                                                      CPU_INT08U       unit_id,
                                                      CPU_INT08U       log_ch_nbr,
                                                      CPU_INT08U      *p_bass);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Bass request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`p_bass`

Pointer to the Bass value to set or get.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Bass Control values range allowed for Feature Unit is:

    a. From +31.75 dB (0x7F) down to –32.00 dB (0x80) for CUR, MIN, and MAX attributes.

    b. From 0.25 dB (0x01) to +31.75 dB (0x7F) for RES attribute.

2. `log_ch_nbr` allows you to get or set the bass for a specific channel. When `log_ch_nbr` is 0, you get or set the bass for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_MidManage

Description

Get or set middle for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_MidManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                     CPU_INT08U       req,
                                                     CPU_INT08U       unit_id,
                                                     CPU_INT08U       log_ch_nbr,
                                                     CPU_INT08U      *p_mid);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Middle request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`p_mid`

Pointer to the Middle value to set or get.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Middle Control values range allowed for Feature Unit is:

   a. From +31.75 dB (0x7F) down to –32.00 dB (0x80) for CUR, MIN, and MAX attributes.

   b. From 0.25 dB (0x01) to +31.75 dB (0x7F) for RES attribute.

2. `log_ch_nbr` allows you to get or set the middle for a specific channel. When `log_ch_nbr` is 0, you get or set the middle for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_TrebleManage

Description

Get or set treble for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_TrebleManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                        CPU_INT08U       req,
                                                        CPU_INT08U       unit_id,
                                                        CPU_INT08U       log_ch_nbr,
                                                        CPU_INT08U      *p_treble);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Treble request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`p_treble`

Pointer to the Treble value to set or get.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Treble Control values range allowed for Feature Unit is:

    a. From +31.75 dB (0x7F) down to –32.00 dB (0x80) for CUR, MIN, and MAX attributes.

    b. From 0.25 dB (0x01) to +31.75 dB (0x7F) for RES attribute.

2. `log_ch_nbr` allows you to get or set the treble for a specific channel. When `log_ch_nbr` is 0, you get or set the treble for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_GraphicEqualizerManage

Description

Get or set graphic equalizer for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_GraphicEqualizerManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                                  CPU_INT08U       req,
                                                                  CPU_INT08U       unit_id,
                                                                  CPU_INT08U       log_ch_nbr,
                                                                  CPU_INT08U       nbr_bands_present,
                                                                  CPU_INT32U      *p_bm_bands_present,
                                                                  CPU_INT08U      *p_buf);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Graphic Equalizer request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`nbr_bands_present`

Number of band presents. Used only with `SET_CUR` request.

`p_bm_bands_present`

Pointer to bitmap indicating a new setting for one or several frequency band(s). Used only with `SET_CUR` request.

`p_buf`

Pointer to the request value to set or get.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Graphic Equalizer Control values range allowed for Feature Unit is:

    a. From +31.75 dB (0x7F) down to –32.00 dB (0x80) for CUR, MIN, and MAX attributes.

    b. From 0.25 dB (0x01) to +31.75 dB (0x7F) for RES attribute.

2. `log_ch_nbr` allows you to get or set the graphic equalizer for a specific channel. When `log_ch_nbr` is 0, you get or set the graphic equalizer for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_AutoGainManage

Description

Get the auto gain state for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_AutoGainManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                          CPU_INT08U       unit_id,
                                                          CPU_INT08U       log_ch_nbr,
                                                          CPU_BOOLEAN      set_en,
                                                          CPU_BOOLEAN     *p_auto_gain);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

set_en

Flag indicating to get or set the gain.

p_auto_gain

Pointer to the auto gain value to get or set.

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

1. log_ch_nbr allows you to get the auto gain state for a specific channel. When log_ch_nbr is 0, you get the auto gain state for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_DlyManage

Description

Get or set the delay for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_DlyManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                     CPU_INT08U       req,
                                                     CPU_INT08U       unit_id,
                                                     CPU_INT08U       log_ch_nbr,
                                                     CPU_INT16U      *p_dly);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Bass request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

`p_dly`

Pointer to the Delay request value to set or get

Endianness

`p_dly` uses a little endian memory organization. Hence, you should use µC/LIB macros `MEM_VAL_GET_INT16U_LITTLE()` and `MEM_VAL_SET_INT16U_LITTLE()` when reading or writing the delay from/to it. This will ensure data is accessed correctly regarding your CPU endianness.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Delay Control values range allowed for Feature Unit is:
    a. From 0 (0x0000) to 1023.9844ms (0xFFFF) for CUR, MIN, MAX and RES attributes.
2. `log_ch_nbr` allows you to get or set the delay for a specific channel. When `log_ch_nbr` is 0, you get or set the delay for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_BassBoostManage

Description

Get or set bass boost state for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_BassBoostManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                           CPU_INT08U       unit_id,
                                                           CPU_INT08U       log_ch_nbr,
                                                           CPU_BOOLEAN      set_en,
                                                           CPU_BOOLEAN     *p_bass_boost);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

set_en

Flag indicating to get or set the bass boost.

p_bass_boost

Pointer to the bass boost value to get or set.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. `log_ch_nbr` allows you to get the bass boost state for a specific channel. When `log_ch_nbr` is 0, you get the bass boost state for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlFU_LoudnessManage

Description

Get or set loudness state for one or all logical channels inside a cluster.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlFU_LoudnessManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                          CPU_INT08U       unit_id,
                                                          CPU_INT08U       log_ch_nbr,
                                                          CPU_BOOLEAN      set_en,
                                                          CPU_BOOLEAN     *p_loudness);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

unit_id

Feature Unit ID.

log_ch_nbr

Logical channel number.

set_en

Flag indicating to get or set the loudness.

p_loudness

Pointer to the bass boost value to get or set.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. `log_ch_nbr` allows you to get the loudness state for a specific channel. When `log_ch_nbr` is 0, you get the loudness state for all channels. Indeed, the logical channel #0 represents the master channel and encompasses all channels.

USBD_Audio_DrvCtrlMU_CtrlManage

Description

Get or set the mix status for a certain logical input and output channels couple.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlMU_CtrlManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                      CPU_INT08U       req,
                                                      CPU_INT08U       unit_id,
                                                      CPU_INT08U       log_in_ch_nbr,
                                                      CPU_INT08U       log_out_ch_nbr,
                                                      CPU_INT16U      *p_ctrl);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

req

Mixer status request:

USBD_AUDIO_REQ_GET_CUR
USBD_AUDIO_REQ_GET_RES
USBD_AUDIO_REQ_GET_MIN
USBD_AUDIO_REQ_GET_MAX
USBD_AUDIO_REQ_SET_CUR

unit_id

Mixer Unit ID.

log_in_ch_nbr

Logical channel In number.

`log_out_ch_nbr`

Logical channel Out number.

`p_ctrl`

Pointer to the mixer control request value to get or set.

Endianness

`p_ctrl` uses a little endian memory organization. Hence, you should use µC/LIB macros `MEM_VAL_GET_INT16U_LITTLE()` and `MEM_VAL_SET_INT16U_LITTLE()` when reading or writing the mixing control value from/to it. This will ensure data is accessed correctly regarding your CPU endianness.

Returned Value

`DEF_OK`, if NO error(s) occurred and request is supported.

`DEF_FAIL`, otherwise.

Callers

Audio Class.

Implementation guidelines

1. The Mixer Control values range allowed for Mixer Unit is:

    a. From +127.9961 dB (0x7FFF) down to -127.9961 dB (0x8001) for CUR, MIN, and MAX attributes.

    b. From 1/256 dB (0x0001) to +127.9961 dB (0x7FFF) for RES attribute.

USBD_Audio_DrvCtrlSU_InPinManage

Description

Get or set the Input Pin of a particular Selector Unit.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvCtrlSU_InPinManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                       CPU_INT08U       unit_id,
                                                       CPU_BOOLEAN      set_en,
                                                       CPU_INT08U      *p_in_pin_nbr);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

unit_id

Selector Unit ID.

set_en

Flag indicating to get or set the loudness.

p_in_pin_nbr

Pointer to the input number to get or set.

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

None.

USBD_Audio_DrvAS_SamplingFreqManage

Description

Get or set current sampling frequency for a particular Terminal (i.e. endpoint).

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvAS_SamplingFreqManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                          CPU_INT08U       terminal_id_link,
                                                          CPU_BOOLEAN      set_en,
                                                          CPU_INT32U      *p_sampling_freq);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id_link

AudioStreaming terminal link.

set_en

Flag indicating to get or set the sampling frequency.

p_sampling_freq

Pointer to the sampling frequency value to get or set.

Endianness

p_sampling_freq uses a little endian memory organization. Hence, you should use the µC/LIB macro MEM_VAL_SET_INT32U_LITTLE() when writing the sampling frequency to it. This will ensure data is accessed correctly regarding your CPU endianness.

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

This listing shows an example usage of this function.

```
static  CPU_INT32U  ActualSamplingFreq = USBD_AUDIO_FMT_TYPE_I_SAMFREQ_48KHZ;

static  CPU_BOOLEAN  USBD_Audio_DrvAS_SamplingFreqManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                          CPU_INT08U       terminal_id_link,
                                                          CPU_BOOLEAN      set_en,
                                                          CPU_INT32U      *p_sampling_freq)
{
    CPU_BOOLEAN  req_ok = DEF_FAIL;


    (void)&p_audio_drv;
    (void)&terminal_id_link;

                                                                  /* ----------------------- GET
------------------------ */
    if (set_en == DEF_FALSE) {                                                  (1)
       *p_sampling_freq = ActualSamplingFreq;                                   (2)
        req_ok = DEF_OK;
                                                                  /* ----------------------- SET
------------------------ */
    } else {
        if ((*p_sampling_freq == USBD_AUDIO_FMT_TYPE_I_SAMFREQ_44_1KHZ) ||      (3)
            (*p_sampling_freq == USBD_AUDIO_FMT_TYPE_I_SAMFREQ_48KHZ)) {
            ActualSamplingFreq = *p_cur_sampling_freq;                          (4)
            req_ok = DEF_OK;
        }
    }
    return (req_ok);                                                            (5)
}
```

Listing - USBD_Audio_DrvCtrlAS_SamplingFreqGet() Example Usage

(1) Based on argument set_en, determine if the function is called to get or set the current

sampling frequency.

(2) If `set_en` is `DEF_FALSE`, the actual sampling frequency is given and set the flag indicating that the request has been correctly processed is set to `DEF_OK`.

(3) If `set_en` is `DEF_TRUE`, the sampling frequency currently in use needs to be changed. Make sure the requested sampling frequency is supported by the codec used.

(4) If the sampling frequency is supported, update the sampling frequency used and set the flag indicating that the request has been correctly processed is set to `DEF_OK`. More actions may be required to actually change the sampling frequency of the codec, such as writing to registers or sending various commands to the codec.

(5) Return status of the function.

USBD_Audio_DrvAS_PitchManage

Description

Get or set pitch state for a particular Terminal (i.e. endpoint).

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvAS_PitchManage (USBD_AUDIO_DRV  *p_audio_drv,
                                                   CPU_INT08U       terminal_id_link,
                                                   CPU_BOOLEAN      set_en,
                                                   CPU_BOOLEAN     *p_pitch);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id_link

AudioStreaming terminal link.

set_en

Flag indicating to get or set the pitch.

p_pitch

Pointer to current pitch control state.

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

None.

USBD_Audio_DrvStreamStart

Description

Start record or playback stream.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvStreamStart (USBD_AUDIO_DRV        *p_audio_drv,
                                                USBD_AUDIO_AS_HANDLE   as_handle,
                                                CPU_INT08U             terminal_id_link);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

as_handle

AudioStreaming handle.

terminal_id_link

AudioStreaming terminal link.

Returned Value

DEF_OK, if NO error(s) occurred.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

This listing shows an example usage of this function for a record stream.

```
#define  DMA_CH_1                                  1u
#define  DMA_MAX_NBR_CH_USED                       2u
static   CPU_INT08U  DMA_AsHandleTbl[DMA_MAX_NBR_CH_USED];

static  CPU_BOOLEAN  USBD_Audio_DrvStreamStart (USBD_AUDIO_DRV  *p_audio_drv,
                                                CPU_INT08U       as_handle,
                                                CPU_INT08U       terminal_id_link)
{
    CPU_INT16U   buf_len;
    CPU_INT08U  *p_buf;


    (void)&p_audio_drv;

    if (terminal_id_link != Mic_OT_USB_IN_ID) {                              (1)
        return (DEF_FAIL);
    }

    DMA_AsHandleTbl[DMA_CH_1] = as_handle;                                   (2)

    /* $$$$ Enable record operations on audio chip if needed. */             (3)

    p_buf = (CPU_INT08U *)USBD_Audio_RecordBufGet(as_handle,                 (4)
                                                  &buf_len);
    if (p_buf == (CPU_INT08U *)0) {
        return (DEF_FAIL);
    }

    /* $$$$ Prepare initial DMA transfer. */             (5)

    return (DEF_OK);
}
```

Listing - USBD_Audio_DrvStreamStart() Example Usage for a Record Stream

(1) You can verify if the stream processing is intended for the record path by checking the terminal ID associated to this stream. The function may be used to process also a playback stream. In that case, the terminal ID will differentiate the different streams.

(2) You should save the AudioStreaming interface handle locally to be used by the ISR managing audio data transfer. This handle is used by the functions `USBD_Audio_RecordBufGet()`, `USBD_Audio_RecordRxCmpl` and `USBD_Audio_RecordBufFree()`.

(3) You may have to enable the record operations on the audio chip. It could be enabling

some clocks, enabling some modules, etc. It may require sending a series of I2C commands for registers access,

(4) Call `USBD_Audio_RecordBufGet()` to get an empty record buffer.

(5) With this buffer, prepare the initial DMA transfer.

> If the peripheral managing audio data transfer can work with a DMA controller, we recommend using DMA transfers. It will improve overall audio performance by offloading the CPU. As opposed to CPU copies involved for FIFO transfers that would be time-consuming.

This listing shows an example usage of this function for a playback stream.

```
#define  DMA_CH_0                                   0u
#define  DMA_MAX_NBR_CH_USED                        2u
static   CPU_INT08U  DMA_AsHandleTbl[DMA_MAX_NBR_CH_USED];

static  CPU_BOOLEAN  USBD_Audio_DrvStreamStart (USBD_AUDIO_DRV  *p_audio_drv,
                                                CPU_INT08U       as_handle,
                                                CPU_INT08U       terminal_id_link)
{
    CPU_INT16U   buf_len;
    CPU_INT08U  *p_buf;


    (void)&p_audio_drv;

    if (terminal_id_link != Speaker_IT_USB_OUT_ID) {        (1)
        return (DEF_FAIL);
    }

    DMA_AsHandleTbl[DMA_CH_0] = as_handle;                                  (2)

    /* $$$$ Enable playback operations on audio chip if needed. */         (3)


    USBD_Audio_PlaybackTxCmpl(as_handle);                                   (4)
    USBD_Audio_PlaybackTxCmpl(as_handle);

    return (DEF_OK);
}
```

Listing - USBD_Audio_DrvStreamStart() Example Usage for a Playback Stream

(1) You can verify if the stream processing is intended for the playback path by checking the terminal ID associated to this stream. The function may be used to process also a record

stream. In that case, the terminal ID will differentiate the different streams.

(2) You should save the AudioStreaming interface handle locally to be used by the ISR managing audio data transfer. This handle is used by the functions `USBD_Audio_PlaybackTxCmpl` and `USBD_Audio_PlaybackBufFree()`.

(3) You may have to enable the playback operations on the audio chip. It could be enabling some clocks, enabling some modules, etc. It may require sending a series of I2C commands for registers access,

(4) Signal to the playback task the number of ready buffers that can be queued by calling `USBD_Audio_PlaybackTxCmpl()`. In this example, 2 playback buffers ready to be consumed can be queued. The playback task will receive 2 events CODEC XFER CMPL and will call `USBD_Audio_DrvStreamPlaybackTx` to provide to the code driver ready buffers. The codec driver could initiate for instance a DMA transfer from `USBD_Audio_DrvStreamPlaybackTx()` to start the streaming.

USBD_Audio_DrvStreamStop

Description

Stop record or playback stream.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  CPU_BOOLEAN  USBD_Audio_DrvStreamStop (USBD_AUDIO_DRV  *p_audio_drv,
                                               CPU_INT08U       terminal_id_link);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id_link

AudioStreaming terminal link.

Returned Value

DEF_OK, if NO error(s) occurred.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

This listing shows an example usage of this function handling record and playback streams.

```
static  CPU_BOOLEAN  USBD_Audio_DrvStreamStop (USBD_AUDIO_DRV  *p_audio_drv,
                                               CPU_INT08U       terminal_id_link)
{
    if (terminal_id_link == Mic_OT_USB_IN_ID) {                                 (1)
     /* $$$$ Abort ongoing DMA transfer(s). */                (2)

     /* $$$$ Disable record operations on audio chip. */                (3)

    } else if (terminal_id_link == Speaker_IT_USB_OUT_ID) {
     /* $$$$ Abort ongoing DMA transfer(s). */                (2)

     /* $$$$ Disable playback operations on audio chip. */             (3)
    }

    return (DEF_OK);
}
```

Listing - USBD_Audio_DrvStreamStop() Example Usage

(1) If the function handles different streams, the terminal ID allows to differentiate them.

(2) Abort ongoing DMA transfer(s). The buffer associated to the DMA transfer may be freed with `USBD_Audio_RecordBufFree()` or `USBD_Audio_PlaybackBufFree` into the ISR responsible for DMA data transfer completion.

(3) You may have to disable the record or playback operations on the audio chip. It could be disabling some clocks, disabling some modules, etc. It may require sending a series of I2C commands for registers access,

USBD_Audio_DrvStreamRecordRx

Description

Get a ready record buffer from codec.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  void  USBD_Audio_DrvStreamRecordRx (USBD_AUDIO_DRV  *p_audio_drv,
                                            CPU_INT08U       terminal_id_link,
                                            void            *p_buf,
                                            CPU_INT16U      *p_buf_len,
                                            USBD_ERR        *p_err)
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id_link

Terminal ID associated to this stream.

p_buf

Pointer to record buffer.

p_buf_len

Pointer to buffer length in bytes.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

USBD_ERR_RX

Returned Value

None.

...Pointer to a ready record buffer, if NO error(s) occurred.

Null pointer, otherwise.

Callers

Audio Class.

Implementation guidelines

This listing shows an example usage of this function.

```
static  void  *USBD_Audio_DrvStreamRecordRx (USBD_AUDIO_DRV  *p_audio_drv,
                                             CPU_INT08U       terminal_id_link,
            void            *p_buf,
                                             CPU_INT16U      *p_buf_len,
                                             USBD_ERR        *p_err)
{
 /* $$$$ Retrieve a ready record buffer. */               (1)
}
```

Listing - USBD_Audio_DrvStreamRecordRx() Example Usage

(1) When an audio transfer is completed, the codec driver should signal the record task that a buffer is ready by sending calling the function `USBD_Audio_RecordRxCmpl`. This function will post a request to the record task. Upon reception of this request, the record task will call the function `USBD_Audio_DrvStreamRecordRx()` through a function pointer to get a ready buffer. If the audio peripheral uses DMA, the buffer is already ready and there is nothing special to do in this function. If the audio peripheral uses a FIFO mode, you may have to copy the received audio samples to the buffer referenced by the pointer `p_buf`.

> The Audio Peripheral Driver should support at least double-buffering to optimize the record streaming.

(2) In case of error, you may return the error code `USBD_ERR_RX`.

USBD_Audio_DrvStreamPlaybackTx

Description

Provide a ready playback buffer to codec.

Files

usbd_audio_drv_<codec-name>.h / usbd_audio_drv_<codec-name>.c

Prototype

```
static  void  USBD_Audio_DrvStreamPlaybackTx (USBD_AUDIO_DRV  *p_audio_drv,
                                              CPU_INT08U       terminal_id_link,
                                              void            *p_buf,
                                              CPU_INT16U       buf_len,
                                              USBD_ERR        *p_err);
```

Arguments

p_audio_drv

Pointer to audio driver structure.

terminal_id_link

Terminal ID associated to this stream.

p_buf

Pointer to ready playback buffer.

buf_len

Buffer length in bytes.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

USBD_ERR_TX

Returned Value

DEF_OK, if NO error(s) occurred and request is supported.

DEF_FAIL, otherwise.

Callers

Audio Class.

Implementation guidelines

This listing shows an example usage of this function.

```
#define  DMA_CH_0                                   0u
#define  DMA_MAX_NBR_CH_USED                        2u
static   CPU_INT08U  DMA_AsHandleTbl[DMA_MAX_NBR_CH_USED];

static  void  USBD_Audio_DrvStreamPlaybackTx (USBD_AUDIO_DRV  *p_audio_drv,
                                              CPU_INT08U       terminal_id_link,
                                              void            *p_buf,
                                              CPU_INT16U       buf_len,
                                              USBD_ERR        *p_err)
{
    /* $$$$ Store ready playback buffer. */                      (1)
    /* $$$$ Prepare DMA transfer for initial buffer. */          (2)
}
```

Listing - USBD_Audio_DrvStreamPlaybackTx() Example Usage

(1) The ready buffer provided by the audio class could be stored internally into the codec driver using any buffer memory management method (for example a circular buffer). Then the stored ready buffer could be retrieved by the audio peripheral ISR.

> The Audio Peripheral Driver should support at least double-buffering to optimize the playback streaming.

(2) If the Audio Peripheral driver support buffer queuing, you may accumulate a certain

number of ready buffers (for instance, 2 buffers should be enough). When the buffers accumulation is done, you should prepare the initial DMA transfer. When the stream is launched, the next DMA transfer may be prepared in an interrupt context when the current DMA transfer completes.

> If the peripheral managing audio data transfer can work with a DMA controller, we recommend using DMA transfers. It will improve overall audio performance by offloading the CPU. As opposed to CPU copies involved for FIFO transfers that would be time-consuming.

(3) In case of error, you may return the error code `USBD_ERR_TX`.

API - Communication Device Class

- USBD_ACM_SerialLineCodingGet
- USBD_ACM_SerialLineCodingSet
- USBD_ACM_SerialLineCodingReg
- USBD_ACM_SerialLineStateSet
- USBD_ACM_SerialLineStateClr

CDC Functions

- USBD_CDC_Init
- USBD_CDC_Add
- USBD_CDC_CfgAdd
- USBD_CDC_IsConn
- USBD_CDC_DataIF_Add
- USBD_CDC_DataRx
- USBD_CDC_DataTx
- USBD_CDC_Notify

USBD_CDC_Init

Description

Initializes all the internal variables and modules used by the CDC. The initialization function is called by the application exactly once.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
static  void  USBD_CDC_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function:
USBD_ERR_NONE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_CDC_Add

Description

Creates a CDC instance.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_INT08U  USBD_CDC_Add(CPU_INT08U              subclass,
                         USBD_CDC_SUBCLASS_DRV  *p_subclass_drv,
                         void                   *p_subclass_arg,
                         CPU_INT08U              protocol,
                         CPU_BOOLEAN             notify_en,
                         CPU_INT16U              notify_interval,
                         USBD_ERR               *p_err);
```

Arguments

subclass

CDC subclass code.

Subclass code	Description
USBD_CDC_SUBCLASS_DLCM	Direct Line Control Model
USBD_CDC_SUBCLASS_ACM	Abstract Control Model
USBD_CDC_SUBCLASS_TCM	Telephone Control Model
USBD_CDC_SUBCLASS_MCC	Multi-Channel Control Model
USBD_CDC_SUBCLASS_CAPICM	CAPI Control Model
USBD_CDC_SUBCLASS_WHCM	Wireless Handset Control Model
USBD_CDC_SUBCLASS_DEV_MGMT	Device Management
USBD_CDC_SUBCLASS_MDLM	Mobile Direct Line Model
USBD_CDC_SUBCLASS_OBEX	Obex
USBD_CDC_SUBCLASS_EEM	Ethernet Emulation Model

USBD_CDC_SUBCLASS_NCM	Network Control Model
USBD_CDC_SUBCLASS_VENDOR	Vendor specific

CDC subclass codes are defined in the Universal Serial Bus Class Definitions for Communication Devices Revision 2.1 Table 4.

p_subclass_drv

Pointer to CDC subclass driver.

p_subclass_arg

Pointer to CDC subclass driver argument.

protocol

CDC protocol code:

Protocol code	Description
USBD_CDC_COMM_PROTOCOL_NONE	None
USBD_CDC_COMM_PROTOCOL_AT_V250	AT commands V250
USBD_CDC_COMM_PROTOCOL_AT_PCCA_101	AT commands defined by PCCA 101
USBD_CDC_COMM_PROTOCOL_AT_PCCA_101_ANNEX	AT commands defined by PCCA 101 & annex O
USBD_CDC_COMM_PROTOCOL_AT_GSM_7_07	AT commands defined by GSM 7.07
USBD_CDC_COMM_PROTOCOL_AT_3GPP_27_07	AT commands defined by 3GPP 27.007
USBD_CDC_COMM_PROTOCOL_AT_TIA_CDMA	AT commands defined by TIA for CDMA
USBD_CDC_COMM_PROTOCOL_EEM	Ethernet Emulation Model
USBD_CDC_COMM_PROTOCOL_EXT	External protocol
USBD_CDC_COMM_PROTOCOL_VENDOR	Vendor specific

CDC protocol codes are defined in the Universal Serial Bus Class Definitions for Communication Devices Revision 2.1 Table 5.

notify_en

Notification enabled.

DEF_ENABLED, CDC notifications are enabled.

DEF_DISABLED, CDC notifications are disabled.

notify_interval

Notification interval in milliseconds. It must be a power of 2.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC

Returned Value

CDC class interface number, if CDC class successfully created.

USBD_CDC_NBR_NONE, otherwise.

Callers

CDC Subclass drivers.

Notes / Warnings

1. The CDC defines a communication class interface consisting of a management element and optionally a notification element. The notification element transports event to the host. The enable_en enable notifications in the CDC. The notification are sent to the host using an interrupt endpoint, the interval of the interrupt endpoint is specified by the notify_interval parameter.

USBD_CDC_CfgAdd

Description

Add a CDC instance to specific USB configuration.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_BOOLEAN  USBD_CDC_CfgAdd (CPU_INT08U   class_nbr,
                              CPU_INT08U   dev_nbr,
                              CPU_INT08U   cfg_nbr,
                              USBD_ERR    *p_err);
```

Arguments

class_nbr

CDC instance number.

dev_nbr

Device number.

cfg_nbr

Configuration number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_INVALID_ARG
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR

```
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_IF_INVALID_NBR
USBD_ERR_IF_GRP_NBR_IN_USE
USBD_ERR_IF_GRP_ALLOC
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

`DEF_OK`, if CDC class instance was added to device configuration successfully.

`DEF_FAIL`, otherwise.

Callers

CDC Subclass drivers.

Notes / Warnings

None.

USBD_CDC_IsConn

Description

Determine if CDC instance is connected.

Files

`usbd_cdc.h/usbd_cdc.c`

Prototype

```
CPU_BOOLEAN  USBD_CDC_IsConn (CPU_INT08U  class_nbr)
```

Arguments

`class_nbr`

CDC instance number.

Returned Value

`DEF_OK`, if CDC instance is connected and device is not in suspended state.

`DEF_FAIL`, otherwise.

Callers

- CDC Subclass drivers
- Application

Notes / Warnings

1. If the `USBD_CDC_IsConn()` returns `DEF_OK`, than the CDC instance is ready for management, notification, read and write operations.

USBD_CDC_DataIF_Add

Description

Add a data interface class to CDC.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_INT08U  USBD_CDC_DataIF_Add (CPU_INT08U    class_nbr,
                                 CPU_BOOLEAN   isoc_en,
                                 CPU_INT08U    protocol,
                                 USBD_ERR     *p_err);
```

Arguments

class_nbr

CDC instance number.

isoc_en

Data interface isochronous enable.

DEF_ENABLED, Data interface uses isochronous endpoints.

DEF_DISABLED, Data interface uses bulk endpoints.

protocol

Data interface protocol code:

Data interface protocol code	Description
USBD_CDC_DATA_PROTOCOL_NONE	No class specific protocol required.
USBD_CDC_DATA_PROTOCOL_NTB	Network Transfer Block.

USBD_CDC_DATA_PROTOCOL_PHY	Physical interface protocol for ISDN BRI.
USBD_CDC_DATA_PROTOCOL_HDLC	HDLC.
USBD_CDC_DATA_PROTOCOL_TRANS	Transparent.
USBD_CDC_DATA_PROTOCOL_Q921M	Management protocol for Q.921 data link protocol.
USBD_CDC_DATA_PROTOCOL_Q921	Data link protocol for Q.921.
USBD_CDC_DATA_PROTOCOL_Q921TM	TEI-multiplexor for Q.921 data link protocol
USBD_CDC_DATA_PROTOCOL_COMPRESS	Data compression procedures.
USBD_CDC_DATA_PROTOCOL_Q9131	Euro-ISDN protocol control.
USBD_CDC_DATA_PROTOCOL_V24	V.24 rate adaptation to ISDN.
USBD_CDC_DATA_PROTOCOL_CAPI	CAPI Commands.
USBD_CDC_DATA_PROTOCOL_HOST	Host based driver.
USBD_CDC_DATA_PROTOCOL_CDC	The protocol(s) are described using a Protocol Unit Function Communication Class Interface.
USBD_CDC_DATA_PROTOCOL_VENDOR	Vendor-specific.

CDC data interface class protocol codes are defined in the Universal Serial Bus Class Definitions for Communication Devices Revision 2.1 Table 7.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_INVALID_ARG
```

Returned Value

Data interface number, if no errors.

`USBD_CDC_DATA_IF_NBR_NONE`, otherwise.

Callers

CDC Subclass drivers.

Notes / Warnings

None.

USBD_CDC_DataRx

Description

Receive data on CDC data interface.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_INT32U  USBD_CDC_DataRx (CPU_INT08U   class_nbr,
                             CPU_INT08U   data_if_nbr,
                             CPU_INT08U  *p_buf,
                             CPU_INT32U   buf_len,
                             CPU_INT16U   timeout,
                             USBD_ERR    *p_err);
```

Arguments

class_nbr

CDC instance number.

data_if_nbr

CDC data interface number.

p_buf

Pointer to destination buffer to receive data.

buf_len

Number of octets to receive.

timeout_ms

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Numbers of octets received, if no errors.

0, otherwise.

Callers

CDC Subclass drivers.

Notes / Warnings

None.

USBD_CDC_DataTx

Description

Send data on CDC data class interface.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_INT32U  USBD_CDC_DataTx (CPU_INT08U   class_nbr,
                             CPU_INT08U   data_if_nbr,
                             CPU_INT08U  *p_buf,
                             CPU_INT32U   buf_len,
                             CPU_INT16U   timeout,
                             USBD_ERR    *p_err);
```

Arguments

class_nbr

CDC instance number.

data_if_nbr

CDC data interface number.

p_buf

Pointer to buffer of data that will be transmitted.

buf_len

Number of octets to transmit.

timeout_ms

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Numbers of octets transmitted, if no errors.

0, otherwise.

Callers

CDC Subclass drivers.

Notes / Warnings

None.

USBD_CDC_Notify

Description

Send communication interface class notification to the host.

Files

usbd_cdc.h/usbd_cdc.c

Prototype

```
CPU_BOOLEAN  USBD_CDC_Notify (CPU_INT08U   class_nbr,
                              CPU_INT08U   notification,
                              CPU_INT16U   value,
                              CPU_INT08U  *p_buf,
                              CPU_INT16U   data_len,
                              USBD_ERR    *p_err);
```

Arguments

class_nbr

CDC instance number.

notification

Notification code (see Note #2).

value

Notification value (see Note #2).

p_buf

Pointer to notification buffer (see Note #1).

data_len

Notification’s data section length.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

None.

Callers

CDC Subclass drivers.

Notes / Warnings

1. The notification buffer size *must* contain space for the notification header (8 bytes) and the variable-length data portion.

2. The following table shows the relationship between CDC request and the parameters passed in the `USBD_CDC_Notify()` function. The bmRequestType and wIndex fields are calculated internally in the CDC module.

bmRequestType	bNotificationCode	wValue	wIndex	wLength	Data
1010001b	notification	value	Interface	data_len	`p_buf[7]` to `p_buf[data_len -1]`

CDC ACM Subclass Functions

- USBD_ACM_SerialInit
- USBD_ACM_SerialAdd
- USBD_ACM_SerialCfgAdd
- USBD_ACM_SerialIsConn
- USBD_ACM_SerialRx
- USBD_ACM_SerialTx
- USBD_ACM_SerialLineCtrlGet
- USBD_ACM_SerialLineCtrlReg
- USBD_ACM_SerialLineCodingGet
- USBD_ACM_SerialLineCodingSet
- USBD_ACM_SerialLineCodingReg
- USBD_ACM_SerialLineStateSet
- USBD_ACM_SerialLineStateClr

USBD_ACM_SerialInit

Description

Initialize CDC ACM serial emulation subclass.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialInit (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function:

USBD_ERR_NONE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialAdd

Description

Add a new CDC ACM serial emulation instance.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
CPU_INT08U  USBD_ACM_SerialAdd (CPU_INT16U   line_state_interval,
                                CPU_INT16U   call_mgmt_capabilities,
                                USBD_ERR    *p_err);
```

Arguments

line_state_interval

Polling interval in frames or microframes for line state notification. The value must be a power of 2.

call_mgmt_capabilities

Call Management Capabilities bitmap. OR'ed of the following flags:

State event	Description
USBD_ACM_SERIAL_CALL_MGMT_DEV	Device handles call management itself
USBD_ACM_SERIAL_CALL_MGMT_DATA_CCI_DCI	Device can send/receive call management information over a Data Class interface

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_INVALID_ARG

Returned Value

CDC ACM serial emulation subclass instance number, if no errors.

`USBD_ACM_SERIAL_NBR_NONE`, otherwise.

Callers

Application.

Notes / Warnings

1. Depending on the operating system (Windows, Linux or Mac OS X), not all the possible flags combinations are supported. This table shows the different combinations and the support by each major operating system (OS):

Flags Combination	Windows	Linux	Mac OS X	Note
`USBD_ACM_SERIAL_CALL_MGMT_DEV`	✓	✓	✗	
`USBD_ACM_SERIAL_CALL_MGMT_DATA_CCI_DCI`	✓	✓	✓	If `USBD_ACM_SERIAL_CALL_MGMT_DEV` is not set, `USBD_ACM_SERIAL_CALL_MGMT_DATA_CCI_DCI` alone is ignored by the OS
`(USBD_ACM_SERIAL_CALL_MGMT_DEV | USBD_ACM_SERIAL_CALL_MGMT_DATA_CCI_DCI)`	✓	✓	✓	

USBD_ACM_SerialCfgAdd

Description

Add CDC ACM subclass instance to USB device configuration.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
CPU_BOOLEAN  USBD_ACM_SerialCfgAdd (CPU_INT08U   subclass_nbr,
                                    CPU_INT08U   dev_nbr,
                                    CPU_INT08U   cfg_nbr,
                                    USBD_ERR    *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

dev_nbr

Device number.

cfg_nbr

Configuration number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_CFG_INVALID_NBR

```
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

`DEF_OK`, If CDC ACM serial emulation subclass instance was added to device configuration successfully.

`DEF_FAIL`, Otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialIsConn

Description

Determine if CDC ACM serial emulation class instance is connected.

Files

`usbd_acm_serial.h/usbd_acm_serial.c`

Prototype

```
CPU_BOOLEAN  USBD_ACM_SerialIsConn (CPU_INT08U  subclass_nbr);
```

Arguments

`subclass_nbr`

CDC ACM serial emulation subclass instance number.

Returned Value

`DEF_OK`, if CDC ACM serial emulation subclass instance is connected and device is not in suspended state.

`DEF_FAIL`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialRx

Description

Receive data on CDC ACM serial emulation subclass.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
CPU_INT32U  USBD_ACM_SerialRx (CPU_INT08U   subclass_nbr,
                               CPU_INT08U  *p_buf,
                               CPU_INT32U   buf_len,
                               CPU_INT16U   timeout,
                               USBD_ERR    *p_err);
```

Arguments

subclass_nbr

Pointer to USB device driver structure.

p_buf

Pointer to destination buffer to receive data.

buf_len

Number of octets to receive.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Numbers of octets received, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialTx

Description

Send data on CDC ACM serial emulation subclass.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
CPU_INT32U  USBD_ACM_SerialTx (CPU_INT08U   subclass_nbr,
                               CPU_INT08U  *p_buf,
                               CPU_INT32U   buf_len,
                               CPU_INT16U   timeout,
                               USBD_ERR    *p_err);
```

Arguments

subclass_nbr

Pointer to USB device driver structure.

p_buf

Pointer to buffer of data that will be transmitted.

buf_len

Number of octets to receive.

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_ABORT
USBD_ERR_OS_FAIL
```

Returned Value

Number of octets transmitted, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialLineCtrlGet

Description

Return current control line state.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
CPU_INT08U  USBD_ACM_SerialLineCtrlGet (CPU_INT08U   subclass_nbr,
                                        USBD_ERR    *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG

Returned Value

Bit-field with the state of the control line.

Bit	Description
USBD_ACM_SERIAL_CTRL_BREAK	Break signal is set.
USBD_ACM_SERIAL_CTRL_RTS	RTS signal is set.
USBD_ACM_SERIAL_CTRL_DTR	DTR signal is set.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialLineCtrlReg

Description

Register line control change notification callback.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineCtrlReg (CPU_INT08U                       subclass_nbr,
                                  USBD_ACM_SERIAL_LINE_CTRL_CHNGD  line_ctrl_chngd,
                                  void                            *p_arg,
                                  USBD_ERR                        *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

line_ctrl_chngd

Line control change notification callback (see note #1).

p_arg

Pointer to callback argument.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG

Returned Value

None.

Callers

Application.

Notes / Warnings

The callback specified by `line_ctrl_chngd` argument is used to notify changes in the control signals to the application. The line control notification function has the following prototype:

```
void AppLineCtrlChngd  (CPU_INT08U   subclass_nbr,
                        CPU_INT08U   events,
                        CPU_INT08U   events_chngd,
                        void        *p_arg);
```

Argument(s)

`subclass_nbr`

CDC ACM serial emulation subclass instance number.

`events`

Current line state.

`events_chngd`

Line state flags that have changed.

`events_chngd`

Pointer to callback argument.

USBD_ACM_SerialLineCodingGet

Description

Get the current state of the line coding.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineCodingGet (CPU_INT08U                   subclass_nbr,
                                    USBD_ACM_SERIAL_LINE_CODING  *p_line_coding,
                                    USBD_ERR                     *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

p_line_coding

Pointer to the structure where the current line coding will be stored.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialLineCodingSet

Description

Set a new line coding.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineCodingSet (CPU_INT08U                   subclass_nbr,
                                    USBD_ACM_SERIAL_LINE_CODING  *p_line_coding,
                                    USBD_ERR                     *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

p_line_coding

Pointer to the structure that contains the new line coding.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialLineCodingReg

Description

Register line coding change notification callback.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineCodingReg(CPU_INT08U                          subclass_nbr,
                                   USBD_ACM_SERIAL_LINE_CODING_CHNGD   line_coding_chngd,
                                   void                               *p_arg,
                                   USBD_ERR                           *p_err);
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

line_coding_chngd

Line coding change notification callback (see Note #1).

p_arg

Pointer to callback argument.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG

Returned Value

None.

Callers

Application.

Notes / Warnings

The callback specified by `line_coding_chngd` argument is used to notify changes in the control signals to the application. The line control notification function has the following prototype:

```
CPU_BOOLEAN  AppLineCodingChngd (CPU_INT08U                   subclass_nbr,
                                 USBD_ACM_SERIAL_LINE_CODING  *p_line_coding,
                                 void                        *p_arg);
```

Arguments

`subclass_nbr`

CDC ACM serial emulation subclass instance number.

`p_line_coding`

Pointer to line coding structure.

`p_arg`

Pointer to callback argument.

Returned value:

`DEF_OK`, If line coding is supported by the application.

`DEF_FAIL`, Otherwise.

USBD_ACM_SerialLineStateSet

Description

Set one or several line state events.

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineStateSet (CPU_INT08U   subclass_nbr,
                                   CPU_INT08U   events,
                                   USBD_ERR    *p_err)
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

events

Line state event(s) to set.

State event	Description
USBD_ACM_SERIAL_STATE_DCD	DCD (Rx carrier)
USBD_ACM_SERIAL_STATE_DSR	DSR (Tx carrier)
USBD_ACM_SERIAL_STATE_BREAK	Break
USBD_ACM_SERIAL_STATE_RING	Ring
USBD_ACM_SERIAL_STATE_FRAMING	Framing error
USBD_ACM_SERIAL_STATE_PARITY	Parity error
USBD_ACM_SERIAL_STATE_OVERUN	Overrun

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_INVALID_CLASS_STATE
```

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_ACM_SerialLineStateClr

Description

Clear one or several line state event(s).

Files

usbd_acm_serial.h/usbd_acm_serial.c

Prototype

```
void  USBD_ACM_SerialLineStateClr (CPU_INT08U   subclass_nbr,
                                   CPU_INT08U   events,
                                   USBD_ERR    *p_err)
```

Arguments

subclass_nbr

CDC ACM serial emulation subclass instance number.

events

Line state event(s) to clear (see Note #1).

State event	Description
USBD_ACM_SERIAL_STATE_DCD	DCD (Rx carrier)
USBD_ACM_SERIAL_STATE_DSR	DSR (Tx carrier)

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_CLASS_INVALID_NBR
USBD_ERR_INVALID_CLASS_STATE

Returned Value

None.

Callers

Application.

Notes / Warnings

Universal Serial Bus Communications Class Subclass Specification for PSTN Devices version 1.2 states: “For the irregular signals like break, the incoming ring signal, or the overrun error state, this will reset their values to zero and again will not send another notification until their state changes”. The irregular events are self-clear and cannot be clear using this function.

API - CDC Ethernet Emulation Model

- USBH_CDC_EEM_Init
- USBH_CDC_EEM_Add
- USBD_CDC_EEM_CfgAdd
- USBD_CDC_EEM_IsConn

USBH_CDC_EEM_Init

Description

Initializes all the internal variables and modules used by the CDC EEM subclass.

Files

usbd_cdc_eem.h/usbd_cdc_eem.c

Prototype

```
void  USBD_CDC_EEM_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

USBD_ERR_OS_SIGNAL_CREATE

Returned Value

None.

Callers

Application.

Notes / Warnings

The initialization function *must* be called only once by the application, and before calling any other CDC EEM API.

USBH_CDC_EEM_Add

Description

Adds a new instance of the CDC EEM class.

Files

usbd_cdc_eem.h/usbd_cdc_eem.c

Prototype

```
CPU_INT08U  USBD_CDC_EEM_Add (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC

Returned Value

Class interface number, if NO error(s).

USBD_CLASS_NBR_NONE, otherwise.

Callers

Application.

Notes / Warnings

None

USBD_CDC_EEM_CfgAdd

Description

Adds CDC EEM subclass instance into USB device configuration.

Files

usbd_cdc_eem.h/usbd_cdc_eem.c

Prototype

```
CPU_BOOLEAN  USBD_CDC_EEM_CfgAdd (       CPU_INT08U   class_nbr,
                                         CPU_INT08U   dev_nbr,
                                         CPU_INT08U   cfg_nbr,
                                   const CPU_CHAR    *p_if_name,
                                         USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

dev_nbr

Device number.

cfg_nbr

Configuration index to add class instance to.

p_if_name

Pointer to string that contains name of the CDC EEM interface. Can be DEF_NULL.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_INVALID_ARG
```

Returned Value

None.

Callers

Application.

Notes / Warnings

This API may be called several times. This allows to create multiple instances of the CDC EEM subclass into different USB device configurations.

USBD_CDC_EEM_IsConn

Description

Returns the CDC EEM subclass connection state.

Files

usbd_cdc_eem.h/usbd_cdc_eem.c

Prototype

```
CPU_BOOLEAN  USBD_CDC_EEM_IsConn (CPU_INT08U  class_nbr)
```

Arguments

class_nbr

Class instance number.

Returned Value

DEF_YES, if class is connected.

DEF_NO, otherwise.

Callers

Application,

Network driver.

Notes / Warnings

The class connected state also implies the USB device is in configured state.

API - Human Interface Device class

- USBD_HID_OS_OutputDataPend
- USBD_HID_OS_OutputDataPendAbort
- USBD_HID_OS_OutputDataPost
- USBD_HID_OS_TxLock
- USBD_HID_OS_TxUnlock
- USBD_HID_OS_TmrTask

HID Class Functions

- USBD_HID_Init
- USBD_HID_Add
- USBD_HID_CfgAdd
- USBD_HID_IsConn
- USBD_HID_Rd
- USBD_HID_RdAsync
- USBD_HID_Wr
- USBD_HID_WrAsync

USBD_HID_Init

Description

Initializes all the internal variables and modules used by the HID class.

Files

`usbd_hid.c`

Prototype

```
void  USBD_HID_Init (USBD_ERR  *p_err);
```

Arguments

`p_err`

Pointer to variable that will receive the return error code from this function.

`USBD_ERR_NONE`

Returned Value

None.

Callers

Application.

Notes / Warnings

The initialization function *must* be called only once by the application, and before calling any other HID API.

USBD_HID_Add

Description

Adds a new instance of the HID class.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
void  USBD_HID_Add (CPU_INT08U              subclass,
                    CPU_INT08U              protocol,
                    USBD_HID_COUNTRY_CODE   country_code,
                    CPU_INT08U             *p_report_desc,
                    CPU_INT16U              report_desc_len,
                    CPU_INT08U             *p_phy_desc,
                    CPU_INT16U              phy_desc_len,
                    CPU_INT16U              interval_in,
                    CPU_INT16U              interval_out,
                    CPU_BOOLEAN             ctrl_rd_en,
                    USBD_HID_CALLBACK      *p_hid_callback,
                    USBD_ERR               *p_err);
```

Arguments

subclass

Subclass code.

protocol

Protocol code.

country_code

Country code ID.

p_report_desc

Pointer to report descriptor structure.

`report_desc_len`

Report descriptor length.

`p_phy_desc`

Pointer to physical descriptor structure.

`phy_desc_len`

Physical descriptor length.

`interval_in`

Polling interval for input transfers, in milliseconds. It must be a power of 2.

`interval_out`

Polling interval for output transfers, in milliseconds. It must be a power of 2. Used only when read operations are not through control transfers.

`ctrl_rd_en`

Enable read operations through control transfers.

`p_hid_callback`

Pointer to HID descriptor and request callback structure. Can be null.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_FAIL
```

Returned Value

Class interface number, if NO error(s).

`USBD_CLASS_NBR_NONE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_HID_CfgAdd

Description

Adds HID class instance into USB device configuration.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
CPU_BOOLEAN  USBD_HID_CfgAdd (CPU_INT08U   class_nbr,
                              CPU_INT08U   dev_nbr,
                              CPU_INT08U   cfg_nbr,
                              USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

dev_nbr

Device number.

cfg_nbr

Configuration index to add class instance to.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE

```
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_ALLOC
```

Returned Value

`DEF_YES`, if NO error(s).

`DEF_NO`, otherwise.

Callers

Application.

Notes / Warnings

This API may be called several times. This allows to create multiple instances of the HID class into different USB device configurations.

USBD_HID_IsConn

Description

Returns the HID class connection state.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
CPU_BOOLEAN  USBD_HID_IsConn (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Returned Value

DEF_YES, if class is connected.

DEF_NO, otherwise.

Callers

Application.

Notes / Warnings

The class connected state also implies the USB device is in configured state.

USBD_HID_Rd

Description

Receives data from the host through an interrupt OUT endpoint.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
CPU_INT32U  USBD_HID_Rd (CPU_INT08U   class_nbr,
                         void        *p_buf,
                         CPU_INT32U   buf_len,
                         CPU_INT16U   timeout,
                         USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length, in octets.

timeout

Timeout, in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets received, if NO error(s).

`0`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_HID_RdAsync

Description

Receives data from the host asynchronously through an interrupt OUT endpoint.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
void  USBD_HID_RdAsync (CPU_INT08U            class_nbr,
                        void                 *p_buf,
                        CPU_INT32U            buf_len,
                        USBD_HID_ASYNC_FNCT   async_fnct,
                        void                 *p_async_arg,
                        USBD_ERR             *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length, in octets.

async_fnct

Receive callback.

p_async_arg

Additional argument provided by application for receive callback.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_FAIL
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

None.

Callers

Application.

Notes / Warnings

This function is non-blocking and returns immediately after transfer preparation. Upon transfer completion, the callback provided is called to notify the application.

USBD_HID_Wr

Description

Transmits data to the host through an interrupt IN endpoint.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
CPU_INT32U  USBD_HID_Wr (CPU_INT08U   class_nbr,
                         void        *p_buf,
                         CPU_INT32U   buf_len,
                         CPU_INT16U   timeout,
                         USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length, in octets.

timeout

Timeout, in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets transmitted, if NO error(s).

`0`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_HID_WrAsync

Description

Transmits data to host asynchronously through an interrupt IN endpoint.

Files

usbd_hid.h/usbd_hid.c

Prototype

```
void  USBD_HID_WrAsync (CPU_INT08U            class_nbr,
                        void                 *p_buf,
                        CPU_INT32U            buf_len,
                        USBD_HID_ASYNC_FNCT   async_fnct,
                        void                 *p_async_arg,
                        USBD_ERR             *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length, in octets.

async_fnct

Transmit callback.

p_async_arg

Additional argument provided by application for transmit callback.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_FAIL
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

None.

Callers

Application.

Notes / Warnings

This function is non-blocking and returns immediately after transfer preparation. Upon transfer completion, the callback provided is called to notify the application.

HID OS Functions

- USBD_HID_OS_Init
- USBD_HID_OS_InputLock
- USBD_HID_OS_InputUnlock
- USBD_HID_OS_InputDataPend
- USBD_HID_OS_InputDataPendAbort
- USBD_HID_OS_InputDataPost
- USBD_HID_OS_OutputLock
- USBD_HID_OS_OutputUnlock
- USBD_HID_OS_OutputDataPend
- USBD_HID_OS_OutputDataPendAbort
- USBD_HID_OS_OutputDataPost
- USBD_HID_OS_TxLock
- USBD_HID_OS_TxUnlock
- USBD_HID_OS_TmrTask

USBD_HID_OS_Init

Description

Initialize HID OS interface.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

OS error code(s) relevant to failure(s).

Callers

HID Class.

Implementation Guidelines

The USBD_HID_Init() function is called only once by the HID class. It usually performs the following operations:

1. For each class instance up to the maximum number of HID class instances defined by the constant USBD_HID_CFG_MAX_NBR_DEV, create all the required semaphores. If the any semaphore creation fails, set p_err to USBD_ERR_OS_SIGNAL_CREATE and return.

2. Create a task used to manage periodic Input reports. If the task creation fails, set p_err

to USBD_ERR_OS_INIT_FAIL and return.

3. Set p_err to USBD_ERR_NONE if the initialization proceeded as expected.

USBD_HID_OS_InputLock

Description

Lock class input report.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_InputLock (CPU_INT08U   class_nbr,
                             USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

OS error code(s) relevant to failure(s).

Callers

HID Class.

Implementation Guidelines

The lock operation typically consists in pending on a semaphore. If the semaphore is free, the task continues normally its execution, otherwise it waits until another task releases the semaphore. p_err argument should be assigned as described in following table:

Operation result	Error code to assign
No error	USBD_ERR_NONE
Pend aborted	USBD_ERR_OS_ABORT
Pend failed for any other reason	USBD_ERR_OS_FAIL

Table - Error code assignment according to pend operation result

USBD_HID_OS_InputUnlock

Description

Unlock class input report.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_InputUnlock (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Callers

HID Class.

Implementation Guidelines

The unlock operation generally consists in posting a mutex/semaphore.

USBD_HID_OS_InputDataPend

Description

Wait for input report data to complete.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_InputDataPend (CPU_INT08U   class_nbr
                                 CPU_INT16U   timeout_ms,
                                 USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

timeout_ms

Signal wait timeout in milliseconds

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

OS error code(s) relevant to failure(s)

Callers

HID Class.

Implementation Guidelines

1. The wait operation typically consists in pending on a semaphore. When the input report transfer has completed, the task is waken up by the Core layer internal task responsible for asynchronous communication. `p_err` argument should be assigned as described in Table - Error code assignment according to pend operation result in the *USBD_HID_OS_InputLock* page.

USBD_HID_OS_InputDataPendAbort

Description

Abort any operation on input report.

Files

`usbd_hid_os.h/usbd_hid_os.c`

Prototype

```
void  USBD_HID_OS_InputDataPendAbort (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

Class instance number.

Callers

HID Class.

Implementation Guidelines

If the input report transfer completes with an error, the task waiting is waken up by aborting the active wait done with `USBD_HID_OS_InputDataPend`. The active wait abortion is executed by the Core layer internal task responsible for asynchronous communication.

USBD_HID_OS_InputDataPost

Description

Signal that Input report data has been sent to the host.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_InputDataPost (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Callers

HID Class.

Implementation Guidelines

If the input report transfer completes without an error, the task waiting is waken up by posting a semaphore. The semaphore post is executedby the Core layer internal task responsible for asynchronous communication.

USBD_HID_OS_OutputLock

Description

Lock class output report.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_OutputLock (CPU_INT08U   class_nbr,
                              USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE: OS error code(s) relevant to failure(s)

Callers

HID Class.

Implementation Guidelines

The lock operation typically consists in pending on a semaphore. If the semaphore is free, the task continues normally its execution, otherwise it waits until another task releases the semaphore. p_err argument should be assigned as described in Table - Error code assignment according to pend operation result in the *USBD_HID_OS_InputLock* page.

USBD_HID_OS_OutputUnlock

Description

Unlock class output report.

Files

`usbd_hid_os.h/usbd_hid_os.c`

Prototype

```
void  USBD_HID_OS_OutputUnlock (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

Class instance number.

Callers

HID Class.

Implementation Guidelines

The unlock operation generally consists in posting a mutex/semaphore.

USBD_HID_OS_OutputDataPend

Description

Wait for Output report data read completion.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_OutputDataPend (CPU_INT08U   class_nbr
                                  CPU_INT16U   timeout_ms,
                                  USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

timeout_ms

Signal wait timeout in milliseconds

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

OS error code(s) relevant to failure(s)

Callers

HID Class.

Implementation Guidelines

The wait operation typically consists of pending on a semaphore. When the output report transfer is complete, the task is woken up by the Core layer internal task responsible for asynchronous communication. The `p_err` argument should be assigned as described in Table - Error code assignment according to pend operation result in the *USBD_HID_OS_InputLock* page

USBD_HID_OS_OutputDataPendAbort

Description

Abort the wait for Output report data read completion.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_OutputDataPendAbort (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Callers

HID Class.

Implementation Guidelines

If the output report transfer completes with an error, the task waiting is waken up by aborting the active wait done with USBD_HID_OS_OutputDataPend. The active wait abortion is executed by the Core layer internal task responsible for asynchronous communication.

USBD_HID_OS_OutputDataPost

Description

Signal that Output report data has been received from the host

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_OutputDataPost (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Callers

HID Class.

Implementation Guidelines

If the output report transfer completes without an error, the task waiting is waken up by posting a semaphore. The semaphore post is executedby the Core layer internal task responsible for asynchronous communication.

USBD_HID_OS_TxLock

Description

Lock class transmit.

Files

`usbd_hid_os.h/usbd_hid_os.c`

Prototype

```
void  USBD_HID_OS_TxLock (CPU_INT08U   class_nbr,
                          USBD_ERR    *p_err);
```

Arguments

`class_nbr`

Class instance number.

`p_err`

Pointer to variable that will receive the return error code from this function.

`USBD_ERR_NONE`: OS error code(s) relevant to failure(s).

Callers

HID Class.

Implementation Guidelines

The lock operation typically consists in pending on a semaphore. If the semaphore is free, the task continues normally its execution, otherwise it waits until another task releases the semaphore. `p_err` argument should be assigned as described in Table - Error code assignment according to pend operation result in the *USBD_HID_OS_InputLock* page.

USBD_HID_OS_TxUnlock

Description

Unlock class transmit.

Files

usbd_hid_os.h/usbd_hid_os.c

Prototype

```
void  USBD_HID_OS_TxUnlock (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Callers

HID Class.

Implementation Guidelines

The unlock operation generally consists in posting a mutex/semaphore.

USBD_HID_OS_TmrTask

Description

Process periodic input reports according to idle duration set by the host with the SET_IDLE request.

Files

`usbd_hid_os.h/usbd_hid_os.c`

Prototype

```
static  void  USBD_HID_OS_TmrTask (void  *p_arg);
```

Arguments

`p_arg`

Pointer to task initialization argument.

Callers

This is a task.

Implementation Guidelines

The task body is usually implemented as an infinite loop. The task should perform the following steps:

1. Delay for 4 ms. This delay corresponds to the 4 ms unit used to express the idle duration transported by the `SET_IDLE` request.

2. Call `USBD_HID_Report_TmrTaskHandler()` function defined in the HID parser module. This function implements the periodic input reports processing.

API - Mass Storage Class

- USBD_StorageAdd
- USBD_StorageCapacityGet
- USBD_StorageRd
- USBD_StorageWr
- USBD_StorageStatusGet
- USBD_StorageLock
- USBD_StorageUnlock
- USBD_StorageRefreshTaskHandler

MSC Functions

- USBD_MSC_Init
- USBD_MSC_Add
- USBD_MSC_CfgAdd
- USBD_MSC_LunAdd
- USBD_MSC_IsConn
- USBD_MSC_TaskHandler

USBD_MSC_Init

Description

Initialize internal structures and local global variables used by the MSC bulk only transport.

Files

usbd_msc.h / usbd_msc.c

Prototype

```
void  USBD_MSC_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function:
USBD_ERR_NONE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_MSC_Add

Description

Create a new instance of the MSC.

Files

usbd_msc.h / usbd_msc.c

Prototype

```
CPU_INT08U  USBD_MSC_Add (USBD_ERR  *p_err)
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC

Returned Value

Class instance number, if NO error(s).

USBD_CLASS_NBR_NONE, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_MSC_CfgAdd

Description

Add an existing MSC instance to the specified configuration and device. The MSC instance was previously created by the function `USBD_MSC_Add()`.

Files

`usbd_msc.h / usbd_msc.c`

Prototype

```
CPU_BOOLEAN  USBD_MSC_CfgAdd (CPU_INT08U   class_nbr,
                              CPU_INT08U   dev_nbr,
                              CPU_INT08U   cfg_nbr,
                              USBD_ERR    *p_err);
```

Arguments

`class_nbr`

MSC instance number.

`dev_nbr`

Device number.

`cfg_nbr`

Configuration index to add MSC instance to.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
USBD_ERR_NULL_PTR
```

```
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

DEF_YES, if MSC instance is added to USB device configuration successfully.

DEF_NO, otherwise.

Callers

Application.

Notes / Warnings

`USBD_MSC_CfgAdd()` basically adds an Interface descriptor and its associated Endpoint descriptor(s) to the Configuration descriptor. One call to `USBD_MSC_CfgAdd()` builds the Configuration descriptor corresponding to a MSC device with the following format:

```
Configuration Descriptor
|-- Interface Descriptor (MSC)
|-- Endpoint Descriptor (Bulk OUT)
|-- Endpoint Descriptor (Bulk IN)
```

If `USBD_MSC_CfgAdd()` is called several times from the application, it allows to create multiple instances and multiple configurations. For instance, the following architecture could be created for an high-speed device:

```
High-speed
|-- Configuration 0
|-- Interface 0 (MSC 0)
|-- Configuration 1
|-- Interface 0 (MSC 0)
|-- Interface 1 (MSC 1)
```

In that example, there are two instances of MSC: 'MSC 0' and 'MSC 1', and two possible configurations for the device: 'Configuration 0' and 'Configuration 1'. 'Configuration 1' is composed of two interfaces. Each class instance has an association with one of the interfaces. If 'Configuration 1' is activated by the host, it allows the host to access two different functionalities offered by the device.

USBD_MSC_LunAdd

Description

Add a logical unit number to the MSC interface.

Files

usbd_msc.h / usbd_msc.c

Prototype

```
void  USBD_MSC_LunAdd (CPU_CHAR     *p_store_name,
                       CPU_INT08U    class_nbr,
                       CPU_CHAR     *p_vend_id,
                       CPU_CHAR     *p_prod_id,
                       CPU_INT32U    prod_rev_level,
                       CPU_BOOLEAN   rd_only,
                       USBD_ERR     *p_err);
```

Arguments

p_store_name

Pointer to logical unit driver.

class_nbr

MSC instance number.

p_vend_id

Pointer to string containing vendor id.

p_prod_id

Pointer to string containing product id.

prod_rev_level

Product revision level.

`rd_only`

Boolean specifying if logical unit is read only or not.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_MSC_MAX_LUN_EXCEED
USBD_ERR_SCSI_LOG_UNIT_NOTRDY
```

Returned Value

None.

Callers

Application.

Notes / Warnings

The pointer to logical unit driver specifies the type and volume of the logical unit to add. Valid logical unit driver names follow the pattern: `<device_driver_name>:<logical_unit_number>:` where `<device_driver_name>` is the name of the device driver and `<logical_unit_number>` is the device's logical unit number. Take special note that the logical unit number starts counting from number 0.

USBD_MSC_IsConn

Description

Get MSC connection state of the device.

Files

`usbd_msc.h / usbd_msc.c`

Prototype

```
CPU_BOOLEAN  USBD_MSC_IsConn (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

MSC instance number.

Returned Value

`DEF_YES`, if MSC is connected.
`DEF_NO`, otherwise.

Callers

Application.

Notes / Warnings

`USBD_MSC_IsConn()` is typically used to verify that the device is in ‘configured’ state and that the MSC instance is ready for communication. The following code illustrates a typical example:

```
CPU_BOOLEAN  conn;

conn = USBD_MSC_IsConn(class_nbr);
if (conn != DEF_YES) {
```

```
    USBD_MSC_OS_EnumSignalPend((CPU_INT16U)0,
                                           &os_err);
}
```

Once the connected status is DEF_YES, the communication can start.

USBD_MSC_TaskHandler

Description

Task to handle transfers for the MSC bulk-only transport protocol.

Files

usbd_msc.h / usbd_msc.c

Prototype

```
void  USBD_MSC_TaskHandler (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

MSC instance number.

Returned Value

None.

Callers

OS layer.

Notes / Warnings

None.

MSC OS Functions

- USBD_MSC_OS_Init
- USBD_MSC_OS_CommSignalPost
- USBD_MSC_OS_CommSignalPend
- USBD_MSC_OS_CommSignalDel
- USBD_MSC_OS_EnumSignalPost
- USBD_MSC_OS_EnumSignalPend
- USBD_MSC_OS_Task
- USBD_MSC_OS_RefreshTask

USBD_MSC_OS_Init

Description

Initialize MSC OS interface.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_Init (USBD_ERR  *p_err)
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

Initialization of the MSC OS interface must include creating:

1. Two semaphores, one for MSC communication and one for enumeration.
2. A MSC task to handle the MSC protocol.

3. A Refresh task if µC/FS storage layer is used with removable media.

USBD_MSC_OS_CommSignalPost

Description

Post a semaphore used for MSC communication.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_CommSignalPost (CPU_INT08U   class_nbr,
                                  USBD_ERR    *p_err)
```

Arguments

class_nbr

MSC instance class number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

This function will generally post a counting semaphore.

USBD_MSC_OS_CommSignalPend

Description

Wait on a semaphore to become available for MSC communication.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_CommSignalPend (CPU_INT08U   class_nbr,
                                  CPU_INT32U   timeout,
                                  USBD_ERR    *p_err);
```

Arguments

class_nbr

MSC instance class number.

timeout

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

The lock operation typically consists in pending on a semaphore. If the semaphore is free, the task continues normally its execution, otherwise it waits until another task releases the semaphore. `p_err` argument should be assigned as described in following table.

Operation result	Error code to assign
No error	`USBD_ERR_NONE`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error code assignment according to pend operation result

USBD_MSC_OS_CommSignalDel

Description

Delete a semaphore if no tasks are waiting on it for MSC communication.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_CommSignalDel (CPU_INT08U   class_nbr,
                                 USBD_ERR    *p_err);
```

Arguments

class_nbr

MSC instance class number.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

None.

USBD_MSC_OS_EnumSignalPost

Description

Post a semaphore for MSC enumeration process.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_EnumSignalPost (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

This function generally posts a counting semaphore.

USBD_MSC_OS_EnumSignalPend

Description

Wait on a semaphore to become available for MSC enumeration process.

Files

usbd_msc_os.h / usbd_msc_os.c

Prototype

```
void  USBD_MSC_OS_EnumSignalPend (CPU_INT32U   timeout,
                                  USBD_ERR    *p_err);
```

Arguments

timeout

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_OS_TIMEOUT
USBD_ERR_OS_FAIL

Returned Value

None.

Callers

Mass Storage Class.

Implementation Guidelines

None.

USBD_MSC_OS_Task

Description

Process the MSC protocol.

Files

`usbd_msc_os.c`

Prototype

```
static  void  USBD_MSC_OS_Task (void  *p_arg);
```

Arguments

`p_arg`

Pointer to task initialization argument.

Callers

This is a task.

Implementation Guidelines

The task should call `USBD_MSC_TaskHandler()` function defined in the MSC layer. This function implements the task body, usually an infinite loop, responsible for the MSC protocol management.

USBD_MSC_OS_RefreshTask

Description

Detect the insertion or removal of removable media. Defined only if the µC/FS storage layer is used.

Files

`usbd_msc_os.c`

Prototype

```
static  void  USBD_MSC_OS_RefreshTask (void  *p_arg);
```

Arguments

`p_arg`

Pointer to task initialization argument.

Callers

This is a task.

Implementation Guidelines

The task body is usually implemented as an infinite loop. The task should perform the following steps:

1. Call `USBD_StorageRefreshTaskHandler()` function defined in the µC/FS storage layer. This function implements the removable media insertion/removable detection..

2. Delay for a certain period of time. This delay is configurable by you through the configuration constant, `USBD_MSC_CFG_DEV_POLL_DLY_mS`. Refer to MSC General Configuration for more details about this constant.

MSC Storage Layer Functions

- USBD_StorageInit
- USBD_StorageAdd
- USBD_StorageCapacityGet
- USBD_StorageRd
- USBD_StorageWr
- USBD_StorageStatusGet
- USBD_StorageLock
- USBD_StorageUnlock
- USBD_StorageRefreshTaskHandler

USBD_StorageInit

Description

Initialize internal structures and local global variables used by the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageInit (USBD_ERR  *p_err)
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageAdd

Description

Initialize the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageAdd (USBD_STORAGE_LUN  *p_storage_lun)
                       USBD_ERR          *p_err)
```

Arguments

p_storage_lun

Pointer to logical unit storage structure.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_SCSI_LU_NOTRDY

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageCapacityGet

Description

Get the capacity of the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageCapacityGet (USBD_STORAGE_LUN  *p_storage_lun),
                               CPU_INT64U        *p_nbr_blks,
                               CPU_INT32U        *p_blk_size,
                               USBD_ERR          *p_err)
```

Arguments

p_storage_lun

Pointer to logical unit storage structure.

p_nbr_blks

Pointer to variable that will receive the number of logical blocks.

p_blk_size

Pointer to variable that will receive the size of each block, in bytes.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_SCSI_MEDIUM_NOTPRESENT

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageRd

Description

Read data from the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageRd (USBD_STORAGE_LUN  *p_storage_lun,
                      CPU_INT32U         blk_addr,
                      CPU_INT32U         nbr_blks,
                      CPU_INT08U        *p_data_buf,
                      USBD_ERR          *p_err);
```

Arguments

p_storage_lun

Pointer to the logical unit storage structure.

blk_addr

Logical Block Address (LBA) of read block start.

nbr_blks

Number of logical blocks to read.

p_data_buf

Pointer to buffer in which data will be stored.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_SCSI_MEDIUM_NOT_PRESENT
```

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageWr

Description

Write data to the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageWr (USBD_STORAGE_LUN  *p_storage_lun,
                      CPU_INT32U         blk_addr,
                      CPU_INT32U         nbr_blks,
                      CPU_INT08U        *p_data_buf,
                      USBD_ERR          *p_err);
```

Arguments

p_storage_lun

Pointer to logical unit storage structure

blk_addr

Logical Block Address (LBA) of write block start.

nbr_blks

Number of logical blocks to write.

p_data_buf

Pointer to buffer in which data is stored.

p_err

Pointer to variable that receives the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_SCSI_MEDIUM_NOTPRESENT
```

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageStatusGet

Description

Get the presence of the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageStatusGet (USBD_STORAGE_LUN  *p_storage_lun,
                             USBD_ERR          *p_err);
```

Arguments

p_storage_lun

Pointer to logical unit storage structure

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_SCSI_MEDIUM_NOTPRESENT
USBD_ERR_SCSI_MEDIUM_NOT_RDY_TO_RDY
USBD_ERR_SCSI_MEDIUM_RDY_TO_NOT_RDY

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageLock

Description

Lock the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageLock (USBD_STORAGE_LUN  *p_storage_lun,
                        CPU_INT32U         timeout_ms,
                        USBD_ERR          *p_err)
```

Arguments

p_storage_lun

Pointer to logical unit storage structure

timeout_ms

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_SCSI_LOCK_TIMEOUT
USBD_ERR_SCSI_LOCK

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageUnlock

Description

Unlock the storage medium.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageLock (USBD_STORAGE_LUN  *p_storage_lun,
                        USBD_ERR          *p_err)
```

Arguments

p_storage_lun

Pointer to logical unit storage structure

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_SCSI_UNLOCK

Returned Value

None.

Callers

Mass Storage Class.

Notes / Warnings

None.

USBD_StorageRefreshTaskHandler

Description

Check the removable media presence status, that is insertion/removal detection. Defined only for the µC/FS storage layer.

Files

usbd_storage.h / usbd_storage.c

Prototype

```
void  USBD_StorageRefreshTaskHandler (void  *p_arg)
```

Arguments

p_arg

Pointer to task initialization argument.

Returned Value

None.

Callers

USBD_MSC_OS_Refresh_Task()

Notes / Warnings

None.

API - Personal Healthcare Device Class

- USBD_PHDC_OS_WrIntrUnLock
- USBD_PHDC_OS_WrBulkLock
- USBD_PHDC_OS_WrBulkUnlock

PHDC Functions

- USBD_PHDC_Init
- USBD_PHDC_Add
- USBD_PHDC_CfgAdd
- USBD_PHDC_IsConn
- USBD_PHDC_RdCfg
- USBD_PHDC_WrCfg
- USBD_PHDC_11073_ExtCfg
- USBD_PHDC_PreambleRd
- USBD_PHDC_Rd
- USBD_PHDC_PreambleWr
- USBD_PHDC_Wr
- USBD_PHDC_Reset

USBD_PHDC_Init

Description

Initialize internal structures and local global variables used by the PHDC.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_PHDC_Add

Description

Create a new instance of the PHDC.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
CPU_INT08U  USBD_PHDC_Add (CPU_BOOLEAN                   data_fmt_11073,
                           CPU_BOOLEAN                   preamble_capable,
                           USBD_PHDC_PREAMBLE_EN_NOTIFY  preamble_en_notify,
                           CPU_INT16U                    low_latency_interval,
                           USBD_ERR                     *p_err)
```

Arguments

data_fmt_11073

Variable that indicates whether the class instance uses IEEE 11073 or a vendor-defined data format.

DEF_YES

Class instance uses IEEE 11073 data format.

DEF_NO

Class instance uses vendor-defined data format.

preamble_capable

Variable that indicates whether the class instance support metadata message preamble or not.

DEF_YES

Class instance support metadata message preamble.

DEF_NO

Class instance doesn’t support metadata message preamble.

preamble_en_notify

Pointer to a callback function that will notify the application if the host enable / disable metadata message preamble.

low_latency_interval

Interrupt endpoint interval in milliseconds. Can be 0 if PHDC device will not send low latency data. When the value is different from 0, it must be a power of 2.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_ALLOC

Returned Value

Class instance number, if NO error(s).

USBD_CLASS_NBR_NONE, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_PHDC_CfgAdd

Description

Add a PHDC instance into the specified configuration. The PHDC instance was previously created by the function USBD_PHDC_Add.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_CfgAdd (CPU_INT08U   class_nbr,
                        CPU_INT08U   dev_nbr,
                        CPU_INT08U   cfg_nbr,
                        USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

dev_nbr

Device number.

cfg_nbr

Configuration index to add PHDC instance to.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
USBD_ERR_NULL_PTR

```
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_CfgAdd()` basically adds an Interface descriptor and its associated Endpoint descriptor(s) to the Configuration descriptor. One call to `USBD_PHDC_CfgAdd()` builds the Configuration descriptor corresponding to a PHDC device with the following format:

```
Configuration Descriptor
|-- Interface Descriptor (PHDC)
    |-- Endpoint Descriptor (Bulk OUT)
    |-- Endpoint Descriptor (Bulk IN)
    |-- Endpoint Descriptor (Interrupt IN) - optional
```

2. The Interrupt IN endpoint is optional. It will be added to the Interface descriptor if application specified that it will send low latency data when calling `USBD_PHDC_WrCfg()`.

3. If `USBD_PHDC_CfgAdd()` is called several times from the application, it allows to create multiple instances and multiple configurations. For instance, the following architecture could be created for an high-speed device:

```
High-speed
|-- Configuration 0
    |-- Interface 0 (PHDC 0)
|-- Configuration 1
    |-- Interface 0 (PHDC 0)
    |-- Interface 1 (PHDC 1)
```

   In that example, there are two instances of PHDC: 'PHDC 0' and 'PHDC 1', and two possible configurations for the device: 'Configuration 0' and 'Configuration 1'. 'Configuration 1' is composed of two interfaces. Each class instance has an association with one of the interfaces. If 'Configuration 1' is activated by the host, it allows the host to access two different functionalities offered by the device.

USBD_PHDC_IsConn

Description

Get PHDC connection state.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
CPU_BOOLEAN  USBD_PHDC_IsConn (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

PHDC instance number.

Returned Value

DEF_YES, if PHDC is connected.

DEF_NO, otherwise.

Callers

Application.

Notes / Warnings

USBD_PHDC_IsConn() is typically used to verify that the device is in ‘configured’ state and that the PHDC instance is ready for communication. The following code illustrates a typical example:

```
CPU_BOOLEAN  conn;

conn = USBD_PHDC_IsConn(class_nbr);
```

```
while (conn != DEF_YES) {
    OSTimeDlyHMSM(0, 0, 0, 250);

    conn = USBD_PHDC_IsConn(class_nbr);
}
```

Once the connected status is `DEF_YES`, the communication can start.

USBD_PHDC_RdCfg

Description

Initialize read communication pipe parameters.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_RdCfg (CPU_INT08U           class_nbr,
                       LATENCY_RELY_FLAGS   latency_rely,
                       CPU_INT08U          *p_data_opaque,
                       CPU_INT08U           data_opaque_len,
                       USBD_ERR            *p_err);
```

Arguments

class_nbr

PHDC instance number.

latency_rely

Bitmap of transfer latency / reliability that this communication pipe will carry. Can be one or more of these values:

```
USBD_PHDC_LATENCY_VERYHIGH_RELY_BEST
USBD_PHDC_LATENCY_HIGH_RELY_BEST
USBD_PHDC_LATENCY_MEDIUM_RELY_BEST
```

p_data_opaque

Pointer to a buffer that contains opaque data related to this communication pipe.

data_opaque_len

Length of opaque data (in octets). If 0, no metadata descriptor will be written for the

endpoint.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
```

Returned Value

None.

Callers

Application.

Notes / Warnings

`USBD_PHDC_RdCfg()` should be called after `USBD_PHDC_Init()` and `USBD_PHDC_Add` but before `USBD_PHDC_CfgAdd`.

USBD_PHDC_WrCfg

Description

Initialize write communication pipe parameters.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_WrCfg (CPU_INT08U           class_nbr,
                       LATENCY_RELY_FLAGS   latency_rely,
                       CPU_INT08U          *p_data_opaque,
                       CPU_INT08U           data_opaque_len,
                       USBD_ERR            *p_err);
```

Arguments

class_nbr

PHDC instance number.

latency_rely

Bitmap of transfer Latency / reliability that this communication pipe will carry. Can be one or more of these values:

```
USBD_PHDC_LATENCY_VERYHIGH_RELY_BEST
USBD_PHDC_LATENCY_HIGH_RELY_BEST
USBD_PHDC_LATENCY_MEDIUM_RELY_BEST
USBD_PHDC_LATENCY_MEDIUM_RELY_BETTER
USBD_PHDC_LATENCY_MEDIUM_RELY_GOOD
USBD_PHDC_LATENCY_LOW_RELY_GOOD
```

p_data_opaque

Pointer to a buffer that contains opaque data related to this communication pipe.

`data_opaque_len`

Length of opaque data (in octets). If 0, no metadata descriptor will be written for the endpoint.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_WrCfg()` should be called after `USBD_PHDC_Init` and `USBD_PHDC_Add` but before `USBD_PHDC_CfgAdd()`.

2. Since low latency transfers will use a different endpoint, it is possible to set different opaque data for that endpoint. In case the application need different opaque data for low latency pipe, `USBD_PHDC_WrCfg()` should be called twice. Once with all the desired latency/reliability flags set except for low latency, opaque data passed at this call will be used for the Bulk endpoint metadata descriptor. `USBD_PHDC_WrCfg()` should then be called once again with only the low latency flag set, opaque data passed at this call will be used for interrupt endpoint metadata descriptor.

USBD_PHDC_11073_ExtCfg

Description

Configure function extension for given class instance.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_11073_ExtCfg (CPU_INT08U   class_nbr,
                              CPU_INT16U  *p_dev_specialization,
                              CPU_INT08U   nbr_dev_specialization,
                              USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

p_dev_specialization

Pointer to an array that contains a list of device specializations.

nbr_dev_specialization

Number of device specializations specified in p_dev_specialization.

p_err

Pointer to variable that will receive the return error code from this function.

USBD_ERR_NONE
USBD_ERR_INVALID_ARG

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_11073_ExtCfg()` should be called only if PHDC instance uses 11073 data format.

2. `USBD_PHDC_11073_ExtCfg()` should be called after `USBD_PHDC_Init` and `USBD_PHDC_Add()` but before `USBD_PHDC_CfgAdd`.

3. For more information on 11073 device specialization, See 'Personal Healthcare Device Class specifications Revision 1.0', Appendix A. For a list of known device specialization, see 'Nomenclature code annex of ISO/IEEE 11073-20601'. Specific code are listed in the 'From Communication infrastructure (MDC_PART_INFRA)' section.

USBD_PHDC_PreambleRd

Description

Read metadata preamble. This function is blocking.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
CPU_INT08U  USBD_PHDC_PreambleRd (CPU_INT08U   class_nbr,
                                  void        *p_buf,
                                  CPU_INT08U   buf_len,
                                  CPU_INT08U  *p_nbr_xfer,
                                  CPU_INT16U   timeout,
                                  USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

p_buf

Pointer to buffer that will contain data from metadata message preamble.

buf_len

Opaque data buffer length in octets.

p_nbr_xfer

Pointer to a variable that will contain the number of transfer the preamble will apply to. After this call, USBD_PHDC_Rd shall be called nbr_xfer times by the application.

timeout

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_ALLOC
USBD_ERR_RX
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_OS_ERR_TIMEOUT
USBD_OS_ERR_ABORT
USBD_OS_ERR_FAIL
```

Returned Value

Length of opaque data read from metadata preamble, if no error.

0, otherwise

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_PreambleRd()` should always be called before `USBD_PHDC_Rd` if metadata message preambles are enabled by the host. Application should then call `USBD_PHDC_Rd` `p_nbr_xfer` times.

2. If host disable preamble while application is pending on this function, the call will immediately return with error '`USBD_OS_ERR_ABORT`'.

USBD_PHDC_Rd

Description

Read PHDC data. This function is blocking.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
CPU_INT08U  USBD_PHDC_Rd (CPU_INT08U   class_nbr,
                          void        *p_buf,
                          CPU_INT16U   buf_len,
                          CPU_INT16U   timeout,
                          USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

p_buf

Pointer to buffer that will contain opaque data from metadata message preamble.

buf_len

Opaque data buffer length in octets.

timeout

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_RX
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_OS_ERR_TIMEOUT
USBD_OS_ERR_ABORT
USBD_OS_ERR_FAIL
```

Returned Value

Number of octets received, if no error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_Rd()` should always be called after `USBD_PHDC_PreambleRd()` if metadata message preambles are enabled by the host.

2. Application should ensure that the length of the buffer provided is large enough to accommodate the incoming transfer. Otherwise, synchronization with metadata preambles might be lost.

3. If host enable preamble while application is pending on this function, the call will immediately return with error '`USBD_OS_ERR_ABORT`'.

USBD_PHDC_PreambleWr

Description

Write metadata preamble. This function is blocking.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_PreambleWr (CPU_INT08U           class_nbr,
                            void                *p_data_opaque,
                            CPU_INT16U           data_opaque_len,
                            LATENCY_RELY_FLAGS   latency_rely
                            CPU_INT08U           nbr_xfers,
                            CPU_INT16U           timeout,
                            USBD_ERR            *p_err);
```

Arguments

class_nbr

PHDC instance number.

p_data_opaque

Pointer to buffer that will supply opaque data.

data_opaque_len

Length of opaque data buffer in octets.

latency_rely

Latency reliability of related transfers.

nbr_xfers

Number of transfers this preamble will apply to.

`timeout`

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_TX
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_OS_ERR_TIMEOUT
USBD_OS_ERR_ABORT
USBD_OS_ERR_FAIL
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_PreambleWr()` should always be called before `USBD_PHDC_Wr()` if metadata message preambles are enabled by the host and if the latency of the transfer is not ‘low’.

2. Application will have to call `USBD_PHDC_Wr()` ‘`nbr_xfers`’ of times with the same latency / reliability parameter after a call to `USBD_PHDC_PreambleWr()`.

USBD_PHDC_Wr

Description

Write PHDC data. This function is blocking.

Files

usbd_phdc.h / usbd_phdc.c

Prototype

```
void  USBD_PHDC_Wr (CPU_INT08U           class_nbr,
                    void                *p_buf,
                    CPU_INT16U           buf_len,
                    LATENCY_RELY_FLAGS   latency_rely
                    CPU_INT16U           timeout,
                    USBD_ERR            *p_err);
```

Arguments

class_nbr

PHDC instance number.

p_buf

Pointer to buffer that will supply data.

buf_len

Buffer length in octets.

latency_rely

Latency / reliability of this transfer.

timeout

Timeout in milliseconds.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
USBD_ERR_TX
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_ADDR
USBD_ERR_EP_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_OS_ERR_TIMEOUT
USBD_OS_ERR_ABORT
USBD_OS_ERR_FAIL
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_PHDC_Wr()` should always be called after `USBD_PHDC_PreambleWr()` if metadata message preambles are enabled by the host and if the latency of the transfer is not ‘low’.

2. Application will have to call `USBD_PHDC_Wr()` ‘`nbr_xfers`’of times with the same latency / reliability parameter after a call to `USBD_PHDC_PreambleWr`.

USBD_PHDC_Reset

Description

Reset PHDC instance.

Files

`usbd_phdc.h / usbd_phdc.c`

Prototype

```
void  USBD_PHDC_Reset (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

PHDC instance number.

Returned Value

None.

Callers

`USBD_PHDC_Disconn()`

Application.

Notes / Warnings

1. `USBD_PHDC_Reset()` should be used to reset internal variables like the transmit priority queue of the PHDC instance.

2. This function should be called when the data layer above PHDC request to terminate communication. For instance, `USBD_PHDC_Reset()` should be called when the host send an ‘11073 Association abort’ request.

PHDC OS Functions

- USBD_PHDC_OS_Init
- USBD_PHDC_OS_RdLock
- USBD_PHDC_OS_RdUnLock
- USBD_PHDC_OS_WrIntrLock
- USBD_PHDC_OS_WrIntrUnLock
- USBD_PHDC_OS_WrBulkLock
- USBD_PHDC_OS_WrBulkUnlock

USBD_PHDC_OS_Init

Description

Initialize PHDC OS layer.

Files

usbd_phdc_os.h / usbd_phdc_os.c

Prototype

```
void  USBD_PHDC_OS_Init (USBD_ERR  *p_err);
```

Arguments

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

1. This function should be used to initialize all RTOS layer’s internal variables / tasks of every class instances. It will be called only once.

2. In case creation of semaphore, mutex, or other signal fails, the function should assign USBD_ERR_OS_SIGNAL_CREATE to p_err and return immediately. If any other error occurs, USBD_ERR_OS_INIT_FAIL should be assigned to p_err. Otherwise, USBD_ERR_NONE should be used.

USBD_PHDC_OS_RdLock

Description

Lock the read pipe.

Files

usbd_phdc_os.h / usbd_phdc_os.c

Prototype

```
void  USBD_PHDC_OS_RdLock (CPU_INT08U   class_nbr,
                           CPU_INT16U   timeout,
                           USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

timeout

Timeout.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

Typical implementation will consist in pending on a semaphore that locks the read pipe. `p_err` argument should be assigned as described in following table.

Operation result	Error code to assign
No error	`USBD_ERR_NONE`
Pend timeout	`USBD_ERR_OS_TIMEOUT`
Pend aborted	`USBD_ERR_OS_ABORT`
Pend failed for any other reason	`USBD_ERR_OS_FAIL`

Table - Error Code Assignment in Function of Lock Operation Result

USBD_PHDC_OS_RdUnLock

Description

Unlock the read pipe.

Files

`usbd_phdc_os.h / usbd_phdc_os.c`

Prototype

```
void  USBD_PHDC_OS_RdUnlock (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

PHDC instance number.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

Typical implementation will consist in posting a mutex/semaphore that locks the read pipe.

USBD_PHDC_OS_WrIntrLock

Description

Lock the write interrupt pipe.

Files

usbd_phdc_os.h / usbd_phdc_os.c

Prototype

```
void  USBD_PHDC_OS_WrIntrLock (CPU_INT08U   class_nbr,
                               CPU_INT16U   timeout,
                               USBD_ERR    *p_err);
```

Arguments

class_nbr

PHDC instance number.

timeout

Timeout.

p_err

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

1. Typical implementation will consist in pending on a semaphore that locks the write interrupt pipe.

2. `p_err` argument should be assigned as described in Table - Error Code Assignment in Function of Lock Operation Result in the *USBD_PHDC_OS_RdLock* page.

USBD_PHDC_OS_WrIntrUnLock

Description

Unlock the write interrupt pipe.

Files

`usbd_phdc_os.h / usbd_phdc_os.c`

Prototype

```
void  USBD_PHDC_OS_WrIntrUnlock (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

PHDC instance number.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

Typical implementation will consist in posting a semaphore that locks the write interrupt pipe.

USBD_PHDC_OS_WrBulkLock

Description

Lock the write bulk pipe.

Files

`usbd_phdc_os.h / usbd_phdc_os.c`

Prototype

```
void  USBD_PHDC_OS_WrBulkLock (CPU_INT08U   class_nbr,
                               CPU_INT08U   prio,
                               CPU_INT16U   timeout,
                               USBD_ERR    *p_err);
```

Arguments

`class_nbr`

PHDC instance number.

`prio`

Priority of the transfer. This value is between 0 and 4 and is computed in function of the transfer's QoS by the caller.

`timeout`

Timeout.

`p_err`

Pointer to variable that will receive the return error code from this function.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

1. Two typical implementations will be possible here. The first one consists in pending on a semaphore that locks the write bulk pipe, just as we saw previously.

2. But since different QoS data can travel using a single bulk IN endpoint, you might want to prioritize them in function of the QoS. See PHDC RTOS QoS-based scheduler for more details on how a priority manager can be implemented.

3. `p_err` argument should be assigned as described in Table - Error Code Assignment in Function of Lock Operation Result in the *USBD_PHDC_OS_RdLock* page.

USBD_PHDC_OS_WrBulkUnlock

Description

Unlock the write bulk pipe.

Files

`usbd_phdc_os.h / usbd_phdc_os.c`

Prototype

```
void  USBD_PHDC_OS_WrBulkUnlock (CPU_INT08U  class_nbr);
```

Arguments

`class_nbr`

PHDC instance number.

Returned Value

None.

Callers

Personal Healthcare Device Class.

Implementation guidelines

Two typical implementations will be possible here.

1. Post the mutex/semaphore that locks the write bulk pipe, if no priority management is implemented.

2. If priority management has been implemented, this call should release the scheduler (See PHDC RTOS QoS-based scheduler).

API - Vendor Class

- USBDev_AltSettingQtyGet
- USBDev_AssociatedIF_QtyGet
- USBDev_AltSettingSet
- USBDev_AltSettingCurGet
- USBDev_IsHighSpeed
- USBDev_BulkIn_Open
- USBDev_BulkOut_Open
- USBDev_IntrIn_Open
- USBDev_IntrOut_Open
- USBDev_PipeAddrGet
- USBDev_PipeClose
- USBDev_PipeStall
- USBDev_PipeAbort
- USBDev_CtrlReq
- USBDev_PipeWr
- USBDev_PipeWrExt
- USBDev_PipeRd
- USBDev_PipeRdAsync

Vendor Class Functions

- USBD_Vendor_Init
- USBD_Vendor_Add
- USBD_Vendor_CfgAdd
- USBD_Vendor_IsConn
- USBD_Vendor_Rd
- USBD_Vendor_RdAsync
- USBD_Vendor_Wr
- USBD_Vendor_WrAsync
- USBD_Vendor_IntrRd
- USBD_Vendor_IntrRdAsync
- USBD_Vendor_IntrWr
- USBD_Vendor_IntrWrAsync
- USBD_Vendor_MS_ExtPropertyAdd

USBD_Vendor_Init

Description

Initialize internal structures and local global variables used by the Vendor class.

Files

`usbd_vendor.h / usbd_vendor.c`

Prototype

```
void  USBD_Vendor_Init (USBD_ERR  *p_err);
```

Arguments

`p_err`

Pointer to variable that will receive the return error code from this function.

`USBD_ERR_NONE`

Returned Value

None.

Callers

Application.

Notes / Warnings

The initialization function *must* be called only once by the application, and before calling any other Vendor API.

USBD_Vendor_Add

Description

Create a new instance of the Vendor class.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_INT08U  USBD_Vendor_Add (CPU_BOOLEAN            intr_en,
                             CPU_INT16U             interval,
                             USBD_VENDOR_REQ_FNCT   req_callback,
                             USBD_ERR              *p_err);
```

Arguments

intr_en

Interrupt endpoints IN and OUT flag:

DEF_TRUE

Pair of interrupt endpoints added to interface.

DEF_FALSE

Pair of interrupt endpoints not added to interface.

interval

Endpoint interval in milliseconds. It must be a power of 2.

req_callback

Vendor-specific request callback.

`p_err`

Pointer to variable that will receive the return error code from this function.

`USBD_ERR_NONE`
`USBD_ERR_INVALID_ARG`
`USBD_ERR_ALLOC`

Returned Value

Class instance number, if NO error(s).

`USBD_CLASS_NBR_NONE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_CfgAdd

Description

Add a Vendor class instance into the specified configuration. The Vendor class instance was previously created by the function `USBD_Vendor_Add()`.

Files

`usbd_vendor.h / usbd_vendor.c`

Prototype

```
void  USBD_Vendor_CfgAdd (CPU_INT08U   class_nbr,
                          CPU_INT08U   dev_nbr,
                          CPU_INT08U   cfg_nbr,
                          USBD_ERR    *p_err);
```

Arguments

`class_nbr`

Class instance number.

`dev_nbr`

Device number.

`cfg_nbr`

Configuration index to add Vendor class instance to.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_ALLOC
USBD_ERR_NULL_PTR
```

```
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_CFG_INVALID_NBR
USBD_ERR_IF_ALLOC
USBD_ERR_IF_ALT_ALLOC
USBD_ERR_IF_INVALID_NBR
USBD_ERR_EP_NONE_AVAIL
USBD_ERR_EP_ALLOC
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. `USBD_Vendor_CfgAdd()` basically adds an Interface descriptor and its associated Endpoint descriptor(s) to the Configuration descriptor. One call to `USBD_Vendor_CfgAdd()` builds the Configuration descriptor corresponding to a Vendor-specific device with the following format:

```
Configuration Descriptor
    |-- Interface Descriptor (Vendor class)
        |-- Endpoint Descriptor (Bulk OUT)
        |-- Endpoint Descriptor (Bulk IN)
        |-- Endpoint Descriptor (Interrupt OUT) - optional
        |-- Endpoint Descriptor (Interrupt IN) - optional
```

2. The pair of Interrupt endpoints are optional. They can be added to the Interface descriptor by setting the parameter `intr_en` to `DEF_TRUE`.

3. If `USBD_Vendor_CfgAdd()` is called several times from the application, it allows to create multiple instances and multiple configurations. For instance, the following architecture could be created for an high-speed device:

```
High-speed
    |-- Configuration 0
        |-- Interface 0 (Vendor 0)
    |-- Configuration 1
        |-- Interface 0 (Vendor 0)
        |-- Interface 1 (Vendor 1)
```

   In that example, there are two instances of Vendor class: 'Vendor 0' and 'Vendor 1', and two possible configurations for the device: 'Configuration 0' and 'Configuration 1'. 'Configuration 1' is composed of two interfaces. Each class instance has an association with one of the interfaces. If 'Configuration 1' is activated by the host, it allows the host to access two different functionalities offered by the device.

USBD_Vendor_IsConn

Description

Get the vendor class connection state.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_BOOLEAN  USBD_Vendor_IsConn (CPU_INT08U  class_nbr);
```

Arguments

class_nbr

Class instance number.

Returned Value

DEF_YES, if Vendor class is connected.

DEF_NO, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_Rd

Description

Receive data from host through Bulk OUT endpoint. This function is blocking.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_INT32U  USBD_Vendor_Rd (CPU_INT08U   class_nbr,
                            void        *p_buf,
                            CPU_INT32U   buf_len,
                            CPU_INT16U   timeout,
                            USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length in octets.

timeout

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets received, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_RdAsync

Description

Receive data from host through Bulk OUT endpoint. This function is non-blocking. It returns immediately after transfer preparation. Upon transfer completion, a callback provided by the application will be called to finalize the transfer.

Files

`usbd_vendor.h / usbd_vendor.c`

Prototype

```
void  USBD_Vendor_RdAsync (CPU_INT08U               class_nbr,
                           void                    *p_buf,
                           CPU_INT32U               buf_len,
                           USBD_VENDOR_ASYNC_FNCT   async_fnct,
                           void                    *p_async_arg,
                           USBD_ERR                *p_err);
```

Arguments

`class_nbr`

Class instance number.

`p_buf`

Pointer to receive buffer.

`buf_len`

Receive buffer length in octets.

`async_fnct`

Receive callback.

`p_async_arg`

Additional argument provided by application for receive callback.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_Wr

Description

Send data to host through Bulk IN endpoint. This function is blocking.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_INT32U  USBD_Vendor_Wr (CPU_INT08U    class_nbr,
                            void         *p_buf,
                            CPU_INT32U    buf_len,
                            CPU_INT16U    timeout,
                            CPU_BOOLEAN   end,
                            USBD_ERR     *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length in octets.

timeout

Timeout in milliseconds.

end

End-of-transfer flag.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets sent, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

If end-of-transfer flag is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate the end of transfer to the host.

USBD_Vendor_WrAsync

Decription

Send data to host through Bulk IN endpoint. This function is non-blocking. It returns immediately after transfer preparation. Upon transfer completion, a callback provided by the application will be called to finalize the transfer.

Files

`usbd_vendor.h / usbd_vendor.c`

Prototype

```
void  USBD_Vendor_WrAsync (CPU_INT08U              class_nbr,
                           void                   *p_buf,
                           CPU_INT32U              buf_len,
                           USBD_VENDOR_ASYNC_FNCT  async_fnct,
                           void                   *p_async_arg,
                           CPU_BOOLEAN             end,
                           USBD_ERR               *p_err);
```

Arguments

`class_nbr`

Class instance number.

`p_buf`

Pointer to transmit buffer.

`buf_len`

Transmit buffer length in octets.

`async_fnct`

Transmit callback.

`p_async_arg`

Additional argument provided by application for transmit callback.

`end`

End-of-transfer flag.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

Number of octets sent, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

If end-of-transfer flag is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate the end of transfer to the host.

USBD_Vendor_IntrRd

Description

Receive data from host through Interrupt OUT endpoint. This function is blocking.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_INT32U  USBD_Vendor_IntrRd (CPU_INT08U   class_nbr,
                                void        *p_buf,
                                CPU_INT32U   buf_len,
                                CPU_INT16U   timeout,
                                USBD_ERR    *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length in octets.

timeout

Timeout in milliseconds.

p_err

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets received, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_IntrRdAsync

Description

Receive data from host through Interrupt OUT endpoint. This function is non-blocking. It returns immediately after transfer preparation. Upon transfer completion, a callback provided by the application will be called to finalize the transfer.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
void  USBD_Vendor_IntrRdAsync (CPU_INT08U              class_nbr,
                               void                   *p_buf,
                               CPU_INT32U              buf_len,
                               USBD_VENDOR_ASYNC_FNCT  async_fnct,
                               void                   *p_async_arg,
                               USBD_ERR               *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length in octets.

async_fnct

Receive callback.

p_async_arg

Additional argument provided by application for receive callback.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBD_Vendor_IntrWr

Description

Send data to host through Interrupt IN endpoint. This function is blocking.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
CPU_INT32U  USBD_Vendor_IntrWr (CPU_INT08U    class_nbr,
                                void         *p_buf,
                                CPU_INT32U    buf_len,
                                CPU_INT16U    timeout,
                                CPU_BOOLEAN   end,
                                USBD_ERR     *p_err);
```

Arguments

class_nbr

Class instance number.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length in octets.

timeout

Timeout in milliseconds.

end

End-of-transfer flag.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
```

Returned Value

Number of octets sent, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

If end-of-transfer flag is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate the end of transfer to the host.

USBD_Vendor_IntrWrAsync

Description

Send data to host through Interrupt IN endpoint. This function is non-blocking. It returns immediately after transfer preparation. Upon transfer completion, a callback provided by the application will be called to finalize the transfer.

Files

`usbd_vendor.h / usbd_vendor.c`

Prototype

```
void  USBD_Vendor_IntrWrAsync (CPU_INT08U              class_nbr,
                               void                   *p_buf,
                               CPU_INT32U              buf_len,
                               USBD_VENDOR_ASYNC_FNCT  async_fnct,
                               void                   *p_async_arg,
                               CPU_BOOLEAN             end,
                               USBD_ERR               *p_err);
```

Arguments

`class_nbr`

Class instance number.

`p_buf`

Pointer to transmit buffer.

`buf_len`

Transmit buffer length in octets.

`async_fnct`

Transmit callback.

`p_async_arg`

Additional argument provided by application for transmit callback.

`end`

End-of-transfer flag.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_NULL_PTR
USBD_ERR_INVALID_ARG
USBD_ERR_INVALID_CLASS_STATE
USBD_ERR_DEV_INVALID_NBR
USBD_ERR_EP_INVALID_NBR
USBD_ERR_DEV_INVALID_STATE
USBD_ERR_EP_INVALID_TYPE
USBD_ERR_EP_INVALID_STATE
```

Returned Value

Number of octets sent, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

If end-of-transfer flag is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate the end of transfer to the host.

USBD_Vendor_MS_ExtPropertyAdd

Description

Adds a Microsoft OS extended property to a vendor class instance.

Files

usbd_vendor.h / usbd_vendor.c

Prototype

```
void  USBD_Vendor_MS_ExtPropertyAdd (       CPU_INT08U   class_nbr,
                                            CPU_INT08U   property_type,
                                    const   CPU_INT08U  *p_property_name,
                                            CPU_INT16U   property_name_len,
                                    const   CPU_INT08U  *p_property,
                                            CPU_INT32U   property_len,
                                            USBD_ERR    *p_err)
```

Arguments

class_nbr

Class instance number.

property_type

Property type.

USBD_MS_OS_PROPERTY_TYPE_REG_SZ
USBD_MS_OS_PROPERTY_TYPE_REG_EXPAND_SZ
USBD_MS_OS_PROPERTY_TYPE_REG_BINARY
USBD_MS_OS_PROPERTY_TYPE_REG_DWORD_LITTLE_ENDIAN
USBD_MS_OS_PROPERTY_TYPE_REG_DWORD_BIG_ENDIAN
USBD_MS_OS_PROPERTY_TYPE_REG_LINK
USBD_MS_OS_PROPERTY_TYPE_REG_MULTI_SZ

p_property_name

Pointer to buffer that contains property name.

`property_name_len`

Length of property name in octets.

`p_property`

Pointer to buffer that contains property.

`property_len`

Length of property in octets.

`p_err`

Pointer to variable that will receive the return error code from this function.

```
USBD_ERR_NONE
USBD_ERR_INVALID_ARG
USBD_ERR_NULL_PTR
```

Returned Value

`None.`

Callers

Application.

Notes / Warnings

1. For more information on Microsoft OS descriptors, see http://msdn.microsoft.com/en-us/library/windows/hardware/gg463179.aspx.

2. For more information on property types, refer to "Table 3. Property Data Types" of "Extended Properties OS Feature Descriptor Specification" document provided by Microsoft available at http://msdn.microsoft.com/en-us/library/windows/hardware/gg463179.aspx.

3. Microsoft OS descriptors must be enabled by setting `USBD_CFG_MS_OS_DESC_EN` to `DEF_ENABLED` before using this function.

USBDev_API Functions

USBDev_API is a library implemented under Windows operating system. Functions return values and parameters use Windows data types such as DWORD, HANDLE, ULONG.

Refer to MSDN online documentation for more details about Windows data types (http://msdn.microsoft.com/en-us/library/aa383751(v=VS.85).aspx).

- USBDev_DevQtyGet
- USBDev_Open
- USBDev_Close
- USBDev_AltSettingQtyGet
- USBDev_AssociatedIF_QtyGet
- USBDev_AltSettingSet
- USBDev_AltSettingCurGet
- USBDev_IsHighSpeed
- USBDev_BulkIn_Open
- USBDev_BulkOut_Open
- USBDev_IntrIn_Open
- USBDev_IntrOut_Open
- USBDev_PipeAddrGet
- USBDev_PipeClose
- USBDev_PipeStall

- USBDev_PipeAbort
- USBDev_CtrlReq
- USBDev_PipeWr
- USBDev_PipeWrExt
- USBDev_PipeRd
- USBDev_PipeRdAsync

USBDev_DevQtyGet

Description

Get number of devices belonging to the specified GUID.

Files

usbdev_api.c

Prototype

```
DWORD  USBDev_DevQtyGet (const  GUID    guid_dev_if,
                                DWORD  *p_err);
```

Arguments

guid_dev_if

Device interface class GUID.

p_err

Pointer to variable that will receive the return error code from this function.

ERROR_SUCCESS

Returned Value

Number of devices for the provided GUID, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

The function `USBDev_DevQtyGet()` uses the concept of device information set. A device information set consists of device information elements for all the devices that belong to some device setup class or device interface class. The GUID passed to `USBDev_DevQtyGet()` function is a device interface class. Internally by using some control options the function retrieves the device information set which represents a list of all devices present in the system and registered under the specified GUID. More details about the device information set can be found at http://msdn.microsoft.com/en-us/library/ff541247(VS.85).aspx.

USBDev_Open

Description

Open a device by retrieving a general device handle.

Files

usbdev_api.c

Prototype

```
HANDLE  USBDev_Open (const  GUID    guid_dev_if,
                            DWORD   dev_nbr,
                            DWORD  *p_err);
```

Arguments

guid_dev_if

Device interface class GUID.

dev_nbr

Device number.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY
ERROR_BAD_DEVICE

Returned Value

Handle to device, if NO error(s).

INVALID_HANDLE_VALUE, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_Close

Description

Close a device by freeing any allocated resources and by releasing any created handles.

Files

usbdev_api.c

Prototype

```
void  USBDev_Close (HANDLE   dev,
                    DWORD   *p_err);
```

Arguments

dev

General handle to device.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE

Returned Value

None.

Callers

Application.

Notes / Warnings

`USBDev_Close()` closes any remaining open pipes. The open pipes are usually closed from the application by calling the function `USBDev_PipeClose`.

USBDev_AltSettingQtyGet

Description

Get number of alternate settings for the specified interface.

Files

usbdev_api.c

Prototype

```
UCHAR  USBDev_AltSettingQtyGet (HANDLE   dev,
                                UCHAR    if_nbr,
                                DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER

Returned Value

Number of alternate setting, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

1. An interface may include alternate settings that allow the endpoints and/or their characteristics to be varied after the device has been configured. The default setting for an interface is always alternate setting zero. Alternate settings allow a portion of the device configuration to be varied while other interfaces remain in operation.

2. The number of alternate settings gotten can be used to open a pipe associated with a certain alternate interface.

USBDev_AssociatedIF_QtyGet

Description

Get number of associated interfaces with the default interface. That is all the interfaces besides the default interface managed by `WinUSB.sys` and registered under the same GUID.

Files

`usbdev_api.c`

Prototype

```
UCHAR  USBDev_AssociatedIF_QtyGet (HANDLE   dev,
                                   DWORD   *p_err);
```

Arguments

`dev`

General handle to device.

`p_err`

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
```

Returned Value

Number of associated interfaces, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

Let’s assume that a device has three interfaces managed by WinUSB.sys driver and belonging to the same GUID: Interface #0, #1 and #2. Interface #0 is the default interface. Interfaces #1 and #2 are the associated interfaces. In that example calling `USBDev_AssociatedIF_QtyGet()` will return 2 associated interfaces.

USBDev_AltSettingSet

Description

Set the alternate setting of an interface.

Files

usbdev_api.c

Prototype

```
void  USBDev_AltSettingSet (HANDLE   dev,
                            UCHAR    if_nbr,
                            UCHAR    alt_set,
                            DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

alt_set

Alternate setting number.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER

Returned Value

None.

Callers

Application.

Notes / Warnings

This function sends a `SET_INTERFACE` request to the device to set the alternate setting number and updates the one used internally by WinUSB.

USBDev_AltSettingCurGet

Description

Get the current alternate setting for the specified interface.

Files

usbdev_api.c

Prototype

```
UCHAR  USBDev_AltSettingCurGet (HANDLE   dev,
                                UCHAR    if_nbr,
                                DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER

Returned Value

Current alternate setting number, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

This function gets the current alternate setting number used internally by WinUSB and gets the one from the device by sending a `GET_INTERFACE` request. Both alternate setting numbers are compared. If they match, the current alternate setting is returned.

USBDev_IsHighSpeed

Description

Specify if the device attached to PC is high speed or not.

Files

usbdev_api.c

Prototype

```
BOOL  USBDev_IsHighSpeed (HANDLE   dev,
                          DWORD   *p_err);
```

Arguments

dev

General handle to device.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_PARAMETER

Returned Value

TRUE, if device is high-speed.

FALSE, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_BulkIn_Open

Description

Open a Bulk IN pipe.

Files

usbdev_api.c

Prototype

```
HANDLE  USBDev_BulkIn_Open (HANDLE   dev,
                            UCHAR    if_nbr,
                            UCHAR    alt_set,
                            DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

alt_set

Alternate setting number for specified interface.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_NO_MORE_ITEMS

Returned Value

Handle to Bulk IN pipe, if NO error(s).

`INVALID_HANDLE_VALUE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_BulkOut_Open

Description

Open a Bulk OUT pipe.

Files

usbdev_api.c

Prototype

```
HANDLE  USBDev_BulkOut_Open (HANDLE   dev,
                             UCHAR    if_nbr,
                             UCHAR    alt_set,
                             DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

alt_set

Alternate setting number for specified interface.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_NO_MORE_ITEMS

Returned Value

Handle to Bulk OUT pipe, if NO error(s).

`INVALID_HANDLE_VALUE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_IntrIn_Open

Description

Open a Interrupt IN pipe.

Files

usbdev_api.c

Prototype

```
HANDLE  USBDev_IntrIn_Open (HANDLE   dev,
UCHAR    if_nbr,
UCHAR    alt_set,
DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

alt_set

Alternate setting number for specified interface.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_NO_MORE_ITEMS

Returned Value

Handle to Interrupt IN pipe, if NO error(s).

`INVALID_HANDLE_VALUE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_IntrOut_Open

Description

Open a Interrupt OUT pipe.

Files

usbdev_api.c

Prototype

```
HANDLE  USBDev_IntrOut_Open (HANDLE   dev,
                             UCHAR    if_nbr,
                             UCHAR    alt_set,
                             DWORD   *p_err);
```

Arguments

dev

General handle to device.

if_nbr

Interface number.

alt_set

Alternate setting number for specified interface.

p_err

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_NO_MORE_ITEMS
```

Returned Value

Handle to Interrupt OUT pipe, if NO error(s).

`INVALID_HANDLE_VALUE`, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_PipeAddrGet

Description

Get pipe address.

Files

usbdev_api.c

Prototype

```
UCHAR  USBDev_PipeAddrGet (HANDLE   pipe,
                           DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE

Returned Value

Pipe address, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

The pipe address is composed of the pipe direction and the pipe logical address. The pipe direction is located at bit 7. A value of '1' indicates an IN pipe and '0' an OUT pipe. The pipe logical address sits in bit 3 to bit 0.

USBDev_PipeClose

Description

Close a pipe.

Files

usbdev_api.c

Prototype

```
void  USBDev_PipeClose (HANDLE   pipe,
                        DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE

Returned Value

None

Callers

Application.

Notes / Warnings

None.

USBDev_PipeStall

Description

Stall a pipe or clear the stall condition of a pipe.

Files

usbdev_api.c

Prototype

```
void  USBDev_PipeStall (HANDLE   pipe,
                        BOOL     stall,
                        DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

stall

Indicate which action to do:

TRUE

Stall pipe.

FALSE

Clear stall condition of the pipe.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS

ERROR_INVALID_HANDLE
ERROR_NOT_ENOUGH_MEMORY

Returned Value

None.

Callers

Application.

Notes / Warnings

The SET_FEATURE standard request is sent to the device to stall the pipe. The CLEAR_FEATURE standard request is sent to the device to clear the stall condition of the pipe.

USBDev_PipeAbort

Description

Aborts all of the pending transfers for a pipe.

Files

usbdev_api.c

Prototype

```
void  USBDev_PipeAbort (HANDLE   pipe,
                        DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_err

Pointer to variable that will receive the return error code from this function:

ERROR_SUCCESS
ERROR_INVALID_HANDLE

Returned Value

None.

Callers

Application.

Notes / Warnings

None.

USBDev_CtrlReq

Description

Send control data over the default control endpoint.

Files

`usbdev_api.c`

Prototype

```
ULONG  USBDev_CtrlReq (HANDLE   dev,
                       UCHAR    bm_req_type,
                       UCHAR    b_request,
                       USHORT   w_value,
                       USHORT   w_index,
                       UCHAR   *p_buf,
                       USHORT   buf_len,
                       DWORD   *p_err);
```

Arguments

`dev`

General handle to device

`bm_req_type`

Variable representing `bmRequestType` of setup packet. `bmRequestType` is a bitmap with the following characteristics:

D7

Data transfer direction:

`‘0’: USB_DIR_HOST_TO_DEVICE`
`‘1’: USB_DIR_DEVICE_TO_HOST`

D6...5

Request type:

```
‘00’: USB_REQUEST_TYPE_STD (standard)
‘01’: USB_REQUEST_TYPE_CLASS
‘10’: USB_REQUEST_TYPE_VENDOR
```

D4...0

Recipient:

```
‘0000’: USB_RECIPIENT_DEV (device)
‘0001’: USB_RECIPIENT_IF (interface)
‘0010’: USB_RECIPIENT_ENDPOINT
```

`b_request`

Variable representing `bRequest` of setup packet. Possible values are:

bRequest	Description
`GET_STATUS`	Returns status for the specified recipient.
`CLEAR_FEATURE`	Clear or disable a specific feature.
`SET_FEATURE`	Set or enable a specific feature.
`SET_ADDRESS`	Set the device address for all future device accesses.
`GET_DESCRIPTOR`	Return the specified descriptor if the descriptor exists.
`SET_DESCRIPTOR`	Update existing descriptors or new descriptors may be added.
`GET_CONFIGURATION`	Return the current device configuration value.
`SET_CONFIGURATION`	Set the device configuration.
`GET_INTERFACE`	Return the selected alternate setting for the specified interface.
`SET_INTERFACE`	Select an alternate setting for the specified interface.
`SYNCH_FRAME`	Set and then report an endpoint’s synchronization frame.

`w_value`

Variable representing wValue of setup packet.

`w_index`

Variable representing wIndex of setup packet.

`p_buf`

Pointer to transmit or receive buffer for data phase of control transfer.

`buf_len`

Length of transmit or receive buffer.

`p_err`

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_NOT_ENOUGH_MEMORY
ERROR_GEN_FAILURE
```

Returned Value

None

Callers

Application.

Notes / Warnings

1. The value of `w_value` and `w_index` arguments vary according to the specific request defined by `b_request` argument.

   a. When the request's recipient is an interface or an endpoint, `w_index` must be properly configured. For an interface recipient, `w_index` will contain the interface number. For an endpoint recipient, `w_index` will contain the endpoint address. This one can be retrieved using the function `USBDev_PipeAddrGet()` .

2. The following code shows an example using `USBDev_CtrlReq()` to send the `SET_INTERFACE` request:

```
DWORD   err;
                            /* Select alternate setting #1 for default interface. */
USBDev_CtrlReq ( dev_handle,
                (USB_DIR_HOST_TO_DEVICE | USB_REQUEST_TYPE_STD | USB_RECIPIENT_IF),
                 SET_INTERFACE,
                 1,         /* Alternate setting #1.                              */
                 0,         /* Interface #0 inside active configuration.          */
                 0,         /* No data phase.                                     */
                 0,
                &err);
if (err != ERROR_SUCCESS) {
    printf("[ERROR #%d] SET_INTERFACE(1) request failed.\n", err);
}
```

   More details about USB device requests can be found in “Universal Serial Bus Specification, Revision 2.0, April 27, 2000”, section 9.3.

3. A control transfer is composed of 3 stages: Setup, Data (IN, OUT or no data stage) and Status. The table below presents the parameters of `USBDev_CtrlReq()` involved in each specific stage.

Control Transfer Stage	Parameter Involved	Note

Setup	`bm_req_type`
	`b_request`
	`w_value`
	`w_index`
		This parameters are used to build the Setup packet sent by the host to the device.

Data	`p_buf` `buf_len`	If no data stage is required, you just need to set `p_buf` to null pointer and `buf_len` to the value 0.

Status	None	NO parameter is involved for the Status stage. It is managed automatically by the Windows Host stack.

USBDev_PipeWr

Description

Write data to device over the specified pipe.

Files

usbdev_api.c

Prototype

```
DWORD  USBDev_PipeWr (HANDLE   pipe,
                      UCHAR   *p_buf,
                      DWORD    buf_len,
                      DWORD    timeout,
                      DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length.

timeout

Timeout in milliseconds. A value of 0 indicates a wait forever.

p_err

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_USER_BUFFER
ERROR_BAD_PIPE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY
ERROR_SEM_TIMEOUT
```

Returned Value

Number of bytes written, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_PipeWrExt

Description

Write data to device over the specified pipe.

Files

usbdev_api.cs

Prototype

```
DWORD  USBDev_PipeWrExt (HANDLE   pipe,
                         UCHAR   *p_buf,
                         DWORD    buf_len,
                         BOOL     end,
                         DWORD    timeout,
                         DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_buf

Pointer to transmit buffer.

buf_len

Transmit buffer length.

end

End-of-transfer flag (see Note #1).

timeout

Timeout in milliseconds. A value of 0 indicates a wait forever.

`p_err`

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_USER_BUFFER
ERROR_BAD_PIPE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY
ERROR_SEM_TIMEOUT
```

Returned Value

Number of bytes written, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

If end-of-transfer is set and transfer length is multiple of maximum packet size, a zero-length packet is transferred to indicate a short transfer to the device.

USBDev_PipeRd

Description

Read data from device over the specified pipe.

Files

usbdev_api.c

Prototype

```
DWORD  USBDev_PipeRd (HANDLE   pipe,
                      UCHAR   *p_buf,
                      DWORD    buf_len,
                      DWORD    timeout,
                      DWORD   *p_err);
```

Arguments

pipe

Pipe handle.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length.

timeout

Timeout in milliseconds. A value of 0 indicates a wait forever.

p_err

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_USER_BUFFER
ERROR_BAD_PIPE
ERROR_INVALID_PARAMETER
ERROR_NOT_ENOUGH_MEMORY
ERROR_SEM_TIMEOUT7
```

Returned Value

Number of bytes received, if NO error(s).

0, otherwise.

Callers

Application.

Notes / Warnings

None.

USBDev_PipeRdAsync

Description

Read data from device over the specified pipe. This function returns immediately if data is not present. The data will be retrieved later.

Files

usbdev_api.c

Prototype

```
void  USBDev_PipeRdAsync (HANDLE                   pipe,
                          UCHAR                   *p_buf,
                          DWORD                    buf_len,
                          USBDEV_PIPE_RD_CALLBACK  callback,
                          void                    *p_callback_arg,
                          DWORD                   *p_err);
```

Arguments

pipe

Pipe handle.

p_buf

Pointer to receive buffer.

buf_len

Receive buffer length.

callback

Pointer to application callback called by Asynchronous thread upon completion.

p_callback_arg

Pointer to argument which can carry private information passed by application. This argument is used when the callback is called.

`p_err`

Pointer to variable that will receive the return error code from this function:

```
ERROR_SUCCESS
ERROR_INVALID_HANDLE
ERROR_INVALID_USER_BUFFER
ERROR_BAD_PIPE
ERROR_NOT_ENOUGH_MEMORY
ERROR_SEM_TIMEOUT
```

Returned Value

None.

Callers

Application.

Notes / Warnings

1. When a IN pipe is open with one of the open functions `USBDev_xxxxIn_Open()`, a thread is automatically created. This thread is in charge of informing the application about a completed asynchronous IN transfer. Upon completion of an asynchronous transfer, the thread is waken up and calls the application callback provided to `USBDev_API` library using the `callback` argument.

2. `USBDev_API` library allows to queue several asynchronous IN transfers for the same pipe.

Error Codes

This page provides a brief explanation of µC/USB-Device error codes defined in `usbd_core.h`. Any error codes not listed here may be searched in `usbd_core.h` for both their numerical value and usage. This appendix also contains class-specific error codes.

Error codes list

Categories	Value	Error Code	Potential Cause(s)	Solution(s)
Generic Errors	0	`USBD_ERR_NONE`	No error, nothing to do.	None.
	1	`USBD_ERR_FAIL`	Generic error occurred.	Varies depending where the error occurred.
	2	`USBD_ERR_RX`	Generic error occurred during a 'Rx' transfer.	Varies depending where the error occurred.
	3	`USBD_ERR_TX`	Generic error occurred during a 'Tx' transfer.	Varies depending where the error occurred.
	4	`USBD_ERR_ALLOC`	Generic allocation error.	The memory segment (or the heap, from uC/LIB) from which the memory is allocated does not have enough space remaining. Try increasing the size of the memory segment or the heap, if possible. If not, try adjusting the configuration values in `usbd_cfg.h` or `app_usbd_cfg.h` to better fit the needs of the application.
	5	`USBD_ERR_NULL_PTR`	Null pointer passed as an argument.	See where error occurred and what was the parameter checked.
	6	`USBD_ERR_INVALID_ARG`	An invalid argument has been passed to the function.	See where error occurred and what was the parameter checked.
	7	`USBD_ERR_INVALID_CLASS_STATE`	The class is in an invalid state.	See where error occurred and what is the current state of the class.
Device Errors	100	`USBD_ERR_DEV_ALLOC`	Tried to allocate more devices than the configured value allows.	`USBD_CFG_MAX_NBR_DEV` must be increased. This define can be found in `usbd_cfg.h`.

	101	USBD_ERR_DEV_INVALID_NBR	Unable to obtain device or driver reference based on specified dev_nbr.	Make sure dev_nbr is correct.
	102	USBD_ERR_DEV_INVALID_STATE	Device is in an incorrect state (None, Init, Attached, Default, Addressed, Configured or Suspended) to execute the requested operation.	Verify what state the device is currently in and see why it cannot execute the requested operation or why it is in this state.
	103	USBD_ERR_DEV_INVALID_SPD	High-speed operation attempted on a driver/controller that does not support high-speed operations.	If the controller used support high-speed operations, make sure the speed declared in the driver configuration (USBD_DrvCfg_xxxx) is correct. If the controller used does not support high-speed operations, high-speed operations cannot be executed.
	104	USBD_ERR_DEV_UNAVAIL_FEAT	The feature requested is unavailable in the module used.	See where/why the error occurred.
Configuration Errors	200	USBD_ERR_CFG_ALLOC	Tried to allocate more USB configuration than the configured value allows.	USBD_CFG_MAX_NBR_CFG must be increased. This define can be found in usbd_cfg.h.
	201	USBD_ERR_CFG_INVALID_NBR	Unable to obtain configuration reference based on cfg_nbr or cfg_nbr passed is invalid.	Make sure cfg_nbr is correct.
	202	USBD_ERR_CFG_INVALID_MAX_PWR	max_pwr parameter is invalid.	Adjust max_pwr value.
	203	USBD_ERR_CFG_SET_FAIL	Call to driver's CfgSet failed.	See each driver's CfgSet() function for more details.
Interface Errors	300	USBD_ERR_IF_ALLOC	Tried to allocate more USB interfaces than the configured value allows.	USBD_CFG_MAX_NBR_IF must be increased. This define can be found in usbd_cfg.h.
	301	USBD_ERR_IF_INVALID_NBR	Unable to obtain interface reference based on if_nbr.	Make sure if_nbr is correct.
	302	USBD_ERR_IF_ALT_ALLOC	Tried to allocate more USB alternate interfaces than the configured value allows.	USBD_CFG_MAX_NBR_IF_ALT must be increased. This define can be found in usbd_cfg.h.

	303	USBD_ERR_IF_ALT_INVALID_NBR	Unable to obtain interface reference based on if_alt_nbr.	Make sure if_alt_nbr is correct.
	304	USBD_ERR_IF_GRP_ALLOC	Tried to allocate more interface groups than the configured value allows.	USBD_CFG_MAX_NBR_IF_GRP must be increased. This define can be found in usbd_cfg.h.
	305	USBD_ERR_IF_GRP_NBR_IN_USE	Interface is already associated with an interface group.	Cannot associate an interface to more than one group.
Endpoint Errors	400	USBD_ERR_EP_ALLOC	Tried to allocate more endpoints than the configured value allows.	USBD_CFG_MAX_NBR_EP_DESC must be increased. This define can be found in usbd_cfg.h.
	401	USBD_ERR_EP_INVALID_ADDR	Unable to obtain endpoint reference based on ep_addr.	Make sure ep_addr and dev_nbr are correct.
	402	USBD_ERR_EP_INVALID_STATE	Endpoint is in an invalid state (Close, Open, Stall) to execute requested operation.	Verify what state the endpoint is currently in and see why it cannot execute the requested operation or why it is in this state.
	403	USBD_ERR_EP_INVALID_TYPE	Endpoint type (Control, Bulk, Interrupt, Isochronous) is invalid.	Make sure the endpoint type matches the type of endpoint of the function called.
	404	USBD_ERR_EP_NONE_AVAIL	Requested endpoint is unavailable.	Try to adjust the USBD_CFG_MAX_NBR_EP_OPEN (in usbd_cfg.h) constant or see if the driver used supports the endpoint type requested and/or has enough endpoints to open the requested one.
	405	USBD_ERR_EP_ABORT	Transfer has been aborted, or error when aborting.	
	406	USBD_ERR_EP_STALL	Unable to execute correctly the stall operation requested.	See driver's EP_Stall() function for more details.
	407	USBD_ERR_EP_IO_PENDING	A transfer is already in progress on the specified endpoint and the core cannot queue the next transfer after it.	Avoid executing synchronous transfers on busy endpoints or trying to queue asynchronous transfers after synchronous ones.

	408	USBD_ERR_EP_QUEUING	Unable to queue URB.	Too much asynchronous transfers are queued at the same time, wait for a transfer to finish before submitting another one.
OS Layer Errors	500	USBD_ERR_OS_INIT_FAIL	OS layer initialization failed.	See OS User's Manual and OS layer where error occurred for more details.
	501	USBD_ERR_OS_SIGNAL_CREATE	OS signal creation failed.	See OS User's Manual and OS layer where error occurred for more details.
	502	USBD_ERR_OS_FAIL	OS layer operation failed.	See OS User's Manual and OS layer where error occurred for more details.
	503	USBD_ERR_OS_TIMEOUT	OS pend/lock operation timed-out.	See OS User's Manual and OS layer where error occurred for more details.
	504	USBD_ERR_OS_ABORT	OS pend/lock operation was aborted.	See OS User's Manual and OS layer where error occurred for more details.
	505	USBD_ERR_OS_DEL	OS layer deletion failed.	See OS User's Manual and OS layer where error occurred for more details.
Device Driver Errors	700	USBD_ERR_DRV_BUF_OVERFLOW	Driver indicated that buffer overflowed.	See device driver for more details.
	701	USBD_ERR_DRV_INVALID_PKT	Driver indicated an invalid packet has been received.	See device driver for more details.
Generic Class Errors	1000	USBD_ERR_CLASS_INVALID_NBR	class_nbr or sublcass_nbr parameter is invalid.	Check the value of class_nbr or subclass_nbr where the error occurred.
	1001	USBD_ERR_CLASS_XFER_IN_PROGRESS	A transfer is already in progress on the endpoint used by the class.	Wait for the transfer to completed before executing another one.
Audio Class Errors	1100	USBD_ERR_AUDIO_INSTANCE_ALLOC	Tried to allocate more audio class instances than configured values allow.	Depending on where the error occurred, either USBD_AUDIO_CFG_MAX_NBR_CFG or USBD_AUDIO_CFG_MAX_NBR_AIC must be increased. These defines can be found in usbd_cfg.h.
	1101	USBD_ERR_AUDIO_AS_IF_ALLOC	Tried to allocate more audio streaming interfaces than configured values allow.	Depending on where the error occurred, either USBD_AUDIO_CFG_MAX_NBR_CFG, USBD_AUDIO_CFG_MAX_NBR_AIC, USBD_AUDIO_CFG_MAX_NBR_AS_IF or USBD_AUDIO_CFG_MAX_NBR_IF_ALT must be increased. These defines can be found in usbd_cfg.h.

1102	USBD_ERR_AUDIO_IT_ALLOC	Tried to allocate more input terminals than configured value allows.	USBD_AUDIO_CFG_MAX_NBR_IT must be increased. This define can be found usbd_cfg.h.
1103	USBD_ERR_AUDIO_OT_ALLOC	Tried to allocate more output terminals than configured value allows.	USBD_AUDIO_CFG_MAX_NBR_OT must be increased. This define can be found usbd_cfg.h.
1104	USBD_ERR_AUDIO_FU_ALLOC	Tried to allocate more feature units than configured value allows.	USBD_AUDIO_CFG_MAX_NBR_FU must be increased. This define can be found usbd_cfg.h.
1105	USBD_ERR_AUDIO_MU_ALLOC	Tried to allocate more mixing units than configured value allows.	USBD_AUDIO_CFG_MAX_NBR_MU must be increased. This define can be found usbd_cfg.h.
1106	USBD_ERR_AUDIO_SU_ALLOC	Tried to allocate more selector units than configured value allows.	USBD_AUDIO_CFG_MAX_NBR_SU must be increased. This define can be found usbd_cfg.h.
1107	USBD_ERR_AUDIO_REQ_INVALID_CTRL	Unable to process class request, the ctrl field is invalid/not supported.	The audio class will stall the control endpoint to indicate to the host that the device does not support this type of request.
1108	USBD_ERR_AUDIO_REQ_INVALID_ATTRIB	Unable to process class request, the attrib field is invalid/not supported.	The audio class will stall the control endpoint to indicate to the host that the device does not support this type of request.
1109	USBD_ERR_AUDIO_REQ_INVALID_RECIPIENT	Unable to process class request, the recipient field is invalid/not supported.	The audio class will stall the control endpoint to indicate to the host that the device does not support this type of request.
1110	USBD_ERR_AUDIO_REQ	Unable to process class request for any other reason.	The audio class will stall the control endpoint to indicate to the host that the device does not support this type of request.
1111	USBD_ERR_AUDIO_INVALID_SAMPLING_FRQ	The requested sampling frequency is not supported.	See specific audio codec's code for setting sampling frequency.
1112	USBD_ERR_AUDIO_CODEC_INIT_FAILED	Audio codec initialization failed.	See specific audio codec's code for initialization.

CDC Errors	1200	USBD_ERR_CDC_INSTANCE_ALLOC	Tried to allocate more CDC instances than configured values allow.	Depending on where the error occurred, either USBD_CDC_CFG_MAX_NBR_DEV, USBD_CDC_CFG_MAX_NBR_CFG or USBD_CDC_CFG_MAX_NBR_DATA_IF must be increased. These defines can be found in usbd_cfg.h.
	1201	USBD_ERR_CDC_DATA_IF_ALLOC	Tried to allocate more CDC data interfaces than configured values allow.	USBD_CDC_CFG_MAX_NBR_DATA_IF must be increased. This define can be found in usbd_cfg.h.
	1250	USBD_ERR_CDC_SUBCLASS_INSTANCE_ALLOC	Tried to allocate more instances of a given CDC subclass than configured values allow.	USBD_ACM_SERIAL_CFG_MAX_NBR_DEV must be increased. This define can be found in usbd_cfg.h.
HID Class Errors	1300	USBD_ERR_HID_INSTANCE_ALLOC	Tried to allocate more HID class instances than configured values allow.	Depending on where the error occurred, either USBD_HID_CFG_MAX_NBR_DEV or USBD_HID_CFG_MAX_NBR_CFG must be increased. These defines can be found in usbd_cfg.h.
	1301	USBD_ERR_HID_REPORT_INVALID	The format of the report descriptor given as parameter to USBD_HID_Add() is invalid.	See where exactly the error occurred in USBD_HID_Report_Parse() to have more details about the reason the parsing failed.
	1302	USBD_ERR_HID_REPORT_ALLOC	Failed to allocate internal data structure for report descriptor.	There can be a few reasons why this failed. You can try increasing USBD_HID_CFG_MAX_NBR_REPORT_ID in usbd_cfg.h, or see if an invalid parameter has been passed to function USBD_HID_ReportID_Get().
	1303	USBD_ERR_HID_REPORT_PUSH_POP_ALLOC	Failed to allocate internal data structure for push-pop item.	USBD_HID_CFG_MAX_NBR_REPORT_PUSHPOP must be increased. This define can be found in usbd_cfg.h.
MSC Errors	1400	USBD_ERR_MSC_INSTANCE_ALLOC	Tried to allocate more MSC instances than configured values allow.	Depending on where the error occurred, either USBD_MSC_CFG_MAX_NBR_DEV or USBD_MSC_CFG_MAX_NBR_CFG must be increased. These defines can be found in usbd_cfg.h.
	1401	USBD_ERR_MSC_INVALID_CBW	Command Block Wrapper (CBW) received by the internal MSC task is invalid.	The CBW content contains an error or its length is incorrect. The MSC class will report to the host through the CSW that the SCSI command has failed and the host will take the proper action to continue.

1402	USBD_ERR_MSC_INVALID_DIR	Mismatch between direction indicated by CBW and the SCSI command.	The MSC class will report to the host through the CSW that the SCSI command has failed and the host will take the proper recovery action.
1403	USBD_ERR_MSC_MAX_LUN_EXCEED	No more logical unit can be added to the MSC class.	USBD_MSC_CFG_MAX_LUN must be increased. This define can be found in usbd_cfg.h .
1404	USBD_ERR_MSC_MAX_VEN_ID_LEN_EXCEED	Vendor ID string associated to a logical unit is too long.	Shorten your vendor string to be less than or equal to 8 characters.
1405	USBD_ERR_MSC_MAX_PROD_ID_LEN_EXCEED	Product ID string associated to a logical unit is too long.	Shorten your vendor string to be less than or equal to 16 characters.
1406	USBD_ERR_SCSI_UNSUPPORTED_CMD	SCSI command not recognized.	The host has sent a SCSI command not supported by MSC (Refer to section SCSI Commands for supported SCSI commands). The MSC class will report to the host through the CSW that the SCSI command has failed. The host will take the proper action.
1407	USBD_ERR_SCSI_MORE_DATA	Read or write SCSI command requires more data to be read or written.	The MSC class will read more data from the storage medium to send it to the host or will wait for data from host to write it to the storage medium.
1408	USBD_ERR_SCSI_LU_NOTRDY	Logical unit not ready to perform any operations.	For any reason, the storage layer has returned that the logical unit is not ready to be accessed. The MSC class will report to the host through the CSW that the SCSI command has failed. The host may send periodically the TEST_UNIT_READY SCSI command until the logical unit is ready to be accessed.
1409	USBD_ERR_SCSI_LU_NOTSUPPORTED	Logical unit number not supported.	The storage layer reports that the logical unit number does not exist. The MSC class will report to the host through the CSW that the SCSI command has failed. The host should stop trying to access the logical unit number.
1410	USBD_ERR_SCSI_LU_BUSY	Logical unit number is busy with other operations.	For any reason, the storage layer has returned that the logical unit is occupied with an operation in progress. The MSC class will report to the host through the CSW that the SCSI command has failed. The host may send periodically the TEST_UNIT_READY SCSI command until the logical unit has finished its ongoing operation.

1411	USBD_ERR_SCSI_LOG_BLOCK_ADDR	Logical block address out of range when host asks the device to test one or more sectors.	The MSC class will report to the host through the CSW that the SCSI command has failed. The host should not try to access again this sector.
1412	USBD_ERR_SCSI_MEDIUM_NOTPRESENT	Removable storage medium is not present.	When an MSC device has a fixed storage medium, this one is always present. For MSC devices with a removable storage medium, upon connection to the PC, the medium may not be present. In that case, the MSC class will report to the host through the CSW that the SCSI command has failed. The host may send periodically the TEST_UNIT_READY SCSI command until the storage medium is inserted.
1413	USBD_ERR_SCSI_MEDIUM_NOT_RDY_TO_RDY	The storage medium is transitioning to the ready state.	The storage medium cannot accept yet to be accessed by the host because it changes its internal state. In that case, the storage layer returns this error code. The MSC class will report to the host through the CSW that the SCSI command has failed. The host may send periodically the TEST_UNIT_READY SCSI command until the storage medium has completed its transition to the ready state.
1414	USBD_ERR_SCSI_MEDIUM_RDY_TO_NOT_RDY	The storage medium is transitioning to the not ready state.	The storage medium won't accept anymore to be accessed by the host because it changes its internal state. In that case, the storage layer returns this error code. T he MSC class will report to the host through the CSW that the SCSI command has failed. The host may send periodically the TEST_UNIT_READY SCSI command until the storage medium is afresh in the ready state.
1415	USBD_ERR_SCSI_LOCK	Locking a storage medium has failed.	When the host accesses a storage medium, it becomes the only owner of this storage. No embedded file system application can access this storage. The ownership is guaranteed with a lock system. If the lock cannot be acquired, it means that an embedded file system application has the storage ownership. The MSC will report to the host through the CSW that the SCSI command has failed. The host should attempt again the SCSI command until the lock can be acquired.

	1416	USBD_ERR_SCSI_LOCK_TIMEOUT	Locking a storage medium has timeout.	If the lock is acquired, an embedded file system application has the storage ownership. The MSC class attempts acquiring the lock during a certain period of time. When this time period has elapsed, the lock attempt timeouts. The MSC will report to the host through the CSW that the SCSI command has failed. The host should attempt again the SCSI command until the lock can be acquired.
	1417	USBD_ERR_SCSI_UNLOCK	Unlocking the medium storage has failed.	The MSC class was not able to release the storage ownership. The MSC will report to the host through the CSW that the SCSI command has failed. The host should attempt again the SCSI command until the lock release succeeds.
PHDC Errors	1500	USBD_ERR_PHDC_INSTANCE_ALLOC	Tried to allocate more PHDC instances than configured values allow.	Depending on where the error occurred, either USBD_PHDC_CFG_MAX_NBR_DEV or USBD_PHDC_CFG_MAX_NBR_CFG must be increased. These defines can be found in usbd_cfg.h.
Vendor Errors	1600	USBD_ERR_VENDOR_INSTANCE_ALLOC	Tried to allocate more Vendor class instances than configured values allow.	Depending on where the error occurred, either USBD_VENDOR_CFG_MAX_NBR_DEV or USBD_VENDOR_CFG_MAX_NBR_CFG must be increased. These defines can be found in usbd_cfg.h.