USB Device CDC-ACM Class Driver User Guide

Version 3.60

For use with USBD CDC-ACM Class Driver versions 5.03 and above

Date: 25-Sep-2017 10:31

Table of Contents

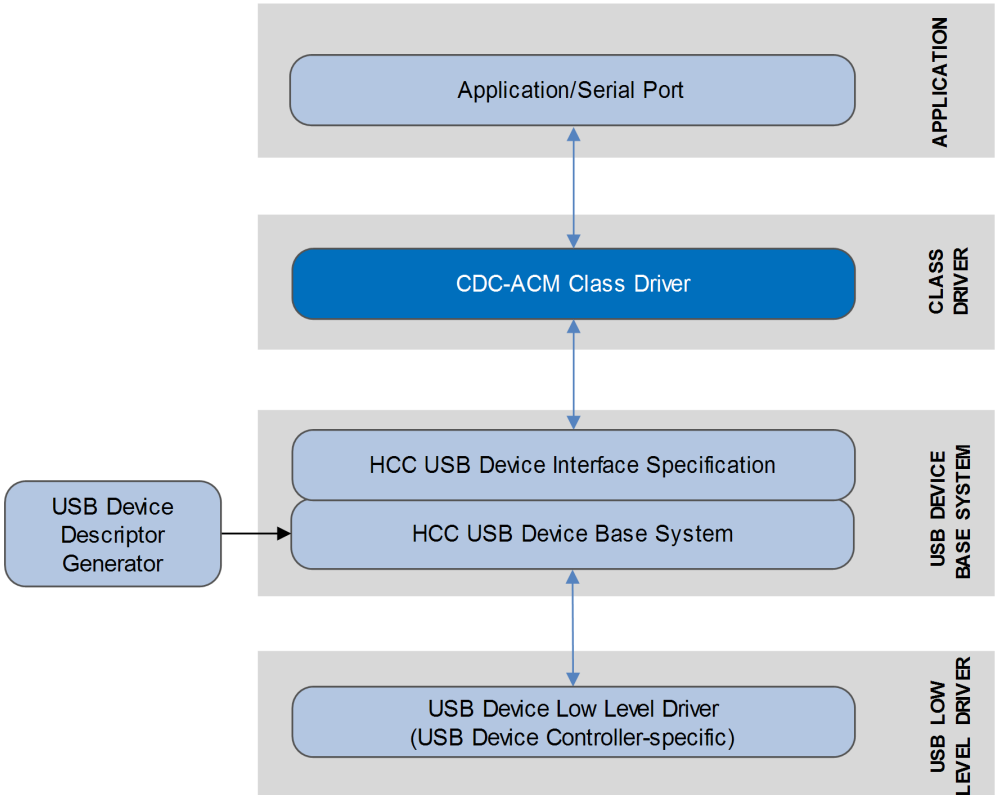
1 System Overview

1.1 Introduction

This guide is for those who want to implement an Embedded USB CDC-ACM class driver. The Communications Device Class - Abstract Control Model subclass (CDC-ACM) is used for connecting legacy serial ports or modems over a USB link to a host system.

The **usbd_cd_cdc_acm** package is a function device implementation of this class. This allows a device to connect to a host system and appear to the host system as one or more locally connected serial ports.

The system structure is shown in the diagram below:



On the embedded device side, this serial port or modem can be either a real device or a virtual device. This class driver is effectively a library. It provides a set of function calls for an application to use to send and receive data through the serial port. The **usbd_cdcacm_init()** function registers the class driver with the Embedded USB Device (EUSBD) base system and this call sets up callbacks for the base system to use.

Note: This module is part of the EUSBD system, as described in the *HCC Embedded USB Device Base System User Guide*. This module communicates with the EUSBD base system through the EUSBD device interface, as described in the above manual.

Note:

- This module conforms to the *Universal Serial Bus Class Definition for Communication Devices Version 1.1* and specifically to the sections relating to the Abstract Control Model. This document may be downloaded from www.usb.org and is a recommended reference.
- As a virtual serial port, the communications settings have no effect on the transmission across the USB interface. These settings are exchanged between the host and the device so that in the scenario where the real port exists the remote device can configure a real COM port under the host's control.

Getting Started

To use this class driver, take the following steps:

1. Generate a device configuration that includes one or more CDC-ACM interfaces.
2. Set the configuration parameters as described in Configuration Options.
3. Call the **usbd_cdcacm_init()** and **usbd_cdcacm_start()** functions to initialize the class driver.
4. Use the Application Programming Interface (API) calls to access and control the serial port.

Features of the CDC-ACM Protocol

The CDC-ACM protocol has the following basic feature set:

- Every CDC-ACM line must have two interfaces configured:
  - Data interface – requires one Bulk IN and one Bulk OUT endpoint.
  - Communications interface – requires one Interrupt IN endpoint.
- For data communication the protocol uses:
  - the Bulk IN USB endpoint to send data to the USB host from the serial port on the embedded system.
  - the Bulk OUT USB endpoint to send data from the USB host to the serial port on the embedded system.
- For control communication it uses:
  - the USB control channel to send line control information from the host to the serial port on the embedded system.
  - the Interrupt IN USB endpoint to send line status information from the serial port on the embedded system to the USB host.
- Line control and status information is typically ignored if the serial port is virtual.

1.2 Feature Check

The main features of the class driver are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Supports all devices that conform to the USB CDC-ACM specification.
- Supports all devices that conform to the HCC Media Driver Interface Specification.
- Compatible with sample device files produced by using the HCC *USB Device Descriptor Generator*.
- Uses a system of callbacks for user-specified events.

1.3 Packages and Documents

Packages

This table lists the packages that you need in order to use this module:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbd_base	The USB device base package. This is the framework used by USB class drivers to communicate over USB using a specific USB device controller package.
usbd_cd_cdc_acm	The USB device CDC-ACM class driver package described by this document.

Documents

For an overview of HCC's embedded USB stacks, see Product Information on the main HCC website.

Readers should note the points in the HCC Documentation Guidelines on the HCC documentation website.

HCC Firmware Quick Start Guide

This document describes how to install packages provided by HCC in the target development environment. Also follow the *Quick Start Guide* when HCC provides package updates.

HCC Source Tree Guide

This document describes the HCC source tree. It gives an overview of the system to make clear the logic behind its organization.

HCC Embedded USB Device Base System User Guide

This document defines the USB device base system upon which the complete USB stack is built.

HCC USB Device CDC-ACM Class Driver User Guide

This is this document.

CDC-ACM Driver for Windows Installation Guide

This document describes how to install the CDC-ACM driver on a Windows system.

HCC USB Device Descriptor Generator User Guide

This document describes the tool that creates USB descriptor files for inclusion in a project that uses the EUSBD stack.

1.4 Change History

This section describes past changes to this manual.

- To view or download earlier manuals, see Archive: Embedded USB Device CDC-ACM Class Driver User Guide.
- For the history of changes made to the package code itself, see History: usbd_cd_cdc_acm.

The current version of this manual is 3.60. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
3.60	2017-09-25	5.03	Added USBD_CDCACM_BUSY error code to function **usbd_cdcacm_send()**.
3.50	2017-06-16	5.02	New *Change History* format.
3.40	2016-04-21	5.02	Added function group descriptions to API.
3.30	2015-04-02	5.01	Added *Change History*.
3.20	2014-08-27	5.01	Changed *Documents* list.
3.10	2014-08-13	5.01	Reorganized *System Overview*.
3.00	2014-05-15	3.11	First online release.

2 Source File List

This section describes all the source code files included in the system. These files follow the HCC Embedded standard source tree system, described in the *HCC Source Tree Guide*. All references to file pathnames refer to locations within this standard source tree, not within the package you initially receive.

> **Note:** Do not modify any files except the configuration file.

2.1 API Header File

The file **src/api/api_usbd_cdcacm.h** is the only file that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

2.2 Configuration File

The file **src/config/config_usbd_cdcacm.h** contains all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

2.3 Source Code

The file **src/usb-device/class-drivers/cdc-acm/usbd_cdcacm.c** is the main code for USB device CDC-ACM class drivers. **This file should only be modified by HCC**.

2.4 Version File

The file **src/version/ver_usbd_cdcacm.h** contains the version number of this module. This version number is checked by all modules that use this module to ensure system consistency over upgrades.

2.5 Windows Driver Files

These files in the directory **driver/usb-device/class-drivers/cdc-ser** are provided for your convenience:

File	Description
usb_cdc_ser.inf	Windows .inf file for starting **usbser.sys** on Windows based PCs.
KB935892.reg	Windows specific patch for **usbser.sys** to work correctly on older versions of Windows.

3 Configuration Options

Set the system configuration options in the file **src/config/config_usbd_cdcacm.h**. This section lists the available configuration options and their default values.

USBD_CDCACM_NOTIFY_ON_IT_EP

Set this to 1 (the default) to send notifications over the Interrupt channel. This is the method defined by the standard. The built-in windows XP driver ignores notifications.

USBD_CDCACM_N_LINES

The number of serial lines to implement. The default is 1. The USB device configuration must match this value. The maximum allowed value is 4 lines.

USBD_CDCACM_DEFAULT_BPS

The default baud rate setting for serial lines. The default is 9600.

USBD_CDCACM_DEFAULT_STOP

The default number of stop bits for serial lines. The default is LCT_STOP_1.

USBD_CDCACM_DEFAULT_PARITY

The default parity type for serial lines. The default is LCT_PARITY_NONE.

USBD_CDCACM_DEFAULT_CLEN

The default number of data bits for serial lines. The default is 8.

USBD_CDCACM_DEFAULT_RXMODE

The default Receive mode after start. Possible settings are:

- USBD_CDCACM_RXMODE_NORMAL (the default).
- USBD_CDCACM_RXMODE_DIRECT.

Set this to USBD_CDCACM_RXMODE_DIRECT if USBD_CDCACM_RX_BUFFER_SIZE is 0.

USBD_CDCACM_CDC_RX_BUFFER_SIZE

The size of a receive buffer for Normal receive mode. Set this to the maximum Bulk IN endpoint packet size. Set it to 0 if Direct mode receive is used. The default is 512.

USBD_CDCACM_TASK_STACK_SIZE

The task stack size. The default is 1024.

USBD_CDCACM_BUF_NUM_LINE_0

The number of Receive buffers on CDC line 0. The default is 2.

USBD_CDCACM_BUF_NUM_LINE_1

The number of Receive buffers on CDC line 1. Keep the default of 0 if USBD_CDCACM_N_LINES < 2.

USBD_CDCACM_BUF_NUM_LINE_2

The number of Receive buffers on CDC line 2. Keep the default of 0 if USBD_CDCACM_N_LINES < 3.

USBD_CDCACM_BUF_NUM_LINE_3

The number of Receive buffers on CDC line 3. Keep the default of 0 if USBD_CDCACM_N_LINES < 4.

4 Application Programming Interface

This section documents the Application Programming Interface (API). It includes all the functions that are available to an application program.

4.1 Module Management

The functions are the following:

Function	Description
usbd_cdcacm_init()	Initializes the module and allocates the required resources.
usbd_cdcacm_start()	Starts the module.
usbd_cdcacm_stop()	Stops the module.
usbd_cdcacm_delete()	Deletes the module and releases the resources it used.

usbd_cdcacm_init

Use this function to initialize the CDC-ACM class driver module and allocate the required resources.

> **Note:** You must call this before any other function.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_init ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_start

Use this function to start the module.

> **Note:** You must call **usbd_cdcacm_init()** before this to initialize the module.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_start ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_stop

Use this function to stop the module.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_stop ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_delete

Use this function to delete the class driver and release the associated resources.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_delete ( void )
```

Arguments

Argument
None.

Return Values

Argument	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

4.2 Line Management

The functions are the following:

Function	Description
usbd_cdcacm_get_control_line_state()	Outputs a bitmask with the bit values of CLS_DTR and CLS_RTS set by the host.
usbd_cdcacm_get_line_coding()	Gets the current line configuration for the specified line.
usbd_cdcacm_set_line_coding()	Configures a serial line (its baud rate, bits, parity, and stop bits).
usbd_cdcacm_present()	Checks whether a device is connected.
usbd_cdcacm_receive()	Gets received data from the internal FIFO.
usbd_cdcacm_receive_start()	Starts reception directly into a user-provided buffer.
usbd_cdcacm_receive_status()	Gets the status of data reception on a channel.
usbd_cdcacm_send()	Sends data on a serial line.
usbd_cdcacm_send_status()	Gets the status of any previous transfers on a serial line. It is good practice to call this before **usbd_cdcacm_send()**.
usbd_cdcacm_set_lsflags()	Sets specified line status flags on a serial line.
usbd_cdcacm_set_rx_mode()	Sets receive mode (Normal or Direct) for a CDC-ACM line.
usbd_cdcacm_reg_ntf_fn()	Registers a notification function for a specified event type.

usbd_cdcacm_get_control_line_state

Use this function to output a bitmask with the bit values of CLS_DTR and CLS_RTS set by the host.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_get_control_line_state (
    const uint8_t      line,
    uint16_t * const   p_flags )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_flags	The control line state bitmask.	uint8_t *

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_get_line_coding

Use this function to get the current line configuration for the specified line.

Generally this function is used to get the line setup parameters (baud rate and so on) in the situation where the virtual serial line is being routed to a real physical serial port and the USB host is responsible for configuring the setup of that port.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_get_line_coding (
    const uint8_t                       line,
    t_usbd_cdcacm_line_coding * const   p_ln_coding )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_ln_coding	The line coding structure to be updated.	t_usbd_cdcacm_line_coding *

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_cdcacm_set_line_coding

Use this function to configure the specified serial line (its baud rate, data bits, stop bits and parity).

This function is used to report the current settings of the port to the USB host.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_set_line_coding (
    const uint8_t                       line,
    t_usbd_cdcacm_line_coding * const   p_ln_coding )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_ln_coding	The line coding structure.	t_usbd_cdcacm_line_coding *

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_cdcacm_present

Use this function to check whether a device is connected.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_present ( const uint8_t line )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t

Return Values

Return value	Description
USBD_CDCACM_ERR_NOT_PRESENT	No device is connected.
USBD_CDCACM_SUCCESS	A device is connected.

usbd_cdcacm_receive

Use this function to get received data from the internal FIFO.

Note:

- This function is intended for use with lines in Normal mode only. It returns an error if called for a line using Direct mode.
- In case more than one FIFO buffer is filled, this function reads as many bytes as possible (len_req), regardless of the internal buffer borders.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_receive (
    const uint8_t       line,
    uint8_t * const     p_buf,
    const uint32_t      len_req,
    uint32_t * const    p_len_done )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_buf	A pointer to the destination buffer.	uint8_t *
len_req	The requested length.	uint32_t
p_len_done	The number of bytes actually read from FIFO.	uint32_t *

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_receive_start

Use this function to start a reception directly into a user-provided buffer.

After this you can call **usbd_cdcacm_receive_status()** to find out how many bytes have been received in the buffer.

Note:

- This function must only be used for lines using Direct receive mode. It returns an error if called for a line using Normal mode.
- The CDC-ACM module only copies one packet of data into the user's buffer for each call to this function.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_receive_start (
    const uint8_t     line,
    uint8_t * const   p_buf,
    const uint32_t    len )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_buf	A pointer to the destination buffer.	uint8_t *
len	The requested length of the buffer.	uint32_t

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_cdcacm_receive_status

Use this function to get the status of data reception on the specified channel.

Note: This function returns the status for lines in both Direct receive mode and Normal receive mode.

- In Direct receive mode the number of bytes written to the direct buffer is stored in *p_len*.
- In Normal receive mode the total number of bytes pending in all RX FIFO buffers is stored in *p_len*.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_receive_status (
    const uint8_t      line,
    uint32_t * const   p_len )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_len	Where to store the status: • In Direct receive mode, the number of bytes written to the direct buffer. • In Normal receive mode, the total number of bytes pending in all RX FIFO buffers.	uint32_t *

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_cdcacm_send

Use this function to send data on the specified serial line.

Note:

- This call only succeeds if all previous sends have completed. If another send is in progress, the call returns USBD_CDCACM_ERROR, in which case you must repeat the send.
- You can check the status of any outstanding sends on the line by calling **usbd_cdcacm_send_status()** before initiating a send.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_send (
    const uint8_t     line,
    uint8_t * const   p_buf,
    const uint32_t    len )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
p_buf	A pointer to the buffer containing data to be sent.	uint8_t *
len	The number of bytes to send.	uint32_t

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_BUSY	The target is busy.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_send_status

Use this function to get the status of any previous transfers on the specified serial line. It is good practice to call this before using **usbd_cdcacm_send()**.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_send_status ( const uint8_t line )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.
USBD_CDCACM_BUSY	The send is still in progress.

usbd_cdcacm_set_lsflags

Use this function to set the specified line status flags on a serial line.

This function is generally only used where a real physical port is connected to the virtual serial line and the device must report state changes or error conditions on that physical port.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_set_lsflags (
    const uint8_t   line,
    const uint8_t   flags )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
flags	The flags to set, defined in LS_XXX.	uint8_t

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

usbd_cdcacm_set_rx_mode

Use this function to set receive mode (Normal or Direct) for a CDC-ACM line.

- In Normal receive mode data is received to the internal buffers of the CDC-ACM module. When requested with **usbd_cdcacm_receive()**, that data is copied to the user's buffer.
- In Direct receive mode data is received into the buffer you provide by using the **usbd_cdcacm_receive_start()** call.

Note: Call this function after **usbd_cdcacm_init()** and before **usbd_cdcacm_start()**.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_set_rx_mode (
    const uint8_t                line,
    const t_usbd_cdcacm_rx_mode  mode )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
mode	The RX mode to set.	t_usbd_cdcacm_rx_mode

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
Else	See Error Codes.

usbd_cdcacm_reg_ntf_fn

Use this function to register a notification function for a specified event notification type.

You must register a notification function for each event type that you want to receive notification for.

> **Note:** It is the user's responsibility to provide any notification functions required by the application. Providing such functions is optional.

Format

```
t_usbd_cdcacm_ret usbd_cdcacm_reg_ntf_fn (
    const t_usbd_cdcacm_ntf_type   ntf,
    const t_usbd_cdcacm_ntf_fn     ntf_fn )
```

Arguments

Argument	Description	Type
ntf	The notification type. Only one may be set.	t_usbd_cdcacm_ntf_type
ntf_fn	The notification function to use when the event occurs.	t_usbd_cdcacm_ntf_fn

Return Values

Return value	Description
USBD_CDCACM_SUCCESS	Successful execution.
USBD_CDCACM_ERROR	Operation failed.

4.3 Error Codes

If a function executes successfully, it returns with USBD_CDCACM_SUCCESS, a value of 0. This table shows the meaning of the error codes:

Return Value	Value	Description
USBD_CDCACM_SUCCESS	0	Operation successful.
USBD_CDCACM_ERROR	1	Operation failed.
USBD_CDCACM_ERR_NOT_PRESENT	2	Line is not available.
USBD_CDCACM_BUSY	3	Target is busy.

4.4 Types and Definitions

t_usbd_cdcacm_ntf_fn

The **t_usbh_ntf_fn** definition specifies the format of the notification function.

Register this function with the CDC-ACM module by calling **usbd_cdcacm_reg_ntf_fn()** .

Format

```
void ( * t_usbd_cdcacm_ntf_fn )(
    const uint8_t                   line,
    const t_usbd_cdcacm_ntf_type    ntf_type )
```

Arguments

Argument	Description	Type
line	The serial line's index.	uint8_t
ntf_type	The event being signaled (see Callback Notification).	t_usbd_cdcacm_ntf_type

Callback Notification

The callback notification definitions in the structure **t_usbd_cdcacm_ntf_type** are as follows:

Name	Description
USBD_CDCACM_NTF_CONNECT	Line connect.
USBD_CDCACM_NTF_DISCONNECT	Line disconnect.
USBD_CDCACM_NTF_SET_LINE_CODING	Set Line Coding received.
USBD_CDCACM_NTF_BREAK	Break.
USBD_CDCACM_NTF_RX	Receive finished.
USBD_CDCACM_NTF_TX	Transmit finished.
USBD_CDCACM_NTF_SET_CONTROL_LINE_STATE	Set Control Line State received.

t_usbd_cdcacm_line_coding

The line coding structure *t_usbd_cdcacm_line_coding* takes this form:

Element	Type	Description
cdcacm_line_bps	uint32_t	Bits per second.
cdcacm_line_n_data	uint8_t	Number of data bits.
cdcacm_line_n_stp	uint8_t	Number of stop bits.
cdcacm_line_parity	uint8_t	Parity.

Parity Definitions for line_coding_t

The definitions are as follows:

Definition	Description
LCT_PARITY_NONE	No parity.
LCT_PARITY_ODD	Odd parity.
LCT_PARITY_EVEN	Even parity.
LCT_PARITY_MARK	Mark parity.
LCT_PARITY_SPACE	Space parity.

Stop Bit Definitions

The definitions in *line_coding_t* are as follows:

Option	Meaning
LCT_STOP_1	One stop bit.
LCT_STOP_15	One and a half stop bits.
LCT_STOP_2	Two stop bits.

Control Line State Definitions

These are bit field definitions.

Bit Definition	Value	Description
CLS_DTR	1 << 0	Control line state flag: Data Terminal Ready.
CLS_RTS	1 << 1	Control line state flag: Request to Send.

Note: These are unique bits in a set of bit values and they can be OR'd together.

Line State Definitions

These are bit field definitions.

Bit Definition	Value	Description
LS_DCD	1 << 0	Data carrier detect. This is asserted when a connection has been established with remote equipment.
LS_DSR	1 << 1	Data Set Ready. This is asserted to indicate an active connection.
LS_BREAK	1 << 2	Break detection state.
LS_RING	1 << 3	Ring indicator state.
LS_FE	1 << 4	Framing error detected.
LS_PE	1 << 5	Parity error detected.
LS_OE	1 << 6	Overrun error detected.

Note: These are unique bits in a set of bit values and they can be OR'd together.

Receive Mode Definitions

The definitions in *t_usbd_cdcacm_rx_mode* are as follows:

Name	Description
USBD_CDCACM_RXMODE_NORMAL	Receive into the internal CDC buffer.
USBD_CDCACM_RXMODE_DIRECT	Receive directly into a user-provided buffer.

5 Integration

This section specifies the elements of this package that need porting, dependent on the target environment.

5.1 OS Abstraction Layer

All HCC modules use the OS Abstraction Layer (OAL) that allows the module to run seamlessly with a wide variety of RTOSes, or without an RTOS.

The class driver uses the following OAL components.

Resource	Requirement
Tasks	1 task to manage events and notifications: **usbd_cdcacm_task()**.
Mutexes	1
Events	1 event group.

5.2 PSP Porting

The Platform Support Package (PSP) is designed to hold all platform-specific functionality, either because it relies on specific features of a target system, or because this provides the most efficient or flexible solution for the developer. For full details of its functions and macros, see the *HCC Base Platform Support Package User Guide*.

The class driver makes use of the following standard PSP functions:

Function	Package	Component	Description
psp_memcpy()	psp_base	psp_string	Copies a block of memory. The result is a binary copy of the data.
psp_memset()	psp_base	psp_string	Sets the specified area of memory to the defined value.

The class driver makes use of the following standard PSP macros:

Macro	Package	Component	Description
PSP_RD_LE32	psp_base	psp_endianness	Reads a 32 bit value stored as little-endian from a memory location.
PSP_WR_LE32	psp_base	psp_endianness	Writes a 32 bit value to be stored as little-endian to a memory location.