

# Embedded USB Host CDC-ECM Class Driver User Guide

Version 1.20

For use with USBH CDC-ECM Class Driver versions  
2.06 and above

**Date:** 19-Jun-2017 10:39

All rights reserved. This document and the associated software are the sole property of HCC Embedded. Reproduction or duplication by any means of any portion of this document without the prior written consent of HCC Embedded is expressly forbidden.

HCC Embedded reserves the right to make changes to this document and to the related software at any time and without notice. The information in this document has been carefully checked for its accuracy; however, HCC Embedded makes no warranty relating to the correctness of this document.

# Table of Contents

System Overview	4
Introduction	4
Feature Check	5
Packages and Documents	6
Packages	6
Documents	6
Change History	7
Source File List	8
API Header File	8
Configuration File	8
Source Code File	8
Version File	8
Configuration Options	9
Application Programming Interface	10
Module Management Functions	10
usbh_cdcecm_init	11
usbh_cdcecm_start	12
usbh_cdcecm_stop	13
usbh_cdcecm_delete	14
Device Management Functions	15
usbh_cdcecm_send	16
usbh_cdcecm_receive	17
usbh_cdcecm_get_send_state	18
usbh_cdcecm_get_receive_state	19
usbh_cdcecm_rx_chars	20
usbh_cdcecm_get_connection_speed	21
usbh_cdcecm_get_eth_statistics	22
usbh_cdcecm_get_mac_address	23
usbh_cdcecm_get_max_segment_size	24
usbh_cdcecm_get_network_state	25
usbh_cdcecm_get_port_hdl	26
usbh_cdcecm_present	27
usbh_cdcecm_set_eth_mcast_filters	28
usbh_cdcecm_set_eth_packet_filter	29
usbh_cdcecm_register_ntf	30
Error Codes	31
Types and Definitions	32
t_usbh_ntf_fn	32
Notification Codes	32
Ethernet Packet Filters	33
Ethernet Statistics	34
Integration	36

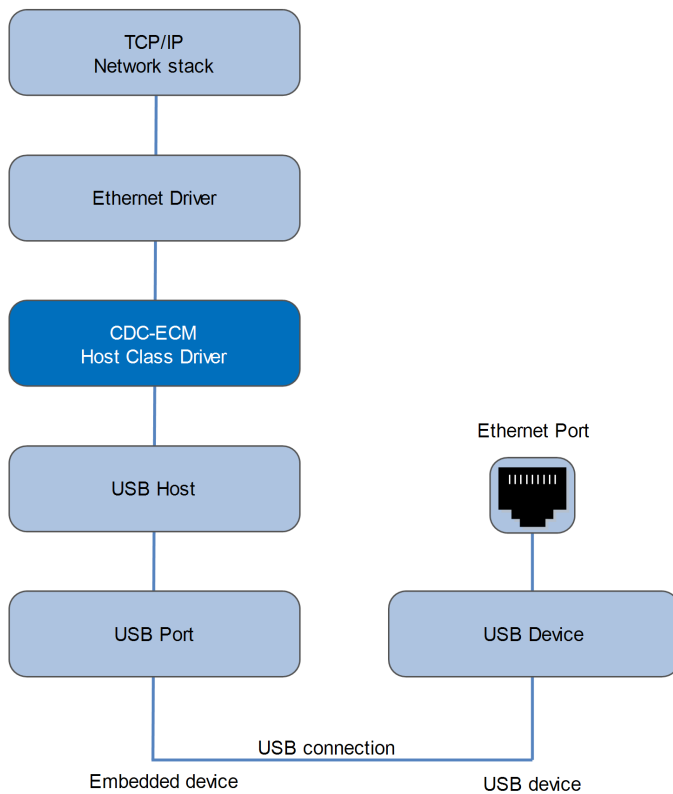
OS Abstraction Layer	36
PSP Porting	36
Sample Code	37
Initialization	37

# 1 System Overview

## 1.1 Introduction

This guide is for those who want to implement an embedded CDC-ECM (Control Device Class - Ethernet Control Module) host class driver. This presents the USB ECM device to the system as a network adapter.

The HCC CDC-ECM package provides a host class driver for a USB stack. The system structure is shown below:



The host implementation allows the host system to transfer Ethernet packets to/from a remote Ethernet port at the other end of the USB connection. It connects to a USB device that has an Ethernet port on it that connects to a real or virtual network.

The lower layer interface is designed to use HCC Embedded’s USB Host Interface Layer. This layer is standard over different host controller implementations; this means that the code is the same, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, refer to the [HCC USB Host Base System User Guide](#) that is shipped with the base system.

The package provides a set of API functions for controlling access to a device. These are described here, with separate sections for module and device management.

## 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.
- Compatible with all HCC USB host controllers.
- Supports all devices that conform to the USB CDC-ECM specification.
- Supports multiple devices connected simultaneously.
- Uses a system of callbacks for user-specified events.

---

## 1.3 Packages and Documents

---

### Packages

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

Package	Description
<code>hcc_base_doc</code>	This contains the two guides that will help you get started.
<code>usbh_base</code>	The USB host base package. This is the framework used by USB class drivers to communicate over USB using a specific USB host controller package.
<code>usbh_cd_cdc_ecm</code>	The USB device CDC-ECM host 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 USB Host Base System User Guide

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

#### HCC Embedded USB Host CDC-ECM Class Driver User Guide

This is this document.

## 1.4 Change History

---

This section describes past changes to this manual.

- To view or download earlier manuals, see [Archive: Embedded USB Host CDC-ECM Class Driver User Guide](#).
- For the history of changes made to the package code itself, see [History: usbh\\_cd\\_cdc\\_ecm](#).

The current version of this manual is 1.20. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.20	2017-06-19	2.06	New <i>Change History</i> format.
1.10	2016-04-20	2.06	Added function group descriptions to API.
1.00	2015-12-07	2.05	First 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_usbh_cdcecm.h` must be included by any application using the system. This 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_usbh_cdcecm.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 File

---

The file `src/usb-host/class-drivers/cdc-ecm/usbh_cdcecm.c` contains the main code for the class driver. **This file should only be modified by HCC.**

### 2.4 Version File

---

The file `src/version/ver_usbh_cdcecm.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.



## 3 Configuration Options

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

### **USBH\_CDCECM\_MAX\_UNITS**

The maximum number of CDC-ECM units the system can handle. This includes the maximum units per interface and the number of connected devices. The default is 2.

### **USBH\_CDCECM\_RXBUF\_COUNT**

The number of receive buffers for receiving in the background. When a device is plugged in, reception starts automatically. When data arrives the user gets a notification or polls for data. In the meantime reception to the next buffer can be started to keep the performance as high as possible. The default is 2.

### **USBH\_CDCECM\_RXBUF\_SIZE**

The size of a receive buffer. The default is 2048.

- For full speed systems, the minimum size is 64 and the value must be a multiple of 64.
- For high speed systems, the minimum size is 512 and the value must be a multiple of 512.

### **USBH\_CDCECM\_COMBUF\_SIZE**

The size of the COM (comms. interface) buffer. It is safe to keep this at the default of 64.

## 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 Functions

---

The functions are the following:

Function	Description
<code>usbh_cdcecm_init()</code>	Initializes the module and allocates the required resources.
<code>usbh_cdcecm_start()</code>	Starts the module.
<code>usbh_cdcecm_stop()</code>	Stops the module.
<code>usbh_cdcecm_delete()</code>	Deletes the module and releases the resources it used.

## usbh\_cdcecm\_init

Use this function to initialize the class driver and allocate the required resources.

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

### Format

```
int usbh_cdcecm_init ( void )
```

### Arguments

#### Parameter

None.

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_start

Use this function to start the class driver.

**Note:** You must call `usbh_cdcecm_init()` before this function.

### Format

```
int usbh_cdcecm_start ( void )
```

### Arguments

#### Parameter

None.

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_stop

Use this function to stop the class driver.

### Format

```
int usbh_cdcecm_stop ( void )
```

### Arguments

Parameter
None.

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_delete

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

### Format

```
int usbh_cdcecm_delete ( void )
```

### Arguments

Parameter
None.

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## 4.2 Device Management Functions

The functions are the following:

Function	Description
<code>usbh_cdcecm_send()</code>	Sends data through a channel.
<code>usbh_cdcecm_receive()</code>	Receives available data.
<code>usbh_cdcecm_get_send_state()</code>	Gets the completion code of the last send.
<code>usbh_cdcecm_get_receive_state()</code>	Gets the completion code of the last receive.
<code>usbh_cdcecm_rx_chars()</code>	Gets the number of characters in the receive buffer.
<code>usbh_cdcecm_get_connection_speed()</code>	Gets the current upstream and downstream connection speeds.
<code>usbh_cdcecm_get_eth_statistics()</code>	Gets Ethernet statistics.
<code>usbh_cdcecm_get_mac_address()</code>	Gets the MAC address of the ECM device.
<code>usbh_cdcecm_get_max_segment_size()</code>	Gets the maximum segment size.
<code>usbh_cdcecm_get_network_state()</code>	Gets the connection state of the network.
<code>usbh_cdcecm_get_port_hdl()</code>	Gets the port handle.
<code>usbh_cdcecm_present()</code>	Checks whether a channel is present.
<code>usbh_cdcecm_set_eth_mcast_filters()</code>	Sets Ethernet multicast filters.
<code>usbh_cdcecm_set_eth_packet_filter()</code>	Sets an Ethernet packet filter.
<code>usbh_cdcecm_register_ntf()</code>	Registers a notification function for a specified event type.

## usbh\_cdcecm\_send

Use this function to send data through a channel.

### Format

```
int usbh_cdcecm_send (
    t_usbh_unit_id  uid,
    uint8_t *      buf,
    uint32_t        length )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the data to send.	uint8_t *
length	The number of bytes to send.	uint32_t

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .



## usbh\_cdcecm\_receive

Use this function to receive available data.

### Format

```
int usbh_cdcecm_receive (
    t_usbh_unit_id  uid,
    uint8_t *      buf,
    uint32_t        max_length,
    uint32_t *      rlength )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
buf	A pointer to the buffer which is to receive the data.	uint8_t *
max_length	The maximum length of the receive buffer.	uint32_t
rlength	Where to put the number of bytes written to the buffer.	uint32_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_send\_state

Use this function to get the completion code of the last send.

**Note:** This function is only required if an event is used for send.

### Format

```
int usbh_cdcecm_get_send_state ( t_usbh_unit_id uid )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_receive\_state

Use this function to get the completion code of the last receive.

**Note:** This is not required generally since **usbh\_cdcecm\_receive()** returns the same information with the possible available data. If events are not used, you can poll this function to find whether data is available or not.

### Format

```
int usbh_cdcecm_get_receive_state ( t_usbh_unit_id uid )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_rx\_chars

Use this function to get the number of characters in the receive buffer.

**Note:** Only call this function if `usbh_cdcecm_get_receive_state()` indicates success, otherwise the result is not valid.

### Format

```
uint32_t usbh_cdcecm_rx_chars ( t_usbh_unit_id uid )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
The number of bytes in the receive buffer.	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_connection\_speed

Use this function to get the current upstream and downstream connection speeds.

### Format

```
int usbh_cdcecm_get_connection_speed (
    t_usbh_unit_id  uid,
    uint32_t *      usbrate,
    uint32_t *      dsbrate )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
usbrate	Where to write the upstream connection speed.	uint32_t *
dsbrate	Where to write the downstream connection speed.	uint32_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_eth\_statistics

Use this function to get Ethernet statistics.

**Note:** This call only returns a valid value if the requested statistic is supported by the ECM device.

### Format

```
int usbh_cdcecm_get_eth_statistics (
    t_usbh_unit_id  uid,
    uint16_t        stat,
    uint32_t *      val )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
stat	The requested statistic, one of the <a href="#">Ethernet Statistics</a> options.	uint16_t
val	Where to write the requested statistic.	uint32_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_mac\_address

Use this function to get the MAC address of the ECM device.

### Format

```
int usbh_cdcecm_get_mac_address (
    t_usbh_unit_id  uid,
    uint8_t *      mac_address )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
mac_address	Where to write the MAC address. This is a pointer to a 6 byte unsigned char array.	uint8_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_max\_segment\_size

Use this function to get the maximum segment size.

### Format

```
int usbh_cdcecm_get_max_segment_size (
    t_usbh_unit_id  uid,
    uint16_t *      max_segment_size )
```

### Arguments

Parameter	Description	Type
uid	The unit ID	t_usbh_unit_id
max_segment_size	Where to write the maximum segment size.	uint16_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .



## usbh\_cdcecm\_get\_network\_state

Use this function to get the connection state of the network.

### Format

```
int usbh_cdcecm_get_network_state (  
    t_usbh_unit_id  uid,  
    uint8_t *      state )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
state	Where to write the state (1 = connected, 0 = disconnected).	uint8_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_get\_port\_hdl

Use this function to get the port handle.

### Format

```
t_usbh_port_hdl usbh_cdcecm_get_port_hdl ( t_usbh_unit_id uid )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
The port handle.	Successful execution.
USBH_PORT_HDL_INVALID	Invalid port handle.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_present

Use this function to check whether a channel is present.

### Format

```
int usbh_cdcecm_present ( t_usbh_unit_id uid )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id

### Return Values

Return value	Description
0	No channel is present.
1	A channel is present.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_set\_eth\_mcast\_filters

Use this function to set Ethernet multicast filters.

### Format

```
int usbh_cdcecm_set_eth_mcast_filters (
    t_usbh_unit_id  uid,
    uint16_t        n,
    uint8_t *       filters )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
n	The number of filters.	uint16_t
filters	A list of 48 bit multicast filters in network byte order.	uint8_t *

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_set\_eth\_packet\_filter

Use this function to set an Ethernet packet filter.

### Format

```
int usbh_cdcecm_set_eth_packet_filter (
    t_usbh_unit_id  uid,
    uint16_t        flags )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
flags	A bitmap of packet type filters from the <a href="#">Ethernet Packet Filters</a> list.	uint16_t

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## usbh\_cdcecm\_register\_ntf

Use this function to register a notification function for the device.

When a device is connected or disconnected, or one of the specific events for this type of device occurs, the notification function is called.

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

### Format

```
int usbh_cdcecm_register_ntf (
    t_usbh_unit_id  uid,
    t_usbh_ntf      ntf,
    t_usbh_ntf_fn   ntf_fn )
```

### Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification ID.	t_usbh_ntf
ntf_fn	The notification function to be used when an event occurs.	<a href="#">t_usbh_ntf_fn</a>

### Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
Else	See <a href="#">Error Codes</a> .

## 4.3 Error Codes

If a function executes successfully it returns with a `USBH_SUCCESS` code, a value of 0. The following table shows the meaning of the error codes:

Return Code	Value	Description
<code>USBH_SUCCESS</code>	0	Successful execution.
<code>USBH_SHORT_PACKET</code>	1	IN transfer completed with short packet.
<code>USBH_PENDING</code>	2	Transfer still pending.
<code>USBH_ERR_BUSY</code>	3	Another transfer in progress.
<code>USBH_ERR_DIR</code>	4	Transfer direction error.
<code>USBH_ERR_TIMEOUT</code>	5	Transfer timed out.
<code>USBH_ERR_TRANSFER</code>	6	Transfer failed to complete.
<code>USBH_ERR_TRANSFER_FULL</code>	7	Cannot process more transfers.
<code>USBH_ERR_SUSPENDED</code>	8	Host controller is suspended.
<code>USBH_ERR_HC_HALTED</code>	9	Host controller is halted.
<code>USBH_ERR_REMOVED</code>	10	Transfer finished due to device removal.
<code>USBH_ERR_PERIODIC_LIST</code>	11	Periodic list error.
<code>USBH_ERR_RESET_REQUEST</code>	12	Reset request during enumeration.
<code>USBH_ERR_RESOURCE</code>	13	OS resource error.
<code>USBH_ERR_INVALID</code>	14	Invalid identifier/type (HC, EP HDL, and so on).
<code>USBH_ERR_NOT_AVAILABLE</code>	15	Item not available.
<code>USBH_ERR_INVALID_SIZE</code>	16	Invalid size.
<code>USBH_ERR_NOT_ALLOWED</code>	17	Operation not allowed.
<code>USBH_ERROR</code>	18	General error.

## 4.4 Types and Definitions

### t\_usbh\_ntf\_fn

The `t_usbh_ntf_fn` definition specifies the format of the notification function. It is defined in the USB host base system in the file `api_usb_host.h`.

#### Format

```
int ( * t_usbh_ntf_fn )(
    t_usbh_unit_id  uid,
    t_usbh_ntf      ntf )
```

#### Arguments

Element	Type	Description
uid	t_usbh_unit_id	The unit ID.
ntf	t_usbh_ntf	The <a href="#">notification code</a> .

### Notification Codes

The standard notification codes shown below are defined in the USB host base system in the file `api_usb_host.h`.

Notification	Value	Description
USBH_NTF_CONNECT	1	Connection notification code.
USBH_NTF_DISCONNECT	2	Disconnection notification code.

The additional notification codes provided by this module are as follows:

Notification	Value	Description
USBH_NTF_CDCECM_RX	USBH_NTF_CD_BASE + 0	Data received notification.
USBH_NTF_CDCECM_TX	USBH_NTF_CD_BASE + 1	Data sent notification.
USBH_NTF_CDCECM_NTF	USBH_NTF_CD_BASE + 2	CDC notification.



## Ethernet Packet Filters

The Ethernet packet filters are used by the `usbh_cdcecm_set_eth_packet_filter()` function:

Code	Value	Description
USBH_CDCECM_PACKET_TYPE_PROMISCUOUS	$1U \ll 0$	Promiscuous.
USBH_CDCECM_PACKET_TYPE_ALL_MULTICAST	$1U \ll 1$	Multicast all.
USBH_CDCECM_PACKET_TYPE_DIRECTED	$1U \ll 2$	Directed.
USBH_CDCECM_PACKET_TYPE_BROADCAST	$1U \ll 3$	Broadcast.
USBH_CDCECM_PACKET_TYPE_MULTICAST	$1U \ll 4$	Multicast.

**Note:** These are unique bits in a set of bit values.

## Ethernet Statistics

These are the options for the *stat* parameter of the **usbh\_cdcecm\_get\_eth\_statistics()** function:

Option	Value	Description
USBH_CDCECM_XMIT_OK	1	Successful transmits.
USBH_CDCECM_RCV_OK	2	Successful receives.
USBH_CDCECM_XMIT_ERROR	3	Transmit errors.
USBH_CDCECM_RCV_ERROR	4	Receive errors.
USBH_CDCECM_RCV_NO_BUFFER	5	Lost because receive buffer full.
USBH_CDCECM_DIRECTED_BYTES_XMIT	6	Directed bytes sent.
USBH_CDCECM_DIRECTED_FRAMES_XMIT	7	Directed frames sent.
USBH_CDCECM_MULTICAST_BYTES_XMIT	8	Multicast bytes sent.
USBH_CDCECM_MULTICAST_FRAMES_XMIT	9	Multicast frames sent.
USBH_CDCECM_BROADCAST_BYTES_XMIT	10	Broadcast bytes sent.
USBH_CDCECM_BROADCAST_FRAMES_XMIT	11	Broadcast frames sent.
USBH_CDCECM_DIRECTED_BYTES_RCV	12	Directed bytes received.
USBH_CDCECM_DIRECTED_FRAMES_RCV	13	Directed frames received.
USBH_CDCECM_MULTICAST_BYTES_RCV	14	Multicast bytes received.
USBH_CDCECM_MULTICAST_FRAMES_RCV	15	Multicast frames received.
USBH_CDCECM_BROADCAST_BYTES_RCV	16	Broadcast bytes received.
USBH_CDCECM_BROADCAST_FRAMES_RCV	17	Broadcast frames received.
USBH_CDCECM_RCV_CRC_ERROR	18	Receive CRC errors.
USBH_CDCECM_TRANSMIT_QUEUE_LENGTH	19	Length of transmit queue.
USBH_CDCECM_RCV_ERROR_ALIGNMENT	20	Receive alignment errors.
USBH_CDCECM_XMIT_ONE_COLLISION	21	Transmit single collisions.
USBH_CDCECM_XMIT_MORE_COLLISIONS	22	Transmit multiple collisions.
USBH_CDCECM_XMIT_DEFERRED	23	Transmit deferrals.
USBH_CDCECM_XMIT_MAX_COLLISIONS	24	Transmit collisions.
USBH_CDCECM_RCV_OVERRUN	25	Receive overruns.

Option	Value	Description
USBH_CDCECM_XMIT_UNDERRUN	26	Transmit underruns.
USBH_CDCECM_XMIT_HEARTBEAT_FAILURE	27	Transmit heartbeat failures.
USBH_CDCECM_XMIT_TIMES_CRIS_LOST	28	Transmit CRS lost.
USBH_CDCECM_XMIT_LATE_COLLISIONS	29	Transmit late collisions.

## 5 Integration

This section specifies the elements of this package that need porting, depending 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 CDC-ECM class driver uses the following OAL components:

OAL Resource	Number Required
Tasks	0
Mutexes	1
Events	0

### 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.

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

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

## 6 Sample Code

This section shows example code for the class driver.

### 6.1 Initialization

This example shows the code used to initialize a USB host with the class driver.

```
/*
** Initialize USB host with CDC-ECM class driver.
*/

int usb_host_init ( void )
{
    int rc;
    rc = hcc_mem_init();

    /* Initialize USB host module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_init();
    }

    /* Initialize specific USB host controller */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_stm32uh_hc, 0 );
    }

    /* Initialize the CDC-ECM Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdcecm_init();
    }

    /* Start the CDC-ECM Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdcecm_start();
    }

    /* Start the USB host stack */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_start(); /* Start the USB host */
    }

    return rc;
} /* usb_host_init */
```