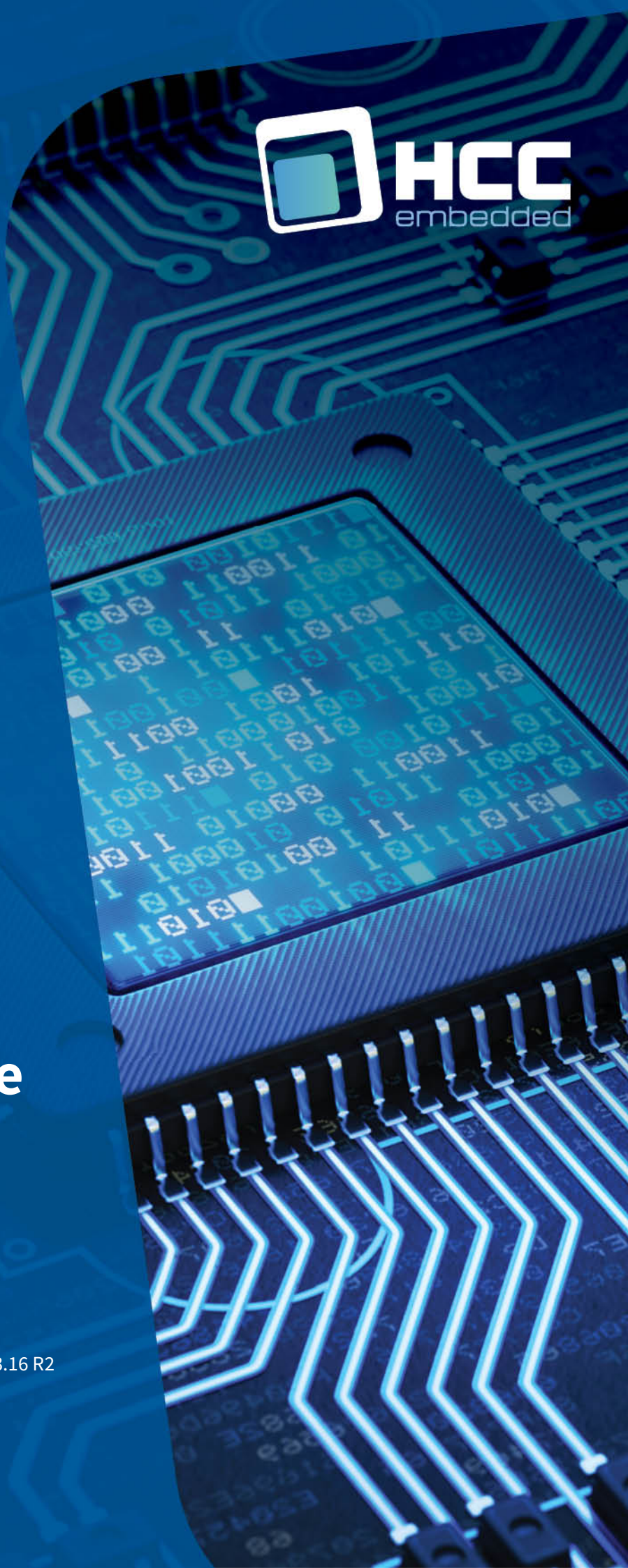


USB Host Base System User Guide

Version 1.80

For use with USB Host Base System versions 3.16 R2 and above

Table of Contents

1 System Overview

This chapter contains the fundamental information for this module.

The component sections are as follows:

- Introduction – describes the main elements of the module.
- Feature Check – summarizes the main features of the module as bullet points.
- Packages and Documents – the *Packages* section lists the packages that you need in order to use this module. The *Documents* section lists the relevant user guides.
- Change History – lists the earlier versions of this manual, giving the software version that each manual describes.

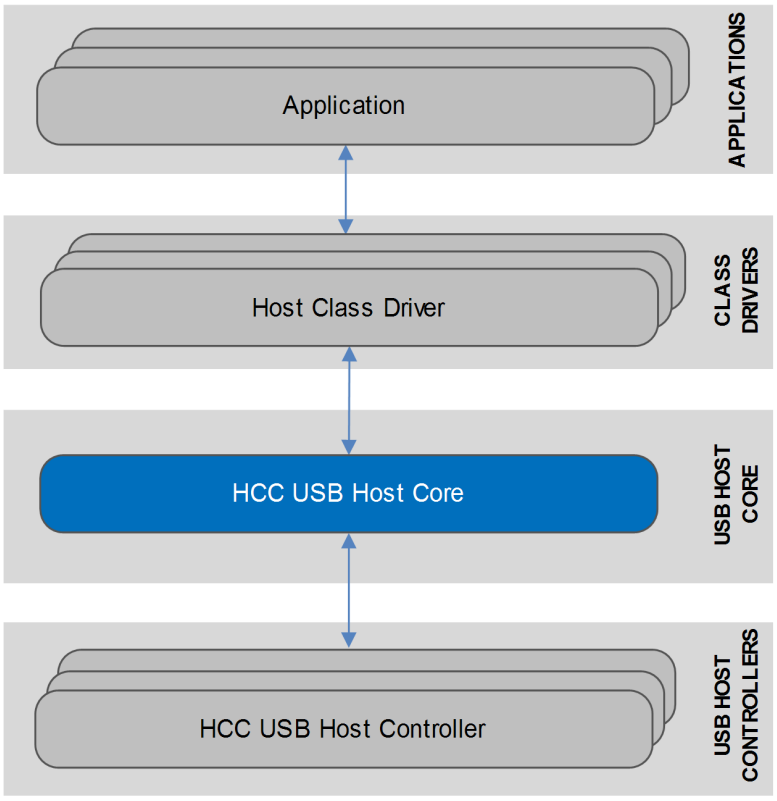
1.1 Introduction

This guide is for those who want to implement an embedded USB host stack using one or more HCC host controllers with one or more host class drivers. The HCC Embedded USB Host base system forms the core of HCC's USB host stack solution.

The system supports any number of USB host controllers, each of which may have multiple units. Supported host controllers include any combination of Enhanced Host Controller Interface (EHCI), Open Host Controller Interface (OHCI), and proprietary host controller types. HCC has many host controller implementations and can add new host controllers on request.

The system provides an interface for any number of USB host class drivers to communicate with their corresponding USB device class drivers. The system supports all USB transfer types (control, interrupt, bulk, and isochronous). This manual also defines how each USB host class driver using this system should be structured.

The system structure is shown in the diagram below:



HCC provides a wide range of USB class drivers to use with the system and other supporting software, such as file systems for Mass Storage (MST) solutions. Any number of HCC class drivers can be added to the system.

1.2 Feature Check

The main features of the system are the following:

- Conforms to the HCC Advanced Embedded Framework.
- Designed for integration with both RTOS and non-RTOS based systems.
- Supports multiple host controllers, each with multiple instances.
- Provides a range of HCC host controllers for EHCI, OHCI, and other types.
- Supports internal or external host controllers.
- Supports all USB endpoint types: control, bulk, interrupt, and isochronous.
- Supports USB low, full, and high speed interfaces.
- Supports external hubs.
- Supports USB On-The-Go (OTG) Service Request Protocol (SRP) and Host Negotiation Protocol (HNP).
- Supported by a range of standard HCC class drivers.
- Supports USB device test modes.
- Provides API functions for reading USB device properties.

1.3 Packages and Documents

Packages

This table lists the packages that you need in order to use this module, and also optional modules that may interact with this module, depending on your particular system's design:

Package	Description
hcc_base_doc	This contains the two guides that will help you get started.
usbh_base	The package described in this guide, the USB host base system package that host class drivers use for communication.
usbh_cd_xxxx	Individual USB host class drivers that are included as required by the system design (see the table below).
usbh_drv_xxxx	Individual USB host controllers as required (see the table below). The system must include at least one of these and supports the use of multiple controllers of different types.
psp_template_base	The base Platform Support Package (PSP).
oal_base	The base OS Abstraction Layer (OAL) package.
util_hcc_mem	The HCC memory management utility; this must be ported if the target system uses cached memory.

One or more of the following class drivers may be included in a system:

Package	Class driver
usbh_cd_audio	Audio 1.0.
usbh_cd_ccid	Chip Card Interface Device (CCID) USB devices.
usbh_cd_cdc_acm	Communications Device Class - Abstract Control Model (CDC-ACM).
usbh_cd_cdc_ecm	Communications Device Class - Ethernet Control Model (CDC-ECM).
usbh_cd_cdc_eem	Communications Device Class - Ethernet Emulation Model (CDC-EEM).
usbh_cd_cdc_ncm	Communications Device Class - Network Control Model (CDC-NCM).
usbh_cd_cp210x	Class Driver for Silicon Labs CP2102/CP2103/CP2105/CP2109 devices.
usbd_cd_ftdi	Future Technology Devices International (FTDI) devices.
usbd_cd_hid	Human Interface Device (HID) class driver for devices including keyboards, joysticks, mice, and "generic devices" (pointers, buttons, sliders, and so on).
usbd_cd_hub	USB hub.
usbh_cd_microchip_lan7500	Class Driver for Microchip LAN7500 and LAN9500.
usbd_cd_midi	Musical Instrument Digital Interface (MIDI).
usbd_cd_mst	Mass Storage (MST).
usbd_cd_printer	Printer.
usbh_cd_printer_fujitsu_ftp6xx	Fujitsu FTP-6xx Printer.
usbd_cd_raw	Raw or Vendor-specific.
usbd_cd_rndis	Remote Network Driver Interface Standard (RNDIS).

This is the current list of supported class drivers; contact sales@hcc-embedded.com for possible updates.

One or more of the following host controllers may be included in a system:

Package	Host Controller
usbh_drv_ehci	Enhanced Host Controller Interface (EHCI).
usbh_drv_ohci	Open Host Controller Interface (OHCI).
usbh_drv_atmel	Atmel Host Controller
usbh_drv_atmel_usbc	Atmel USBC Host Controller
usbh_drv_isp1161	USB ISP1161 Host Controller.
usbh_drv_isp1362	USB ISP1362 Host Controller.
usbh_drvc_lms	LM3S and TM4C devices.
usbh_drv_max3421	MAX3421 devices.
usbh_drv_musb_cppi	Devices that have the Mentor Graphics® USB core and CPPI DMA (DM814x/DM816x).
usbh_drv_musb_dma	Analog Devices Blackfin® BF60x MCUs with the Mentor Graphics® MUSB device core.
usbh_drv_renesas	Renesas devices.
usbh_drv_isp176x, usbc_drv_isp1582_isp176x	USB SAF1761 Host Controller.
usbh_drv_stm32_otg	USB STM32 OTG Host Controller.
usbh_drv_synopsys_otg	Synopsys® OTG devices.
usbh_drv_template	USB Template host controller for new developments with the USB host stack.
usbh_drv_vusb	USB VUSB Host Controller.

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 is this document.

1.4 Change History

This section describes past changes to this manual.

- To download this manual or a PDF describing an earlier software version, see USB Host PDFs.
- For the history of changes made to the package code itself, see History: usbh_base.

The current version of this manual is 1.80. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.80	2019-07-22	3.16 R2	Changed default value of USBH_OTG_SRPHNP_SUPPORT back to 0.
1.70	2019-07-02	3.16	Changed default value of USBH_OTG_SRPHNP_SUPPORT to 1.
1.60	2019-05-31	3.15	Added USBH_OTG_SRPHNP_SUPPORT configuration option for SRP and HNP support.
1.50	2017-09-07	3.11	Updated *Packages* list and subsidiary tables.
1.40	2017-06-19	3.11	New *Change History* format.
1.30	2015-11-16	3.09	Added *String Descriptors*, other small changes.
1.20	2015-07-17	3.09	Added many more functions.
1.10	2015-05-12	3.08	Added class driver table to *System Overview*.
1.00	2015-03-06	3.08	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 Files

The file **src/api/api_usb_host.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_usb_host.h** contains all the configurable parameters. Configure this as required. For details of these options, see Configuration Options.

2.3 Version File

The file **src/version/ver_usb_host.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.

2.4 USB Host System

These files are in **src/usb-host/usb-driver/common**. **These files should only be modified by HCC**.

File	Description
usb_host.c	Main code for USB host.
usb_host.h	Internal main header file.
usbh_desc.h	USB descriptors header file.
usbh_hc.h	Host controller descriptor header file.
usbh_hdl.h	USB host handlers header file.
usbh_utils.c	Utility functions.
usbh_utils.h	Utility header file.

3 Configuration Options

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

USBH_PMGR_TASK_STACK_SIZE

The stack size of the port manager task. The default is 1024.

USBH_PMGR_REQ_TASK_STACK_SIZE

The stack size of the port manager request task. The default is 1024.

USBH_MAX_HOST_CONTROLLERS

The maximum number of host controllers supported. The default is 1.

This depends on the target platform and should reflect the number of host controllers available. For example, if an external ISP1561 is used, this has to be 3 (2 OHCI controllers + 1 EHCI).

USBH_MAX_CLASS_DRIVERS

The maximum number of class drivers supported. The default is 2.

USBH_MAX_EXT_HUBS

The maximum number of external hubs supported. The default is 2.

USBH_MAX_PORTS

The maximum number of host ports supported. The default is 10. Calculate this as:

- number of root hub ports + total number of ports available on the attached hub(s).

USBH_MAX_INTERFACE_PER_DEVICE

The maximum number of interfaces on a device. The default is 2.

USBH_TIMEOUT_CHECK_INTERVAL

The check interval of the timeout task which handles timeouts for all active USB transfers in the system. This task wakes up after every USBH_TIMEOUT_CHECK_INTERVAL milliseconds and checks whether the transfer has timed out. The default is 10 ms.

Increasing this number reduces load on the system but provides less accurate timeouts (though this is not really critical). Note that:

- Setting it to 0 disables timeout handling completely.
- Setting it to 1 provides exact timeout handling but a task that wakes up every ms.

USBH_CFG_DESC_BUF_SIZE

The configuration descriptor buffer size. This buffer contains part of the configuration descriptor processed at enumeration. The default is 128, which should always be enough.

Set this to:

- maximum EP0 size (64) + the maximum size of a descriptor within the configuration descriptor.

USBH_OTG_SRPHNP_SUPPORT

Set this to 1 if On-The-Go Service Request Protocol (SRP) and Host Negotiation Protocol (HNP) are supported. The default is 0.

4 Application Programming Interface

This section documents the Application Programming Interface (API). There are several sets of functions:

- Module management – these initialize, start, stop, and delete the USB host stack.
- Host controller management – these initialize, start, stop, and delete a host controller.
- Class driver management – these create and delete a class driver.
- Application functions – these are used by applications (normally USB class drivers) to communicate using a USB host controller.
- Callback functions – these are used to register user-written callback functions.

4.1 Module Management

The functions are the following:

Function	Description
usbh_init()	Initializes the host stack and allocates the required resources.
usbh_start()	Starts the host stack.
usbh_stop()	Stops the host stack.
usbh_delete()	Deletes the host stack and releases the resources it used.

usbh_init

Use this function to initialize the USB host stack.

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

Format

```
int usbh_init ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_start

Use this function to start the USB host stack.

> **Note:** You must call **usbh_init()** before this to initialize the stack.

Format

```
int usbh_start ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_stop

Use this function to stop the USB host stack.

Format

```
int usbh_stop ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_delete

Use this function to delete the USB host stack and release the associated resources.

Format

```
int usbh_delete ( void )
```

Arguments

Parameter
None.

Return Values

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

4.2 Host Controller Management

The functions are the following:

Function	Description
usbh_hc_init()	Initializes the host controller and allocates the required resources.
usbh_hc_start()	Starts the host controller.
usbh_hc_stop()	Stops the host controller.
usbh_hc_delete()	Deletes the host controller and releases the resources it used.

usbh_hc_init

Use this function to add a USB host controller to the system.

You can add up to USBH_MAX_HOST_CONTROLLERS host controllers to the system; each host controller is addressed by the ID given in this call.

Format

```
int usbh_hc_init (
    uint8_t         id,
    void *          hc,
    t_usbh_unit_id  unit )
```

Arguments

Parameter	Description	Type
id	The ID of the host controller.	uint8_t
hc	A pointer to the host controller descriptor. To obtain this, include the specific host controller API file and add "**_hc**" to the end of its name. For example, you would change **api_usbh_ohci.h** to **api_usbh_ohci_hc.h**.	void *
unit	The unit number of the host controller. For example, one host controller might be able to handle several controllers of the same type.	t_usbh_unit_id

Return Values

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

usbh_hc_start

Use this function to start a USB host controller.

> **Note:** You must call **usbh_hc_init()** before this to initialize the host controller.

Format

```
int usbh_hc_start ( uint8_t id )
```

Arguments

Parameter	Description	Type
id	The ID of the host controller.	uint8_t

Return Values

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

usbh_hc_stop

Use this function to stop a USB host controller.

Format

```
int usbh_hc_stop ( uint8_t id )
```

Arguments

Parameter	Description	Type
id	The ID of the host controller.	uint8_t

Return Values

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

usbh_hc_delete

Use this function to delete a USB host controller and release the associated resources.

Format

```
int usbh_hc_delete ( uint8_t id )
```

Arguments

Parameter	Description	Type
id	The ID of the host controller.	uint8_t

Return Values

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

4.3 Class Driver Management

The functions are the following:

Function	Description
usbh_add_cd()	The function each class driver (except a hub class driver) must call to register itself with the host system.
usbh_delete_cd()	Deletes a class driver (but not a hub class driver).
usbh_add_hub()	The function a hub class driver must call to register itself with the host system.
usbh_delete_hub()	Deletes a hub class driver.

Note:

- These functions are only to be used by those who are developing class drivers.
- These functions are only called by class drivers that include the **usbh_host.h** header file that includes the prototypes of these functions.

usbh_add_cd

This is the function each class driver (except for a hub class driver) must call to register itself with the host system.

When the class driver registers itself with the host system, it uses the t_usbh_cd structure to provide a set of callback functions for the host to call if certain events occur. The class driver must provide these functions, although some of them may be null.

The class driver implementation calls the **usbh_add_cd()** function with the set of callbacks it has provided for the host to carry out the functions of the class driver.

Format

```
int usbh_add_cd ( const t_usbh_cd * cd )
```

Arguments

Parameter	Description	Type
cd	A pointer to the class driver callback information.	t_usbh_cd *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The class driver information is not available.

usbh_delete_cd

Use this function to delete a class driver (but not a hub class driver).

Format

```
int usbh_delete_cd ( const t_usbh_cd * cd )
```

Arguments

Parameter	Description	Type
cd	A pointer to the class driver information.	t_usbh_cd *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERR_NOT_AVAILABLE	The class driver information is not available.

usbh_add_hub

This is the function a hub class driver must call to register itself with the host system.

When the class driver registers itself with the host system, it uses the t_usbh_hub structure to provide a set of callback functions for the host to call if certain events occur. The class driver must provide these functions, although some of them may be null.

The class driver implementation calls the **usbh_add_hub()** function with the set of callbacks it has provided for the host to carry out the functions of the class driver.

> **Note:** This function can only be called from the port manager task, so mutex protection is not required.

Format

```
t_usbh_hub_hdl usbh_add_hub (
    t_usbh_ifc_hdl   ifc_hdl,
    t_usbh_hub *     hub )
```

Arguments

Parameter	Description	Type
ifc_hdl	The interface handle.	t_usbh_ifc_hdl
hub	A pointer to the hub class driver callback information.	t_usbh_hub *

Return Values

Return value	Description
t_usbh_hub_hdl	The hub handle.
NULL	No more entries are available.

usbh_delete_hub

Use this function to delete a hub class driver.

> **Note:** This function can only be called from the port manager task, so mutex protection is not required.

Format

```
int usbh_delete_hub ( t_usbh_hub_hdl hub_hdl )
```

Arguments

Parameter	Description	Type
hub_hdl	The hub handle.	t_usbh_hub_hdl

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Deletion of one of the devices previously attached to the hub failed.

4.4 Application Functions

These are functions used by applications (normally USB class drivers) to communicate using a USB host controller.

Function	Description
usbh_delay()	Delays for a number of milliseconds.
usbh_get_port_inf()	Gets port information identified by the port handle.
usbh_get_port_inf_port()	Gets port information identified by host controller ID and port path.
usbh_get_string()	Gets a string descriptor.
usbh_reenumerate()	Requests reenumeration of a device.
usbh_suspend()	Called from the host, suspends a device.
usbh_resume()	Resumes a device previously suspended by the host.
usbh_test_mode_device()	Puts a device into test mode.
usbh_test_mode_port()	Enters test mode on a high speed root hub or an external hub port.

usbh_delay

Use this function to delay for a number of milliseconds.

This function checks all the registered host controllers and uses the frame counter of the first one it finds that has started.

> **Note:** This is guaranteed to work only if all host controllers have the frame counter running, even if no device is connected.

Format

```
void usbh_delay ( uint32_t ms )
```

Arguments

Parameter	Description	Type
ms	The number of milliseconds to delay for.	uint32_t

Return Values

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

usbh_get_port_inf

Use this function to get port information identified by the port handle.

Format

```
int usbh_get_port_inf(
    t_usbh_port_hdl     port_hdl,
    t_usbh_port_inf *   p_port_inf )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
p_port_inf	Where to write the returned port information.	t_usbh_port_inf *

Return Values

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

usbh_get_port_inf_port

Use this function to get port information identified by host controller ID and port path.

Format

```
int usbh_get_port_inf_port (
    uint8_t             id,
    uint8_t *           p_path,
    uint8_t             path_len,
    t_usbh_port_inf *   p_port_inf )
```

Arguments

Parameter	Description	Type
id	The host controller ID.	uint8_t
p_path	A pointer to the path to the port.	uint8_t *
path_len	The length of the path.	uint8_t
p_port_inf	Where to write the returned port information.	t_usbh_port_inf *

Return Values

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

usbh_get_string

Use this function to get a string descriptor.

Format

```
int usbh_get_string (
    t_usbh_port_hdl  port_hdl,
    uint16_t         idx,
    uint16_t         lang_id,
    uint8_t *        str,
    uint16_t         mlen )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle of the device.	t_usbh_port_hdl
idx	The string index.	uint16_t
lang_id	The language ID.	uint16_t
str	Where to write the string descriptor.	uint8_t *
mlen	The maximum length of the string.	uint16_t

Return Values

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

usbh_reenumerate

Use this function to request re-enumeration of a device.

You can use this function if the software decides to use a configuration index different from the active one.

Format

```
int usbh_reenumerate ( t_usbh_port_hdl port_hdl )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl

Return Values

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

usbh_suspend

Use this function from the host to suspend a device.

When this is called the device must suspend itself and draw less than 500µA from the bus. You can use **usbh_resume()** to re-activate the device.

Format

```
int usbh_suspend (
    t_usbh_port_hdl   port_hdl,
    uint8_t           rwkup_en )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
rwkup_en	Set this to enable the device's remote wakeup capability. If the remote wakeup flag is set the device is allowed to wake itself up normally in response to a key being pressed or similar input. The device achieves this wakeup by using a special signalling sequence on the bus, which the host controller can detect, to tell the host it is active again.	uint8_t

Return Values

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

usbh_resume

Use this function to resume a device previously suspended by the host.

Format

```
int usbh_resume ( t_usbh_port_hdl port_hdl )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl

Return Values

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

usbh_test_mode_device

Use this function to put a device into test mode.

Format

```
int usbh_test_mode_device (
    t_usbh_port_hdl    port_hdl,
    t_usbh_test_mode   mode )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
mode	The test mode.	t_usbh_test_mode

Return Values

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

usbh_test_mode_port

Use this function to enter test mode on a high speed root hub or an external hub port.

The port is identified by the host controller ID and port path.

Format

```
int usbh_test_mode_port (
    uint8_t             id,
    uint8_t *           p_path,
    uint8_t             path_len,
    t_usbh_test_mode    mode )
```

Arguments

Parameter	Description	Type
id	The host controller ID.	uint8_t
p_path	A pointer to the path to the port. For example, "1,2,4" means the fourth port of the external hub, connected to the second port of another external hub which is connected to the first port of the root hub.	uint8_t *
path_len	The number of entries in the path array.	uint8_t
mode	The test mode.	t_usbh_test_mode

Return Values

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

4.5 Callback Functions

Use these to register user-written callback functions.

Note:

- It is the class driver's responsibility to provide the **t_usbh_cd_xxx()** or **t_usbh_hub_xxx()** functions.
- It is the user's responsibility to provide the **usbh_register_xxx()** functions.

The functions are the following:

Function	Description
t_usbh_cd_connect_fn()	Connects a device.
t_usbh_cd_disconnect_fn()	Disconnects a device.
t_usbh_cd_check_fn()	Checks the device's configuration.
t_usbh_cd_descriptor_fn()	Gets a device descriptor.
t_usbh_hub_fn()	Disables, suspends, or resumes a hub.
t_usbh_hub_reset_fn()	Resets a hub.
t_usbh_hub_scan_fn()	Scans for a device status change.
t_usbh_hub_test_fn()	Sets a hub's test mode.
usbh_register_config_select_cb()	Registers a "select configuration" callback function.
usbh_register_enum_failed_cb()	Registers an "enumeration failed" callback function.

t_usbh_cd_connect_fn

This defines the class driver callback function that may be called to connect a device.

Format

```
int ( * t_usbh_cd_connect_fn )(
    t_usbh_dev_hdl   dev_hdl,
    t_usbh_ifc_hdl   ifc_hdl )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
ifc_hdl	A pointer to the interface handle.	t_usbh_ifc_hdl

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_cd_disconnect_fn

This defines the class driver callback function that may be called to disconnect a device.

Format

```
int ( * t_usbh_cd_disconnect_fn )( t_usbh_dev_hdl   dev_hdl )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_cd_check_fn

This defines the class driver callback function that may be called to check the device's configuration.

Format

```
t_usbh_ep_hdl *( * t_usbh_cd_check_fn )(
    t_usbh_port_hdl    port_hdl,
    t_usbh_dsc_par *   dsc,
    t_usbh_ep_hdl      ep0_hdl,
    t_usbh_dev_hdl *   dev_hdl )
```

Arguments

Argument	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
dsc	A pointer to the descriptor.	t_usbh_dsc_par *
ep0_hdl	The endpoint 0 handle.	t_usbh_ep_hdl
dev_hdl	A pointer to the device handle.	t_usbh_dev_hdl *

Return Values

A pointer to the endpoint handle.

t_usbh_cd_descriptor_fn

This defines the class driver callback function that may be called to get a device descriptor.

Format

```
int * t_usbh_cd_descriptor_fn )(
    t_usbh_dev_hdl   dev_hdl,
    uint8_t *        dsc )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
dsc	A pointer to the descriptor.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_hub_fn

This defines the class driver callback function that may be called to disable, suspend, or resume a hub.

Format

```
int ( * t_usbh_hub_fn )(
    t_usbh_dev_hdl   dev_hdl,
    uint8_t          port )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
port	The port the hub is connected to.	uint8_t

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_hub_reset_fn

This defines the class driver callback function that may be called to reset a hub.

Format

```
int ( * t_usbh_hub_reset_fn )(
    t_usbh_dev_hdl   dev_hdl,
    uint8_t          port,
    uint8_t *        speed )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
port	The port the hub is connected to.	uint8_t
speed	A pointer to the speed of the new connection.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_hub_scan_fn

This defines the class driver callback function that may be called to scan for a device status change.

Format

```
int ( * t_usbh_hub_scan_fn )(
    t_usbh_dev_hdl     dev_hdl,
    t_hub_scan_inf *   hsi,
    uint8_t *          hsi_cnt )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
hsi	A pointer to the scan information.	t_hub_scan_inf *
hsi_cnt	A pointer to the scan count.	uint8_t *

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

t_usbh_hub_test_fn

This defines the class driver callback function that may be called to set a hub's test mode.

Format

```
int ( * t_usbh_hub_test_fn )(
    t_usbh_dev_hdl     dev_hdl,
    uint8_t            port,
    t_usbh_test_mode   mode )
```

Arguments

Argument	Description	Type
dev_hdl	The device handle.	t_usbh_dev_hdl
port	The port the hub is connected to.	uint8_t
mode	The test mode.	t_usbh_test_mode

Return Values

Return value	Description
USBH_SUCCESS	Successful execution.
USBH_ERROR	Operation failed.

usbh_register_config_select_cb

Use this function to register a "select configuration" callback function.

After it is registered, the system calls this function every time a device is enumerated. This allows you to select which configuration to activate on the device, based on its vendor and product IDs.

Format

```
int usbh_register_config_select_cb ( t_usbh_config_select_cb p_cb )
```

Arguments

Parameter	Description	Type
p_cb	The callback function.	t_usbh_config_select_cb

Return Values

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

usbh_register_enum_failed_cb

Use this function to register an "enumeration failed" callback function.

After it is registered, the enumeration failed callback is executed if a device is attached without any class driver being able to mount any interface on it.

Format

```
int usbh_register_enum_failed_cb ( t_usbh_enum_failed_cb p_cb );
```

Arguments

Parameter	Description	Type
p_cb	The callback function.	t_usbh_enum_failed_cb

Return Values

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

4.6 Error Codes

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

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

4.7 Types and Definitions

This section describes the main elements that are defined in the API Header file.

t_usbh_port_inf

This information is returned by the **usbh_get_port_inf()** function.

Element	Type	Description
state	uint16_t	See Connection States.
hc_uid	uint8_t	The host controller ID.
path_len	uint8_t	The number of elements in the path.
path[USBH_MAX_EXT_HUBS + 1]	uint8_t	The port path, starting from the root hub.
speed	uint8_t	The speed (USBH_LOW_SPEED, USBH_FULL_SPEED, or USBH_HIGH_SPEED).
rwkup	uint8_t	The device on the port supports remote wakeup.
vid	uint16_t	The vendor ID.
pid	uint16_t	The product ID.

t_usbh_cd

For all class drivers except for hub class drivers, the *t_usbh_cd* structure lists the set of callback functions the host can call if certain events occur.

> **Note:** The class driver must provide these functions, although some of them may be null.

The structure takes this form:

Element	Type	Description
vid	uint16_t	The vendor ID.
pid	uint16_t	The product ID.
check	t_usbh_cd_check_fn	The function used to perform a check.
connect	t_usbh_cd_connect_fn	The function used to make a connection.
disconnect	t_usbh_cd_disconnect_fn	The function used to disconnect a device.
descriptor	t_usbh_cd_descriptor_fn	The function used to get a device descriptor.
ep_dsc	t_usbh_cd_ep_dsc *	The endpoint descriptor.

t_usbh_hub

For hub class drivers, the *t_usbh_hub* structure lists the set of callback functions the host can call if certain events occur.

> **Note:** The class driver must provide these functions, although some of them may be null.

The structure takes this form:

Element	Type	Description
scan	t_usbh_hub_scan_fn	The function used to scan for a device status change.
reset	t_usbh_hub_reset_fn	The function used to reset a hub port.
disable	t_usbh_hub_fn	The function used to disable a hub port.
suspend	t_usbh_hub_fn	The function used to suspend a hub port.
resume	t_usbh_hub_fn	The function used to resume a hub port.
test	t_usbh_hub_test_fn	The function used to set test mode on a hub port.

Test Modes

The test modes are as follows:

Mode	Description
USBH_TEST_MODE_OFF	Test mode disabled (used for internal purposes).
USBH_TEST_MODE_J_STATE	J state test mode. The port's transceiver enters the high speed J state and remains in this mode throughout. This lets you test the high output drive level on the D+ line.
USBH_TEST_MODE_K_STATE	K state test mode. The port's transceiver enters the high speed K state and remains in this mode throughout. This lets you test the high output drive level on the D- line.
USBH_TEST_MODE_SE0_NAK	SE0/NAK test mode. The port's transceiver enters high speed receive mode and remains in this mode throughout. This lets you test output impedance, low level output voltage, and loading characteristics. In this mode upstream facing ports must respond to any IN token packet (that has a correct packet CRC) with a NAK handshake within the allowed device response time. This lets you test the device squelch level circuitry and also provides a stimulus/ response test for general functional testing.
USBH_TEST_MODE_PACKET	Packet test mode. A port transmits a test packet repeatedly. This lets you test rise and fall times, eye patterns, jitter, and any other dynamic waveform specifications. The inter-packet timing must not be less than the minimum allowed inter-packet gap and must not exceed 125 µs.
USBH_TEST_MODE_FORCE_ENABLE	Force enable test mode. Here downstream-facing hub ports are enabled in high speed mode, whether or not a device is attached. Packets that arrive at the hub's upstream-facing port are repeated on the port that is in force enable test mode. This lets you test the hub's disconnect detection: by polling the disconnect detect bit while varying the loading on the port, you can measure the disconnect detection threshold voltage.

Connection States

The connection states are as follows:

State	Description
USBH_STATE_FREE	The port is free (this is not seen from the user space).
USBH_STATE_INVALID	Invalid state.
USBH_STATE_DISCONNECTED	Disconnected.
USBH_STATE_CONNECTED	Connected.
USBH_STATE_SUSPENDED	Suspended.
USBH_STATE_ISUSPENDED	Indirectly suspended.
USBH_STATE_RWKUP	Remote wakeup request not processed yet.
USBH_STATE_OVERCURRENT	Overcurrent.
USBH_STATE_ENUM	Enumeration request.
USBH_STATE_CHANGED	Changed.
USBH_STATE_OPERATIONAL	Operational.

Notification Codes

The standard notification codes shown below are defined in the file **api_usb_host.h**.

Notification	Description
USBH_NTF_CONNECT	Connection notification code.
USBH_NTF_DISCONNECT	Disconnection notification code.
USBH_NTF_CD_BASE	This is the first notification a class driver can use.

t_usbh_config_select_cb

The **t_usbh_config_select_cb** definition specifies the format of the select configuration callback function.

Format

```
typedef uint8_t ( * t_usbh_config_select_cb )(
    t_usbh_port_hdl   port_hdl,
    uint16_t          vid,
    uint16_t          pid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle. This is used to specify the device if **usbh_reenumerate()** is to be called.	t_usbh_port_hdl
vid	The vendor ID.	uint16_t
pid	The product ID.	uint16_t

Return Codes

Parameter	Description
Index.	The index of the configuration to use.
USBH_CONFIG_SELECT_ALL	Any configuration can be used. This is returned if the system needs to parse all the configurations and select the first one that has a valid interface (depending on the available class drivers).

t_usbh_enum_failed_cb

The **t_usbh_config_select_cb** definition specifies the format of the enumeration failed callback function.

Format

```
typedef void ( * t_usbh_enum_failed_cb )(
    t_usbh_port_hdl   port_hdl,
    uint16_t          vid,
    uint16_t          pid )
```

Arguments

Parameter	Description	Type
port_hdl	The port handle.	t_usbh_port_hdl
vid	The vendor ID.	uint16_t
pid	The product ID.	uint16_t

String Descriptors

String descriptors provide human-readable information and are optional.

The string descriptors are as follows:

Mode	Description
USBH_STR_MANUFACTURER	The index to use to get the manufacturer ID string.
USBH_STR_PRODUCT	The index to use to get the product ID string.
USBH_STR_SERIAL	The index to use to get the serial number string.
USBH_LANG_ID_DEFAULT	The default language ID when requesting a string.

Note: If any descriptors are not used, their index fields must be set to zero to indicate that no string descriptor is available.

5 Integration

This section describes all aspects of the module that require integration with your target project. This includes porting and configuration of external resources.

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 USB host base system uses the following OAL components:

OAL Resource	Number Required
Tasks	2
Mutexes	4
Events	4

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 USB host base system 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 USB host base system makes use of the following standard PSP macros:

Macro	Package	Component	Description
PSP_RD_LE16	psp_base	psp_endianness	Reads a 16 bit value stored as little-endian from a memory location.
PSP_WR_LE16	psp_base	psp_endianness	Writes a 16 bit value to be stored as little-endian to a memory location.

6 Sample Code

This example shows code that you can use to start the USB host stack:

```
void setup_usb_host_stack()
{
    int       rc;

    rc = usbh_init();                /* Initialize the USB host stack */

    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_init( 0, usbh_stm32uh_hc, 0 );   /* Add a host controller (0) */
    }
    /* Add more host controllers as required.... */

    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdc_acm();         /* Initialize class driver (here it's CDC-ACM) */
    }
    /* Add more class drivers as required.... */

    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_cdc_acm_start();   /* Start class driver */
    }

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

    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_hc_start( 0 );     /* Start the USB host controller */
    }

    /* USB host stack is now ready to use */

    return rc;
}
```