

Embedded USB Host FTDI Class Driver User Guide

Version 1.50

For use with USBH FTDI Class Driver versions 3.09 and
above

Date: 19-Jun-2017 11:42

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
FTDI System	8
Version File	8
Configuration Options	9
Application Programming Interface	10
Module Management Functions	10
usbh_ftdiser_init	11
usbh_ftdiser_start	12
usbh_ftdiser_stop	13
usbh_ftdiser_delete	14
Device Management Functions	15
usbh_ftdiser_send	16
usbh_ftdiser_get_send_state	17
usbh_ftdiser_receive	18
usbh_ftdiser_get_status	19
usbh_ftdiser_reset	20
usbh_ftdiser_set_control_line_state	21
usbh_ftdiser_set_line_coding	22
usbh_ftdiser_get_port_hdl	23
usbh_ftdiser_present	24
usbh_ftdiser_register_ntf	25
Error Codes	26
Types and Definitions	27
t_usbh_ntf_fn	27
Notification Codes	27
Data Bit Codes	28
Line Status Codes	28
Modem Status Codes	29
Parity Codes	29
Stop Bit Codes	29
Integration	30
OS Abstraction Layer	30
PSP Porting	30

Sample Code	31
Initialization	32
Data Transfer	33

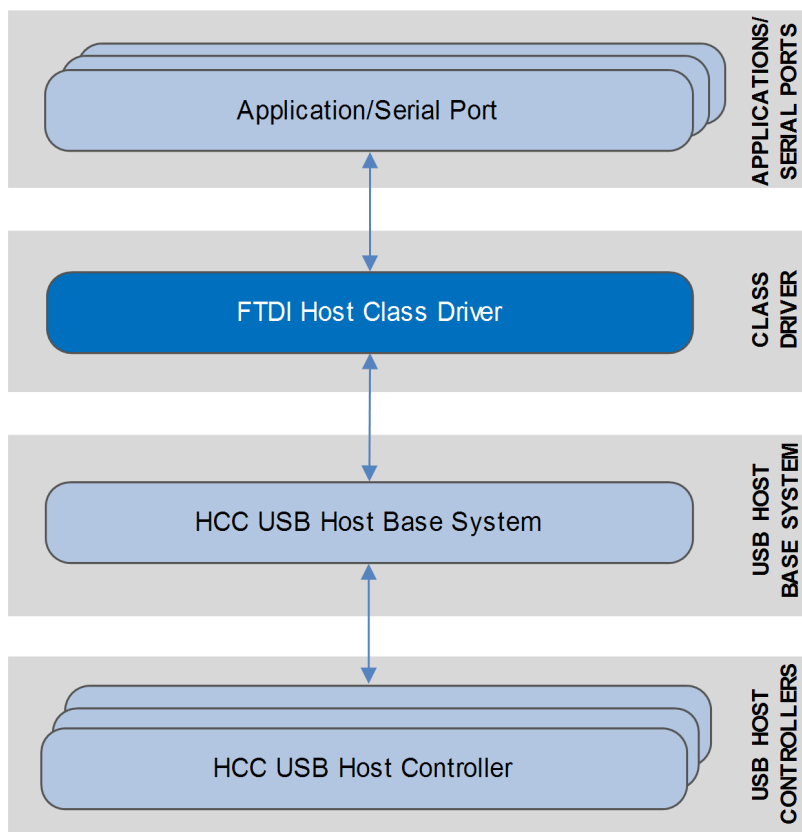
1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB host class driver to control FTDI (Future Technology Devices International) USB devices. These devices convert RS-232 or TTL serial transmissions to USB signals, allowing legacy peripherals to be used with USB.

The `usbh_cd_ftdi` package provides an FTDI host class driver for a USB stack. This is used for connecting FTDI devices over a USB link to a host.

The system structure is shown in the diagram below:



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 unchanged, whichever HCC USB host controller it is interfaced to. For detailed information about this layer, consult 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.

Sample code in this guide shows how to initialize the system and perform data transfer.

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 FTDI 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_ftdi</code>	The USB device FTDI 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 FTDI 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 FTDI Class Driver User Guide](#).
- For the history of changes made to the package code itself, see [History: usbh_cd_cdc_ftdi](#).

The current version of this manual is 1.50. The full list of versions is as follows:

Manual version	Date	Software version	Reason for change
1.50	2017-06-19	3.09	New <i>Change History</i> format.
1.40	2016-04-20	3.09	Added function group descriptions to API.
1.30	2015-12-05	3.03	Various small changes.
1.20	2015-04-02	3.03	Added <i>Change History</i>
1.10	2015-01-13	3.03	Added <i>Sample Code</i> .
1.00	2014-08-27	3.03	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_ftdiser.h` should 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_ftdiser.h` contains all the configurable parameters of the system. Configure these as required. For details of these options, see [Configuration Options](#).

2.3 FTDI System

The code file is `src/usb-host/class-drivers/ftdi-ser/usbh_ftdiser.c`. **This file should only be modified by HCC.**

2.4 Version File

The file `src/version/ver_usbh_ftdiser.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_ftdiser.h`.

USBH_FTDISER_MAX_UNITS

The number of FTDI lines. The default is 2.

USBH_FTDISER_RXBUF_SIZE

The size of the receive buffer. The default is 64.

The minimum size for full speed systems is 64 and, for these, the buffer size should always be a multiple of 64. The minimum size for high speed systems is 512 and, for these, the buffer size should always be a multiple of 512.

USBH_FTDISER_CLK_SPEED

The speed of the FTDI clock. Keep this at 48000000 (the default).

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_ftdiser_init()</code>	Initializes the module and allocates the required resources.
<code>usbh_ftdiser_start()</code>	Starts the module.
<code>usbh_ftdiser_stop()</code>	Stops the module.
<code>usbh_ftdiser_delete()</code>	Deletes the module and releases the resources it used.

usbh_ftdiser_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_ftdiser_init ( void )
```

Arguments

Parameter

None.

Return Values

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

usbh_ftdiser_start

Use this function to start the class driver.

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

Format

```
int usbh_ftdiser_start ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_ftdiser_stop

Use this function to stop the class driver.

Format

```
int usbh_ftdiser_stop ( void )
```

Arguments

Parameter
None.

Return Values

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

usbh_ftdiser_delete

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

Format

```
int usbh_ftdiser_delete ( void )
```

Arguments

Parameter
None.

Return Values

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

4.2 Device Management Functions

The functions are the following:

Function	Description
<code>usbh_ftdiser_send()</code>	Sends data to the device.
<code>usbh_ftdiser_get_send_state()</code>	Gets the state of the last send.
<code>usbh_ftdiser_receive()</code>	Receives data on the FTDI serial channel.
<code>usbh_ftdiser_get_status()</code>	Gets the modem/line status.
<code>usbh_ftdiser_reset()</code>	Resets the FTDI device.
<code>usbh_ftdiser_set_control_line_state()</code>	Configures the control line.
<code>usbh_ftdiser_set_line_coding()</code>	Sets the line coding (baud rate, data and stop bits, and parity).
<code>usbh_ftdiser_get_port_hdl()</code>	Gets the port handle.
<code>usbh_ftdiser_present()</code>	Checks whether an FTDI device is connected.
<code>usbh_ftdiser_register_ntf()</code>	Registers a notification function for a specified event type.

usbh_ftdiser_send

Use this function to send data to the device.

- In RTOS Mode – the call returns immediately if a notification function has previously been registered for the [USBH_NTF_FTDISER_TX](#) event. If this function has not been registered the call returns when the transfer completes.
- In non-RTOS Mode – before using this function you must register a TX notification function by using **usbh_ftdiser_register_ntf()**. This will be called when the transfer completes; see [USBH_NTF_FTDISER_TX](#).

Note: **usbh_ftdiser_send()** cannot be called from the notification function. The notification function can be used to wake up a task that calls **usbh_ftdiser_send()** or to send an event to a task that calls **usbh_ftdiser_send()**.

Format

```
int usbh_ftdiser_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 buffer.	uint8_t *
length	The number of bytes to send.	uint32_t

Return Values

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

usbh_ftdiser_get_send_state

Use this function to get the state of the last send.

A notification function may call this function to find whether there was an error during transmission, or whether the transfer was successful.

Format

```
int usbh_ftdiser_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
The completion code of the last send.	Successful execution.
Else	See Error Codes .

usbh_ftdiser_receive

Use this function to receive data on the FTDI serial channel.

This call returns immediately, even if nothing is received.

An RX notification function may be registered by using **usbh_ftdiser_register_ntf()**. This will be called when at least one character is received; see [USBH_NTF_FTDISER_RX](#).

Note: **usbh_ftdiser_receive()** cannot be called from the notification function. The notification function can be used to wake up a task that calls **usbh_ftdiser_receive()**, or send an event to a task that calls **usbh_ftdiser_receive()**.

Format

```
int usbh_ftdiser_receive (
    t_usbh_unit_id  uid,
    uint8_t *      p_buf,
    uint32_t       max_length,
    uint32_t *     rlength )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_buf	The buffer to hold incoming data.	uint8_t *
max_length	The maximum length of the receive buffer.	uint32_t
rlength	The number of bytes received.	uint32_t *

Return Values

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

usbh_ftdiser_get_status

Use this function to get the modem/line status.

The status values are returned as masks.

Format

```
int usbh_ftdiser_get_status (
    t_usbh_unit_id  uid,
    uint8_t *      p_modem_status,
    uint8_t *      p_line_status )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
p_modem_status	Where to write the modem status .	uint8_t *
p_line_status	Where to write the line status .	uint8_t *

Return Values

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

usbh_ftdiser_reset

Use this function to reset the FTDI device.

Format

```
int usbh_ftdiser_reset ( 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 Error Codes .

usbh_ftdiser_set_control_line_state

Use this function to configure the control line.

Format

```
int usbh_ftdiser_set_control_line_state (
    t_usbh_unit_id  uid,
    uint8_t         rts,
    uint8_t         dtr,
    uint8_t         use_rts,
    uint8_t         use_dtr )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
rts	The RTS state.	uint8_t
dtr	The DTR state.	uint8_t
use_rts	Use RTS state.	uint8_t
use_dtr	Use DTR state.	uint8_t

Return Values

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

usbh_ftdiser_set_line_coding

Use this function to set the line coding.

Format

```
int usbh_ftdiser_set_line_coding (
    t_usbh_unit_id  uid,
    uint32_t        br,
    uint8_t         b,
    uint8_t         p,
    uint8_t         s )
```

Arguments

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
br	The baud rate.	uint32_t
b	The Data bits .	uint8_t
p	The Parity code .	uint8_t
s	The Stop bits .	uint8_t

Return Values

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

usbh_ftdiser_get_port_hdl

Use this function to get the port handle.

Format

```
t_usbh_port_hdl usbh_ftdiser_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 Error Codes .

usbh_ftdiser_present

Use this function to check whether an FTDI device is connected.

Format

```
int usbh_ftdiser_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 FTDI device is present.
1	An FTDI device is present.
Else	See Error Codes .

usbh_ftdiser_register_ntf

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

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_ftdiser_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 use when an event occurs.	t_usbh_ntf_fn

Return Values

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

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

Parameter	Description	Type
uid	The unit ID.	t_usbh_unit_id
ntf	The notification code .	t_usbh_ntf

Notification Codes

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

Notification	Description
USBH_NTF_CONNECT	Connection notification code.
USBH_NTF_DISCONNECT	Disconnection notification code.

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

Notification	Description
USBH_NTF_FTDISER_RX	Data has been received.
USBH_NTF_FTDISER_TX	Data has been successfully transmitted.

Data Bit Codes

The data bit codes are as follows:

Code	Value
USBH_FTDI_BITS_5	5
USBH_FTDI_BITS_6	6
USBH_FTDI_BITS_7	7
USBH_FTDI_BITS_8	8

Line Status Codes

These line status codes may be returned by the `usbh_ftdiser_get_status()` function.

Code	Value	Description
USBH_FTDI_DR	$1U \ll 0$	Data ready.
USBH_FTDI_OE	$1U \ll 1$	Overrun error.
USBH_FTDI_PE	$1U \ll 2$	Parity error.
USBH_FTDI_FE	$1U \ll 3$	Framing error.
USBH_FTDI_BI	$1U \ll 4$	Break interrupt.
USBH_FTDI_THRE	$1U \ll 5$	Transmitter holding register empty.
USBH_FTDI_TEMPT	$1U \ll 6$	Transmitter empty.
USBH_FTDI_RXERROR	$1U \ll 7$	Receiver error.

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

Modem Status Codes

These modem status codes may be returned by the `usbh_ftdiser_get_status()` function.

Code	Value	Description
USBH_FTDI_CTS	1U << 4	Clear to send.
USBH_FTDI_DSR	1U << 5	Data set ready.
USBH_FTDI_RI	1U << 6	Ring indicator.
USBH_FTDI_RLSD	1U << 7	Receive line signal detect.

Parity Codes

The parity codes are as follows:

Code	Value
USBH_FTDI_PARITY_NONE	0
USBH_FTDI_PARITY_ODD	1
USBH_FTDI_PARITY_EVEN	2
USBH_FTDI_PARITY_MARK	3
USBH_FTDI_PARITY_SPACE	4

Stop Bit Codes

The stop bit codes are as follows:

Code	Value
USBH_FTDI_STOP_1	0
USBH_FTDI_STOP_1_5	1
USBH_FTDI_STOP_2	2

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 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 FTDI class driver makes use of the following standard PSP macro:

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

6 Sample Code

This section gives example code for the class driver.

6.1 Initialization

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

```
/*
** Initialize USB host with FTDI class driver.
*/

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

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

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

    /* Initialize the FTDI Class driver module */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_ftdiser_init();
    }

    /* Start the FTDI Class driver */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_ftdiser_start();
    }

    /* Set line coding for the specified FTDI serial line */
    if ( rc == USBH_SUCCESS )
    {
        rc = usbh_ftdiser_set_line_coding( 0, 115200, USBH_FTDI_BITS_8, USBH_FTDI_PARITY_NONE,
        USBH_FTDI_STOP_1 );
    }

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

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


6.2 Data Transfer

This example shows the code of a simple FTDI class driver data handler.

```
/*
** Simple FTDI class driver data handler
** usbh_ftdi_echo()
*   Send back data received by FTDI host.
*/

void usbh_ftdi_echo ( void )

{
    uint32_t len;
    int rc;

    /* FTDI-ser echo demo. */

    if ( usbh_ftdiser_present( 0 ) ) /* Check that device is present */
    {
        rc = usbh_ftdiser_receive( 0, dbuf, BUF_SIZE, &len ); /* Wait for data */
        if ( ( rc == USBH_SUCCESS ) && ( len != 0 ) )
        {
            usbh_ftdiser_send( 0, dbuf, len ); /* Echo data back to device */
        }
    }
}
```