USB Device Audio Class Driver User Guide

Version 1.20

For use with USBD Audio Class Driver versions 3.04 and above

Date: 16-Jun-2017 13:29

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.

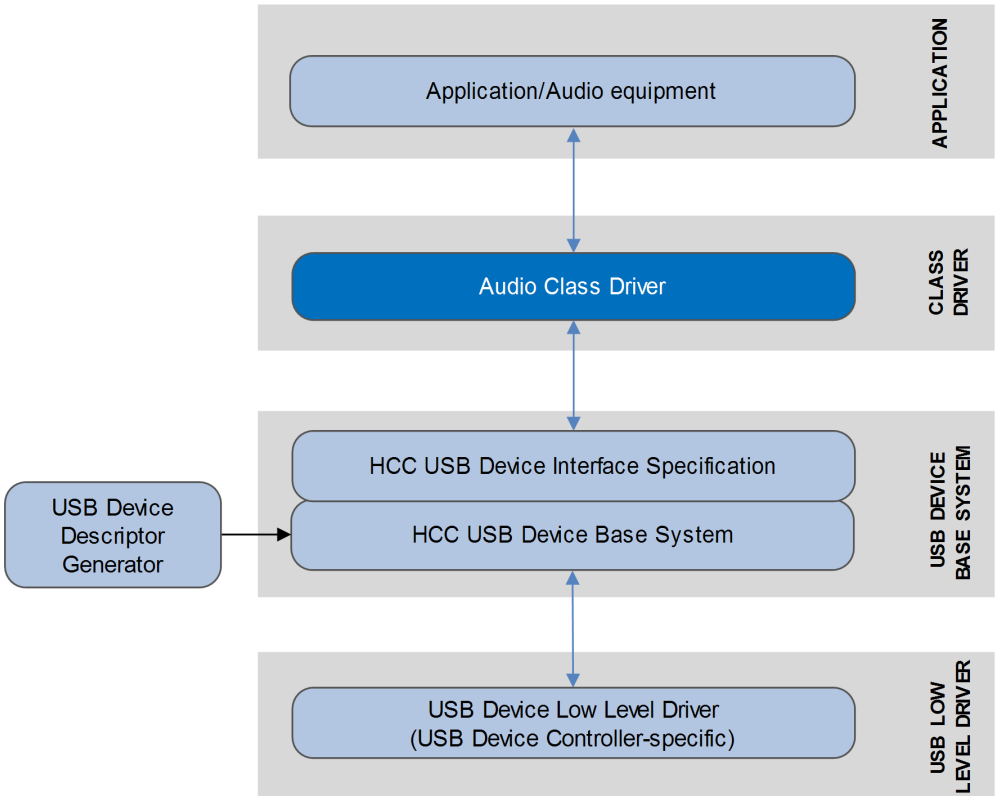
Table of Contents

1 System Overview

1.1 Introduction

This guide is for those who want to implement an embedded USB class driver to control audio devices. The audio class driver enables the device to act as a microphone and/or speaker when connected to a USB host over USB.

The system structure is shown in the diagram below:



Note:

- This module is part of HCC's Embedded USB Device (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.
- Other types of HCC class driver can be added to the system, for example CDC-ACM, mass storage, and printer. For the current list of supported class drivers, contact sales@hcc-embedded.com.

The package provides a set of API functions. The API description in this manual has separate sections describing module management and the audio functions.

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 1.0 audio interface specification.
- Supports the asynchronous, synchronous, and adaptive transfer modes.
- 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

The table below 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_audio	The audio class driver.

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 describes the Embedded USB Device base system.

HCC USB Device Audio Class Driver User Guide

This is this document.

HCC USB Device Descriptor Generator User Guide

This document describes the tool for creating 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 Audio Class Driver User Guide.
- For the history of changes made to the package code itself, see History: usbd_cd_audio.

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	2017-06-16	3.04	New *Change History* format.
1.10	2016-04-21	3.04	Added function group descriptions to API.
1.00	2015-04-17	3.04	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 files.

2.1 API Header Files

These files in the directory **src/api** are the only files that should be included by an application using this module. For details of the API functions, see Application Programming Interface.

File	Description
api_usbd_audio.h	Base audio module API.
api_usbd_audio_map.h	Audio map API.
api_usbd_mic.h	Microphone API.
api_usbd_speaker.h	Speaker API.

2.2 Configuration Files

These files in the directory **src/config** contain all the configurable parameters of the system. Configure these as required. For details of these options, see Configuration Options.

File	Description
config_usbd_audio.h	Base audio configuration file.
config_usbd_mic.h	Microphone configuration file.
config_usbd_speaker.h	Speaker configuration file.

2.3 Source Code Files

These files are in the directory **src/usb-device/class-drivers/audio**. **These files should only be modified by HCC**.

File	Description
usbd_audio.c	Audio device source code.
usbd_audio.h	Audio header file.
usbd_audio_feat_unit.c	Feature unit control code.
usbd_audio_feat_unit.h	Feature unit control header file.
usbd_audio_map.c	Audio map source code.
usbd_audio_map.h	Audio map header file.
usbd_fifo_init.c	FIFO source code.
usbd_fifo_init.h	FIFO header file.
usbd_mic.c	Microphone source code.
usbd_mic.h	Microphone header file.
usbd_speaker.c	Speaker source code.
usbd_speaker.h	Speaker header file.

2.4 Version File

The file **src/version/ver_usbd_audio.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 three configuration files described in this section. The available configuration options and their default values are described.

3.1 Audio Configuration

Set the audio configuration options in the file **src/config/config_usbd_audio.h**.

AUDIO_MIC_SUPPORT

Set this to 0 if microphone support is not needed. The default is 1.

AUDIO_SPK_SUPPORT

Set this to 0 if speaker support is not needed. The default is 1.

USBD_AUDIO_MAX_NO_OF_CHANNELS

The number of audio channels supported by the audio streams (left, right, and so on). The default is 2.

3.2 Microphone Configuration

Set the single microphone configuration option in the file **src/config/config_usbd_mic.h**.

USBD_MIC_BUFFER_PER_EP

The number of packets the audio sample FIFO can hold. The default is 4.

3.3 Speaker Configuration

Set the speaker configuration options in the file **src/config/config_usbd_speaker.h**.

USBD_SPK_BUFFER_PER_EP

The number of packets the audio sample FIFO can hold. The default is 4.

USBD_SPK_SR_REFRESH_PERIOD

The sample rate refresh period. The default is 0. There are two options:

- Synchronization type 1 (for proper operation, use this) – defines a synchronization endpoint for an audio stream to notify the host of the actual sampling rate. Set this parameter to non-zero to use this. The value specified must match the periodicity of this endpoint.

- Disable type 1 synchronization (the default) – the device performs its own sample rate synchronization, dropping or repeating samples when needed. Use this if no synchronization endpoint is defined.

USBD_SPK_VOLUME_UNIT_ID

The unit ID of the feature unit which is responsible for the speaker's volume control. Set this to 0 (the default) to disable volume control.

USBD_SPK_VOLUME_MIN

If volume control is enabled, this is the minimum volume level supported by the device. The value is a signed fixed point number where the fractional part is 8 bits wide. The default is 0x0000 (0dB).

USBD_SPK_VOLUME_MAX

If volume control is enabled, this is the maximum volume level supported by the device. The value is a signed fixed point number where the fractional part is 8 bits wide. The default is 0x1500 (21dB).

USBD_SPK_VOLUME_RES

If volume control is enabled, this is the resolution it uses. The value is a signed fixed point number where the fractional part is 8 bits wide. The value must be greater than or equal to zero. The default is 0x0100 (1dB).

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_audio_init()	Initializes the module and allocates the required resources.
usbd_audio_start()	Starts the module.
usbd_audio_stop()	Stops the module.
usbd_audio_delete()	Deletes the module and releases the resources it used.

usbd_audio_init

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

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

Format

```
int usbd_audio_init ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_audio_start

Use this function to start the audio class driver.

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

Format

```
int usbd_audio_start ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_audio_stop

Use this function to stop the audio class driver.

Format

```
int usbd_audio_stop ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_audio_delete

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

Format

```
int usbd_audio_delete ( void )
```

Arguments

Argument
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

4.2 Packet Size Management

The functions are the following:

Function	Description
get_pkt_size_in()	Returns the packet size of the specified input stream (microphone).
get_pkt_size_out()	Returns the packet size of the specified output stream (speaker).

get_pkt_size_in

Use this function to return the packet size of the specified input stream (microphone).

Format

```
unsigned short get_pkt_size_in ( unsigned char stream )
```

Arguments

Parameter	Description	Type
stream	The stream ID.	unsigned char

Return Values

Return value	Description
The packet size.	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

get_pkt_size_out

Use this function to return the packet size of the specified output stream (speaker).

Format

```
unsigned short get_pkt_size_out ( unsigned char stream )
```

Arguments

Parameter	Description	Type
stream	The stream ID.	unsigned char

Return Values

Return value	Description
The packet size.	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

4.3 Microphone Management

The functions are the following:

Function	Description
usbd_mic_get_in_stream()	Returns a pointer to the FIFO of the specified audio stream.
usbd_mic_get_sr()	Finds the sample rate set by the host.
usbd_mic_is_active()	Finds whether the specified microphone stream is active.
usbd_mic_set_onoff_notify()	Registers a callback function that is called when the microphone is activated or deactivated.
usbd_mic_set_sr_notify()	Registers a callback function that is called when the host changes the sample rate of the microphone.

usbd_mic_get_in_stream

Use this function to return a pointer to the FIFO of the specified audio stream.

Format

```
rngbuf_t * usbd_mic_get_in_stream ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_mic_get_sr

Use this function to find the sample rate set by the host.

This function can be called from the sample rate change notification callback to get the new value.

Format

```
uint32_t usbd_mic_get_sr ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_mic_is_active

Use this function to find whether the specified microphone stream is active.

Format

```
int usbd_mic_is_active ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
0	Stream is inactive.
Non-zero	Stream is active.

usbd_mic_set_onoff_notify

Use this function to register a callback function which is called when the microphone is activated or de-activated.

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

Format

```
void  usbd_mic_set_onoff_notify (
    int        ( * fn )( uint32_t param ),
    uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_mic_set_sr_notify

Use this function to specify a callback function which is called when the host changes the sample rate of the microphone.

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

Format

```
void  usbd_mic_set_sr_notify (
    int       ( * fn )( uint32_t param ),
    uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

4.4 Speaker Management

The functions are the following:

Function	Description
usbd_spk_get_out_stream()	Returns a pointer to the FIFO of the audio stream assigned to the specified stream.
usbd_spk_is_active()	Determines whether the specified speaker stream is active.
usbd_spk_get_sr()	Gets the sample rate set by the host.
usbd_spk_set_synch_rate()	Tells the USB host the actual sample rate.
usbd_spk_get_volume_info()	Returns a pointer to an array that holds the current volume level of all audio channels.
usbd_spk_get_mute_info()	Returns a pointer to an array that holds the mute states of all audio channels.
usbd_spk_set_onoff_notify()	Registers a callback function which is called when the speaker is activated or deactivated.
usbd_spk_set_mute_notify()	Registers a callback function which is called when the host mutes or unmutes any audio channel.
usbd_spk_set_sr_notify()	Registers a callback function which is called when the host changes the sample rate of the speaker.
usbd_spk_set_volume_notify()	Registers a callback function which is called when the host changes the volume level of any audio channel.

usbd_spk_get_out_stream

Use this function to return a pointer to the FIFO of the audio stream assigned to the specified stream.

The FIFO can be used to receive audio samples from the host.

Format

```
rngbuf_t * usbd_spk_get_out_stream ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_is_active

Use this function to determine whether the specified speaker stream is active.

Format

```
int usbd_spk_is_active ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
0	The speaker stream is inactive
Non-zero.	The speaker stream is active.

usbd_spk_get_sr

Use this function to get the sample rate set by the host.

This function can be called from the sample rate change notification callback to get the new value.

Format

```
uint32_t usbd_spk_get_sr ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_set_synch_rate

Use this function to tell the USB host the actual sample rate.

The reported value is used by the host during sample rate synchronization.

> **Note:** This function is only available if sample rate synchronization type 1 is enabled.

Format

```
void usbd_spk_set_synch_rate ( uint32_t rate )
```

Arguments

Parameter	Description	Type
rate	Returns the sample rate.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_get_volume_info

Use this function to get a pointer to an array that holds the current volume level for all audio channels.

The array has USBD_AUDIO_MAX_NO_OF_CHANNELS elements.

Format

```
uint16_t * usbd_spk_get_volume_info ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_get_mute_info

Use this function to get a pointer to an array that holds the mute states for all audio channels.

The number of elements in the array is USBD_AUDIO_MAX_NO_OF_CHANNELS.

Format

```
uint8_t * usbd_spk_get_mute_info ( void )
```

Arguments

Parameter
None.

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_set_onoff_notify

Use this function to register a callback function which is called when the speaker is activated or de-activated.

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

Format

```
void  usbd_spk_set_onoff_notify (
   int        ( * fn )( uint32_t param ),
   uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_set_mute_notify

Use this function to specify a callback function which is called when the host mutes or unmutes any audio channel.

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

Format

```
void  usbd_spk_set_mute_notify (
    int        ( * fn )( uint32_t param ),
    uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_set_sr_notify

Use this function to specify a callback function which is called when the host changes the sample rate of the speaker.

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

Format

```
void  usbd_spk_set_sr_notify (
   int        ( * fn )( uint32_t param ),
   uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

usbd_spk_set_volume_notify

Use this function to specify a callback function which is called when the host changes the volume level of any audio channel.

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

Format

```
void  usbd_spk_set_volume_notify (
    int        ( * fn )( uint32_t param ),
    uint32_t   param )
```

Arguments

Parameter	Description	Type
fn	A pointer to the callback function.	int *
param	The value of the parameter passed to the callback.	uint32_t

Return Values

Return value	Description
USBD_AUDIO_SUCCESS	Successful execution.
USBD_AUDIO_ERROR	Operation failed.

4.5 Error Codes

If a function executes successfully it returns with USBD_AUDIO_SUCCESS, a value of zero.

Return Value	Value	Description
USBD_AUDIO_SUCCESS	0	Successful execution.
USBD_AUDIO_ERROR	1	Operation failed.

5 Integration

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

5.1 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, see the *HCC Base Platform Support Package User Guide*.

The audio module 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.