UM2195
User manual

USB device audio streaming Expansion Package for STM32Cube

Introduction

The universal serial bus (USB) is a widely and frequently used communication protocol. In addition to enabling robust and fast data transfer, it has evolved to encompass data interfacing for various application fields, such as audio streaming for data playback and recording.

This user manual is associated with the X-CUBE-USB-AUDIO Expansion Package that supplements the STM32Cube™ firmware library. It describes the X-CUBE-USB-AUDIO firmware and how to use the library for audio playback or recording.

This user manual covers the following topics to aid in the use of the Expansion Package:

- Introduction to STM32Cube™
- USB Audio Class overview
- Software architecture
- Structure list used
- Sequence diagrams showing USB audio playback interactions

.

Contents

List of tables

List of figures

1 General information

The X-CUBE-USB-AUDIO Expansion Package runs on STM32 microcontrollers, based on the Arm®(a) Cortex® core.

1.1 References

The following sources are references for the development of the STM32Cube™ Expansion Packages:

1. *Universal Serial Bus Device Class Definition for Audio Devices* Release 1.0 March 18, 1998
2. *Universal Serial Bus Device Class Definition for Audio Devices* Release 2.0 May 31, 2006
3. *STM32Cube USB device library* (UM1734)
4. *STM32CubeF4 firmware package*
5. *STM32CubeF7 firmware package*

1.2 Acronyms and abbreviations

Table 1 presents the definition of acronyms and abbreviations that are relevant for a better understanding of this document.

Table 1. List of acronyms

Term	Definition
API	Application programming interface
AS	Audio streaming
BSP	Board support package
CLK	Clock
CMSIS	Cortex® microcontroller system interface standard
EP	USB end point
FU	Feature unit
FW	Firmware
HAL	Hardware abstraction layer
IP	Semiconductor intellectual property core
I^2C	Inter-integrated circuit
I^2S	Inter-IC sound

a. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and or elsewhere.

Table 1. List of acronyms (continued)

Term	Definition
LL	Low-layer
MEMS	Microelectromechanical systems
MIDI	Musical instrument digital interface
MS	MIDI streaming
PCM	Pulse-code modulation
PDM	Pulse-density modulation
SAI	Serial audio interface
SOF	Start-of-frame packet
UAC	USB Audio Class
USB	Universal serial bus
USBD	Universal serial bus driver

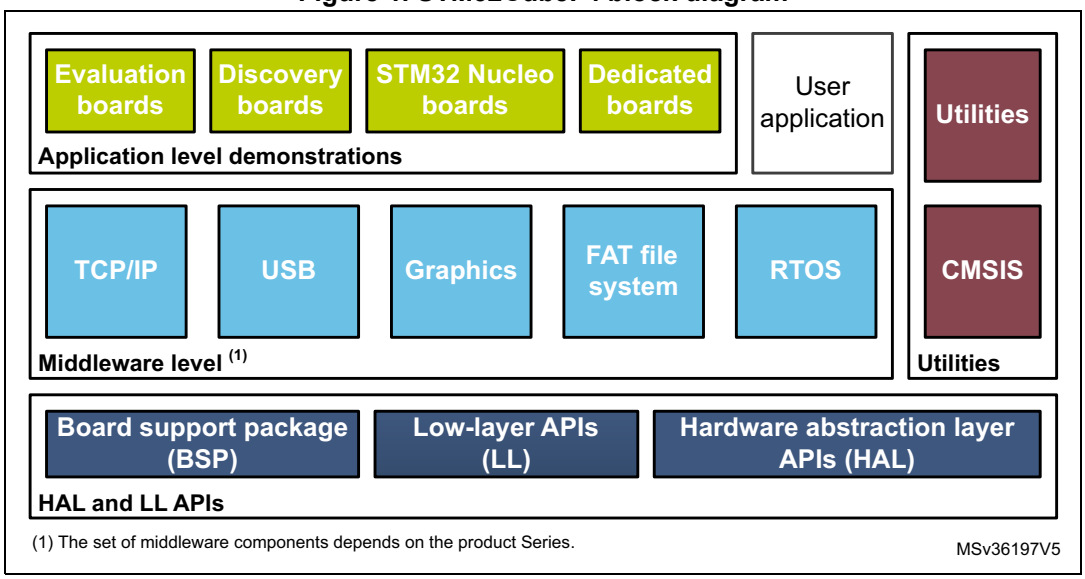
1.3 Introduction to STM32Cube

STM32Cube™ is an STMicroelectronics original initiative to significantly improve designer's productivity by reducing development effort, time and cost. STM32Cube™ covers the whole STM32 portfolio.

STM32Cube™ includes:

- A set of user-friendly software development tools to cover project development from the conception to the realization, among which:
  - STM32CubeMX, a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards.
  - STM32CubeProgrammer (STM32CubeProg), a programming tool available in graphical and command-line versions.
  - STM32CubeMonitor-Power (STM32CubeMonPwr), a monitoring tool to measure and help in the optimization of the power consumption of the MCU.
- STM32Cube™ MCU Packages, comprehensive embedded-software platforms specific to each microcontroller series (such as STM32CubeF4 for the STM32F4 Series), which include:
  - STM32Cube™ hardware abstraction layer (HAL), ensuring maximized portability across the STM32 portfolio.
  - STM32Cube™ low-layer APIs, ensuring the best performance and footprints with a high degree of user control over the HW.
  - A consistent set of middleware components such as RTOS, USB, TCP/IP, and graphics.
  - All embedded software utilities with full sets of peripheral and applicative examples.

Figure 1. STM32CubeF4 block diagram



2 USB audio streaming overview

This chapter gives an overview of the USB Audio Class 1.0 standard used as a reference for the development of the X-CUBE-USB-AUDIO Expansion Package.

2.1 Protocol overview

The USB communications protocol provides specifications for a reliable and fast data transmission between a device and a host such as a PC. The protocol defines four types of data transfer (bulk, interrupt, control, and isochronous). USB Device classes employ transfer types that suit the needs of particular applications and systems. The USB specification documents group devices with similar capabilities into device classes.

The USB Audio Class uses the isochronous transfer type to stream audio across a USB link. Three interfaces are defined within the Audio Class: audio control (AC) interface, zero or more audio streaming (AS) interfaces and zero or more MIDI streaming (MS) interfaces (for MIDI devices).

There are two incompatible versions of the USB Audio Class:

- USB Audio Class 1.0 defined in [*1*]
- USB Audio Class 2.0 defined in [*2*]

The content of this user manual applies to Audio Class specification 1.0, natively supported by the current versions of Windows® (7, 8, and 10) and macOS®(a).

2.2 USB audio function description

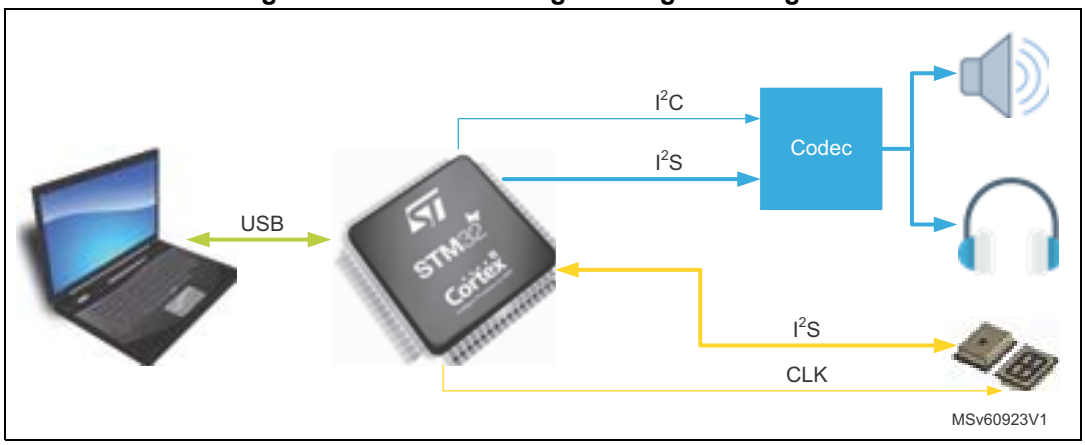
UAC 1.0 and later versions define the audio function. To easily represent the topology and manipulate the physical properties of an audio function, two types of generic entities are defined: the units and the terminals.

The audio function is located at the interface level in the device class hierarchy. It consists of a number of interfaces grouping related pipes that together implement the interface to the audio function. The Audio Interface class groups all functions that can interact with USB-compliant audio data streams.

The USB audio streaming function of the X-CUBE-USB-AUDIO package is depicted in *Figure 2*.

a. macOS® is a trademark of Apple Inc. registered in the U.S. and other countries.

Figure 2. Audio streaming model global diagram



X-CUBE-USB-AUDIO is designed to support:

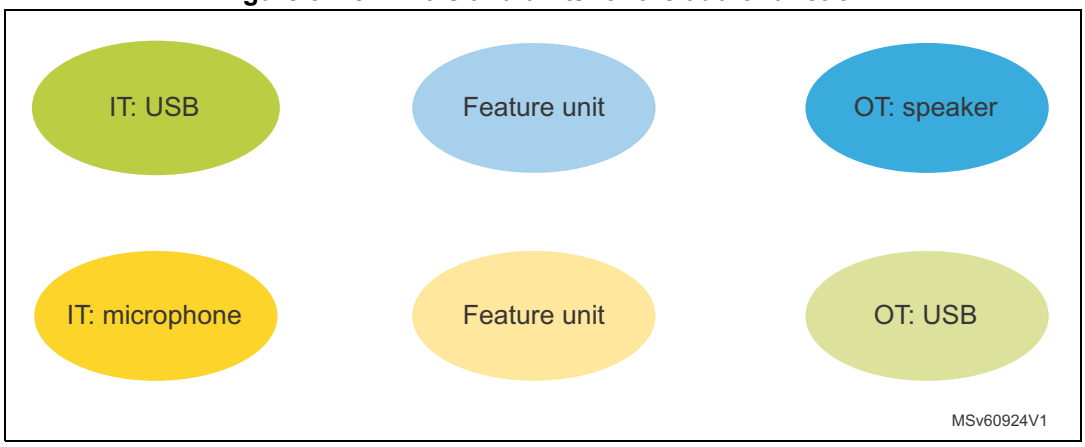
1. USB playback device.
2. USB recording device.
3. USB playback and recording device.

For audio playback, the STM32 board is recognized by the PC (USB Host) as a USB speaker. The PC sends audio samples to the board via the USB. The STM32 MCU configures the audio codec via the I^2C and transmits audio samples via the I^2S. The codec is connected to the analog speaker.

For audio recording, the STM32 board is recognized by the PC as a USB microphone. MEMS microphones capture the sound as PDM samples. The STM32 converts the PDM samples to the PCM format and transmits them to the host through the USB.

The terminals and units that implement the USB audio function are depicted in *Figure 3*.

Figure 3. Terminals and units for the audio function



The playback function is composed of:

- Input terminal: the USB input, which represents the USB OUT isochronous endpoint, where the host sends audio samples. When feedback synchronization is activated, an additional IN isochronous endpoint is required.
- Output terminal: the speaker, where data are sent to the codec
- Feature unit: it provides mute/unmute and volume-change control.

The recording function is composed of:

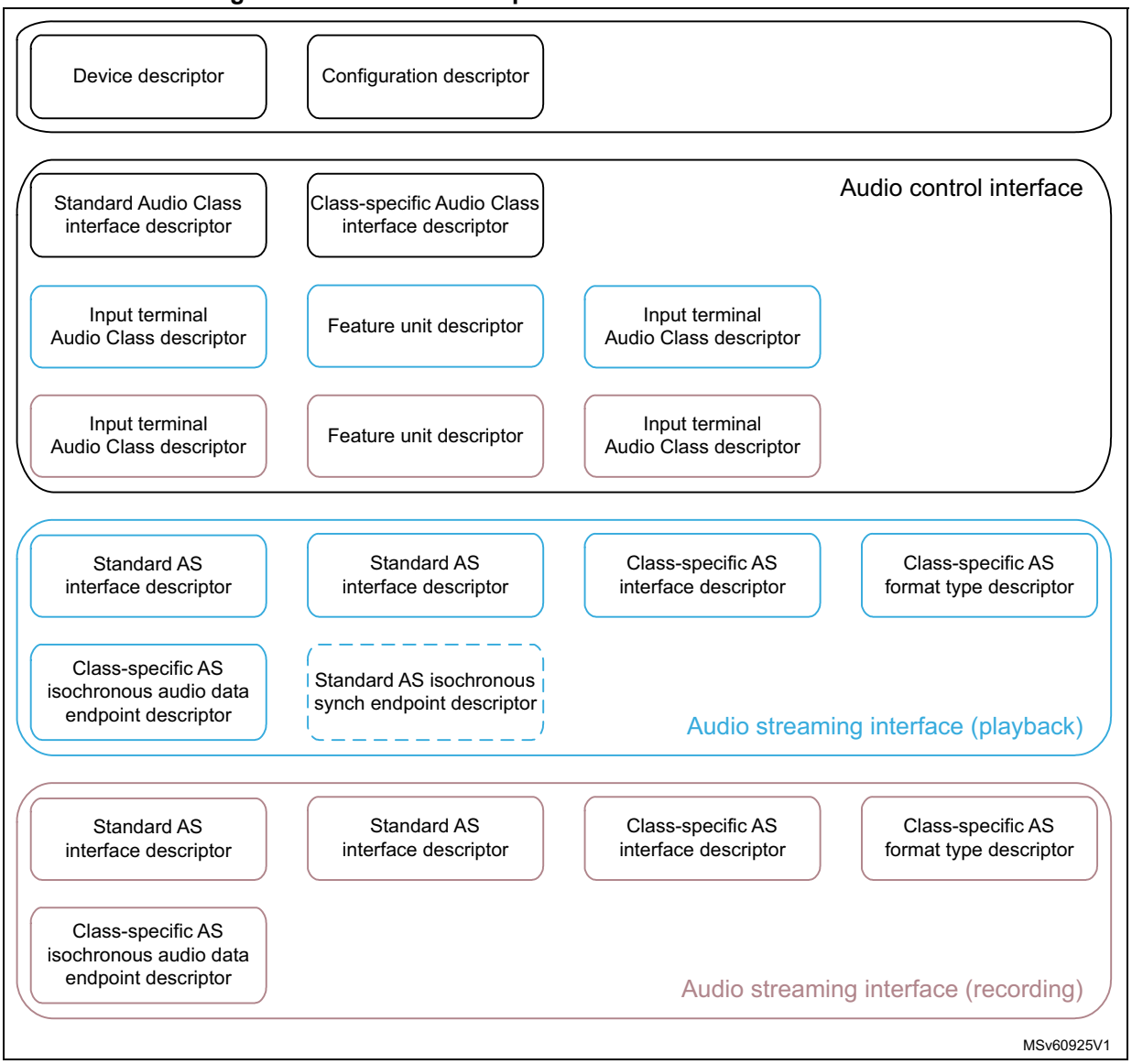
- Input terminal: the microphone, which captures and provides audio samples.
- Output terminal: the USB output, through which the STM32 sends data via an IN isochronous endpoint.
- Feature unit: it provides mute/unmute and volume-change control.

The descriptors used to support such topology are as illustrated in *Figure 4*. In addition to device descriptor and configuration description, three interfaces are defined:

1. Audio control interface: contains the description of used terminals and units.
2. Audio streaming interface for playback, with two alternate settings:
   a) The Zero bandwidth alternate setting: set when playback is not running.
   b) The Alternate setting 1: set to start playback. It defines the used OUT endpoint and Synch endpoint. It provides supported format frequencies and sample resolution.
3. Audio streaming interface for recording, with two alternate settings:
   a) The Zero bandwidth alternate setting: set when recording is off.
   b) The Alternate setting 1: set to start recording. It defines the used IN endpoint. It provides supported format frequencies and sample resolution.

Blue outlined descriptors are used for playback and brown outlined descriptors are used for recording.

Figure 4. List of the descriptors used for X-CUBE-USB-AUDIO



2.3 Synchronization

The main issue with USB audio applications is that different clock domains co-exist quite independently. The USB low-layer protocol does not provide any intrinsic clock synchronization mechanism, unlike other protocols like I^2S or SPDIF. Since clock domains are not synchronized, and due to several clock attributes (clock drift, clock jitter, clock-to-clock phase differences), it can happen that one clock becomes slightly faster or slower than another one despite all measures taken to have very accurate clock sources. This means that a sink or a source consumes or produces data faster or slower than its source or sink. In this situation, data loss results from overrun or under-run occurs. Such event is propagated in time, periodically or permanently.

For audio streaming, such behavior results in audible artifacts (such as clicks, pops, or distortion). Note that having very accurate clock sources, even with very low jitter, does not

help preventing synchronization issues, which are mainly due to drift and phase attributes. To overcome such issues, it is mandatory to synchronize the Device and Host clock domains.

The USB specification determines three clocking models: asynchronous, synchronous, and adaptive. For more details about these models, refer to *section 5.12: Special Considerations for Isochronous Transfers* in *USB Specification 2.0*.

Synchronous

The clock system of synchronous isochronous audio endpoints is controlled externally through SOF synchronization. Such endpoint must do one of the following:

1. Slave its sample clock to the 1 ms SOF tick.
2. Control the rate of USB SOF generation so that its data rate becomes automatically locked to SOF.

Adaptive

Adaptive isochronous audio endpoints are able to source or sink data at any rate within their operating range. This implies that these endpoints must run an internal process that allows them to match their rate.

Asynchronous

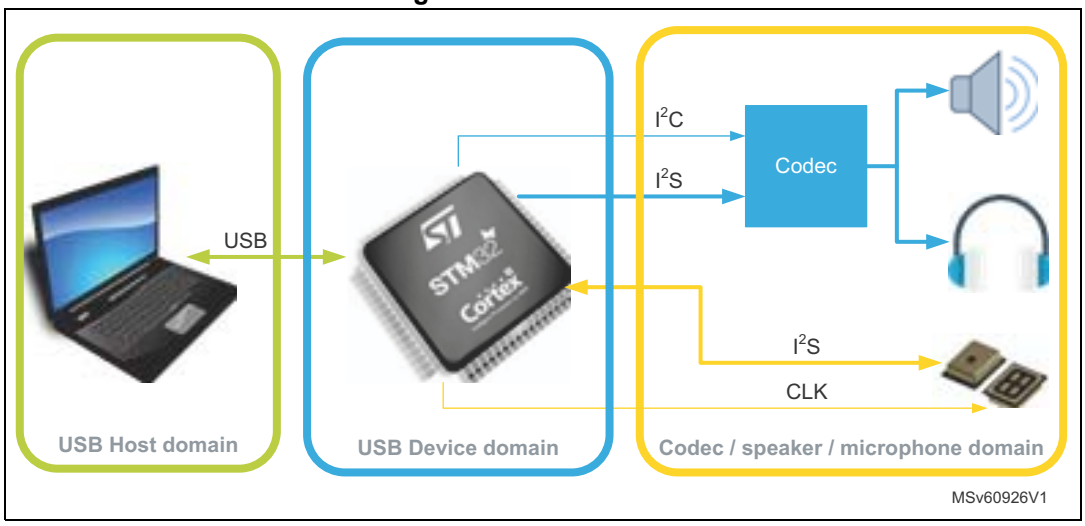
Asynchronous isochronous audio endpoints produce or consume data at a rate that is locked to a local clock. In this example application, only the Asynchronous synchronization model is implemented.

For playback, the device provides periodically the codec rate to the host through the feedback endpoint (when feedback synchronization is activated). Hence, the host must adapt its sampling rate to the required rate. To get a precise measurement of the codec sampling rate, the application reads the amount of data remaining in the DMA buffer when a SOF is received, then counts the number of samples consumed since the previous SOF. The estimated sampling rate is the sum of samples in one second.

For recording, implicit feedback is used; the device computes the microphone sampling rate, then adapts the USB sampling rate. The application estimates the sampling rate in the same way as for playback, then removes samples if the estimated frequency exceeds the required frequency. However, if a `USE_AUDIO_RECORDING_USB_NO_REMOVE` compilation flag is activated, the application does not remove samples but just adds at most one sample to the USB packet.

Figure 5 summarizes the clock domains to be considered in an USB audio streaming application.

Figure 5. Clock domains



2.4 Data flow

Figure 6 illustrates the data flow for the audio expansion.

All USB controls are received by EP 0.

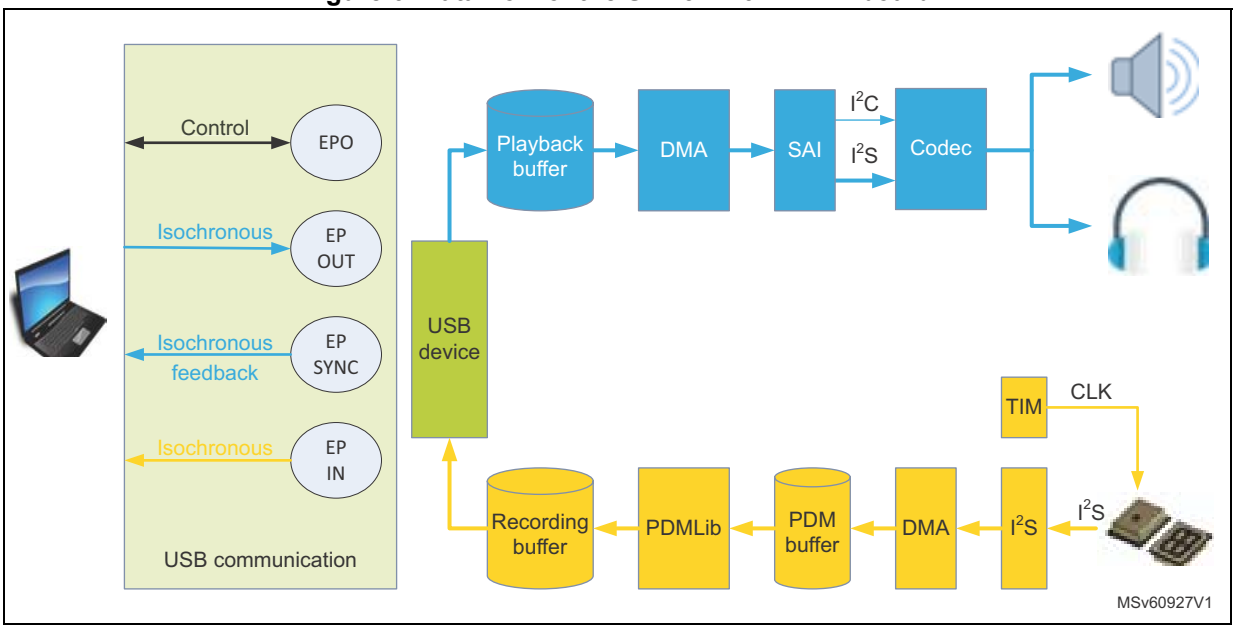
For playback (STM32446E-EVAL and 32F769IDISCOVERY):

1. After the host sets the alternate setting to “1” for the audio streaming interface, it starts transmitting PCM samples.
2. For each packet received, the USB IP writes it in the FIFO memory and then raises an interrupt to the application.
3. The application copies the data to a circular buffer.
4. In the other side, the SAI IP transmits data to the audio codec via I^2S.
5. SAI is fed using a DMA. Each time the DMA completes packet transfer, it raises a transfer-complete interrupt.
6. After receiving a transfer-complete interrupt, the DMA is programmed to read the next packet from the circular buffer.
7. When synchronization is activated, the application estimates the codec sampling rate. Each time the host requires data (isochronous IN to EP Synch), the device sends the computed sampling rate.

For recording with the STM32446E-EVAL Eval board:

1. After the host sets the alternate setting to “1” for the audio streaming interface, the STM32 device starts sending PCM samples.
2. MEMS microphones produce PDM samples via the I^2S channel. The DMA transmits them to an intermediate buffer.
3. The PDM library converts PDM samples to PCM samples. The application writes them to a recording circular buffer.
4. Each millisecond, the host requests a USB packet from the recording IN endpoint.
5. The application copies one packet to the USB FIFO.
6. The STM32 USB IP transmits data to the host.
7. When synchronization is activated, the application may send one sample more or less. If no data is ready to transmit, the application sends a zero-padded packet.

Figure 6. Data flow of the STM32446E-EVAL board

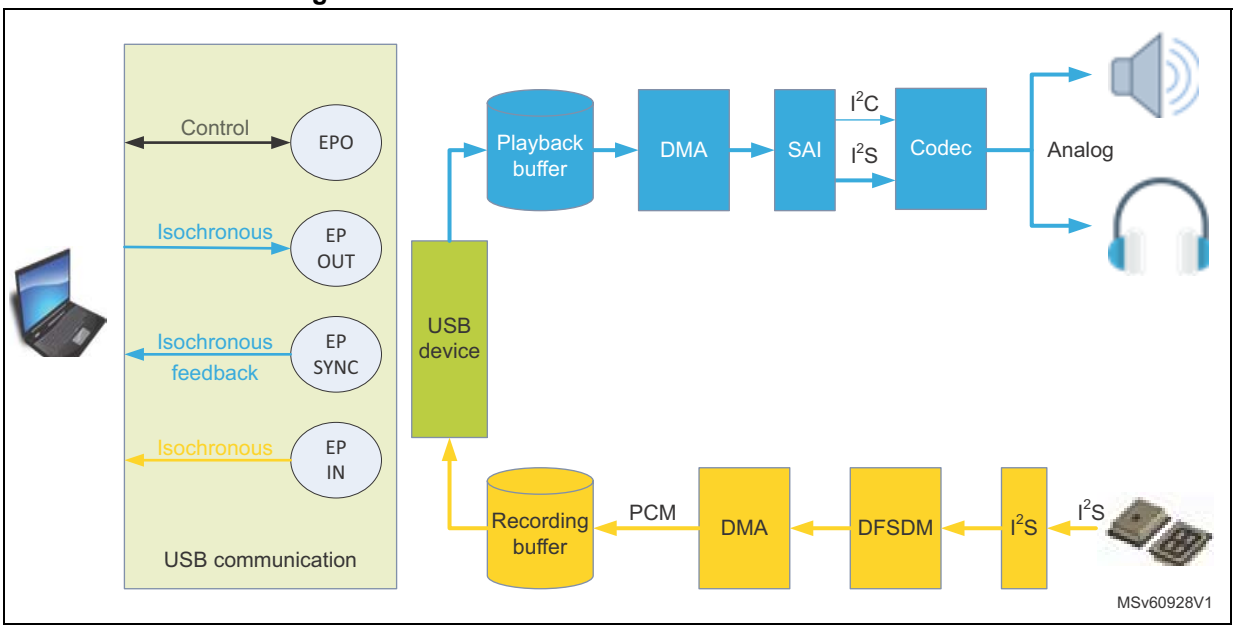


For recording with the 32F769IDISCOVERY Discovery board:

- The STM32F769 device on the board integrates the DFSDM IP. There is no need for PDM to PCM software conversion. The application directly reads PCM samples using the DMA.
- The rest of the data path is the same as for the STM32446E-EVAL board.

Figure 7 illustrates the data flow for the 32F769IDISCOVERY Discovery board.

Figure 7. Data flow of the 32F769IDISCOVERY board



3 USB audio streaming expansion architecture

3.1 Overview

The X-CUBE-USB-AUDIO USB audio streaming Expansion Package is based on the STM32Cube™ firmware. It offers all the APIs required to develop an USB audio streaming package. It supports the STM32446E-EVAL and 32F769IDISCOVERY boards.

3.2 Features

X-CUBE-USB-AUDIO provides the support of audio playback and recording with the advanced features requested by the users:

- Audio playback
- Audio recording
- Playback sampling rate: 96 kHz (for hi-fi audio), 48 kHz and 44.1 kHz
- Playback audio resolution: 24 bits (for hi-fi audio) and 16 bits
- Playback synchronization using feedback
- Recording synchronization using implicit feedback
  (remove samples / decrease packet length)
- Recording sampling rate: 96 kHz (for hi-fi audio), 48 kHz, 44.1 kHz and 16 kHz
- Recording audio resolution: 24 bits for (hi-fi audio) and 16 bits
- Both recording and playback support many sampling rates, which may be fixed in compilation stage
- Both recording and playback support multi-frequency: switch between sampling rate on runtime by host request
- Both recording and playback support mute and volume control

3.3 Expansion global architecture

Figure 8. Global architecture diagram

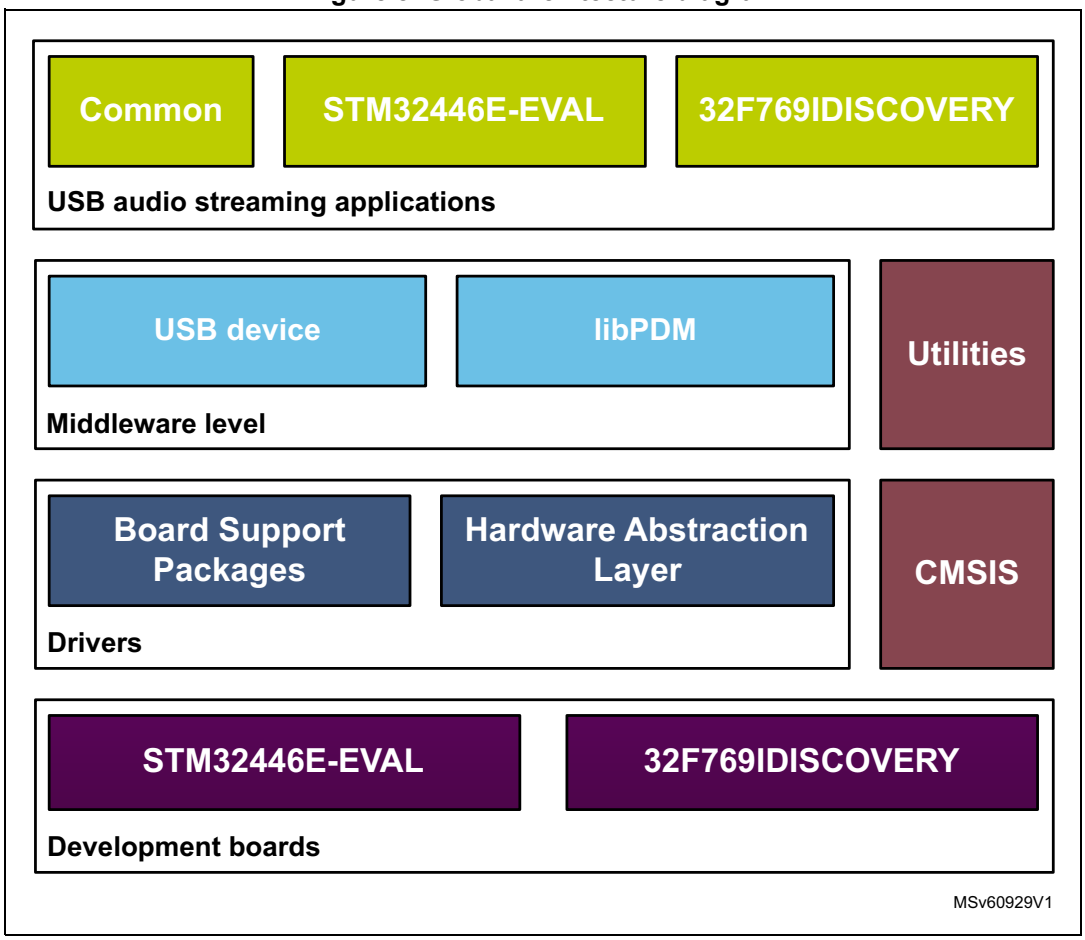
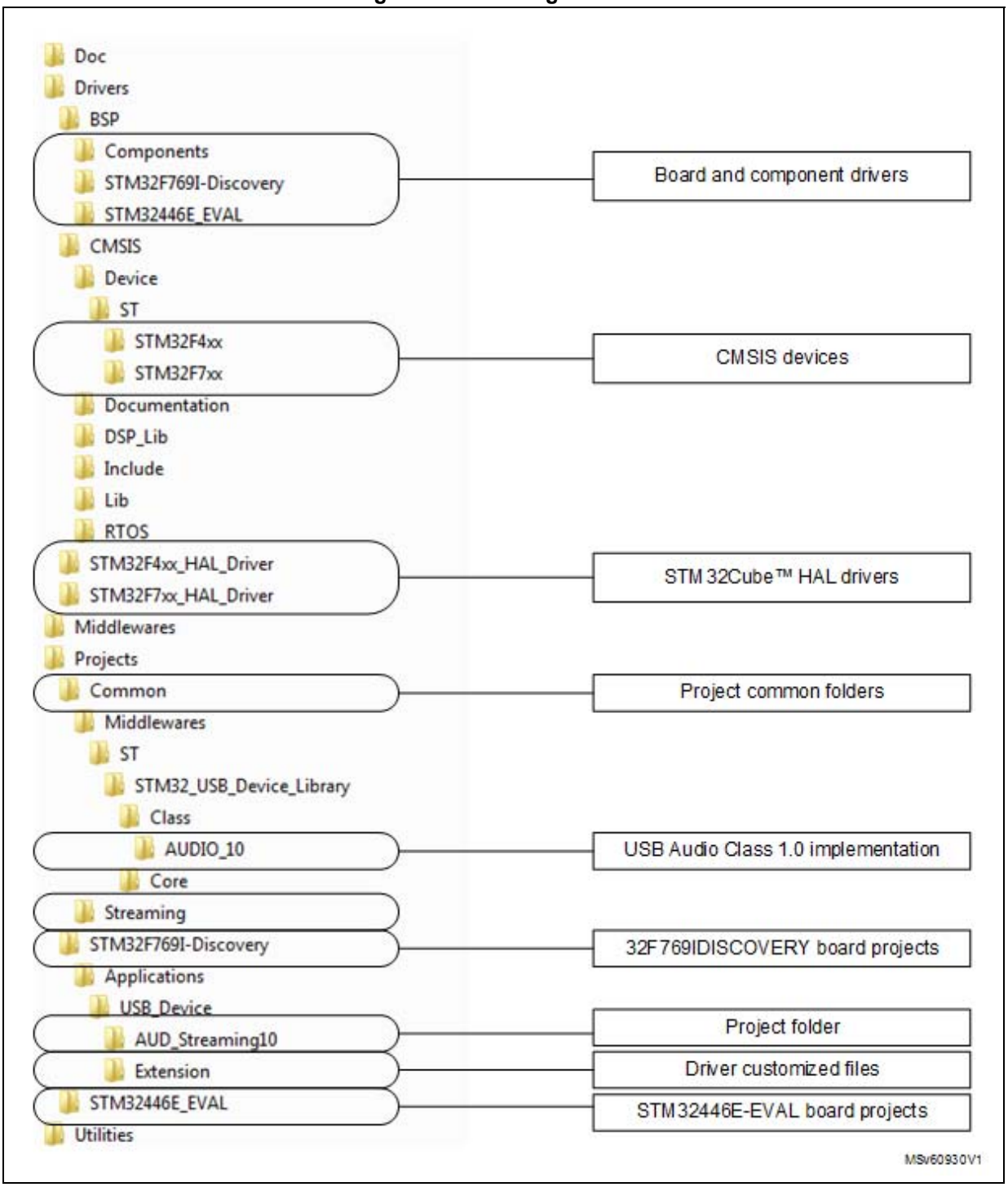


Figure 8 shows how the Expansion Package is built in reference to standard STM32Cube™ MCU Packages (refer to [4] and [5] for more details). *Figure 9 on page 19* depicts the folder structure of the Expansion Package. All modified files are located in project folders; the other folders are inherited from standard STM32Cube™ MCU Packages.

The project folders contain:

- One common folder, which contains:
  - Specialization of USB Device Audio Class; it is based on the USB Device library from the STM32Cube™ middleware.
  - STM32Cube™ MCU Package
  - Common files for the USB audio streaming package.
- Two board specific folders, which contain user-specific configurations:
  - 32F769IDISCOVERY
  - STM32446E-EVAL

Figure 9. Folder organization



3.4 Detailed architecture

In order to support multi uses cases (recording standalone, playback standalone and simultaneous recording and playback), two basic object have been used in the expansion:

1. Nodes: Each terminal input or output or intermediary unit is represented by a node.
2. Sessions: represent streaming interfaces.

Sessions and nodes have a standard interface (`init` function and a list of callbacks such as `start`, `stop`, and others. Sessions and nodes are passive entities, which means that they must be called by a callback or a task in order to perform some job.

Figure 10 depicts the most frequently used nodes and sessions in the current example.

Figure 10. Sessions and nodes interactions

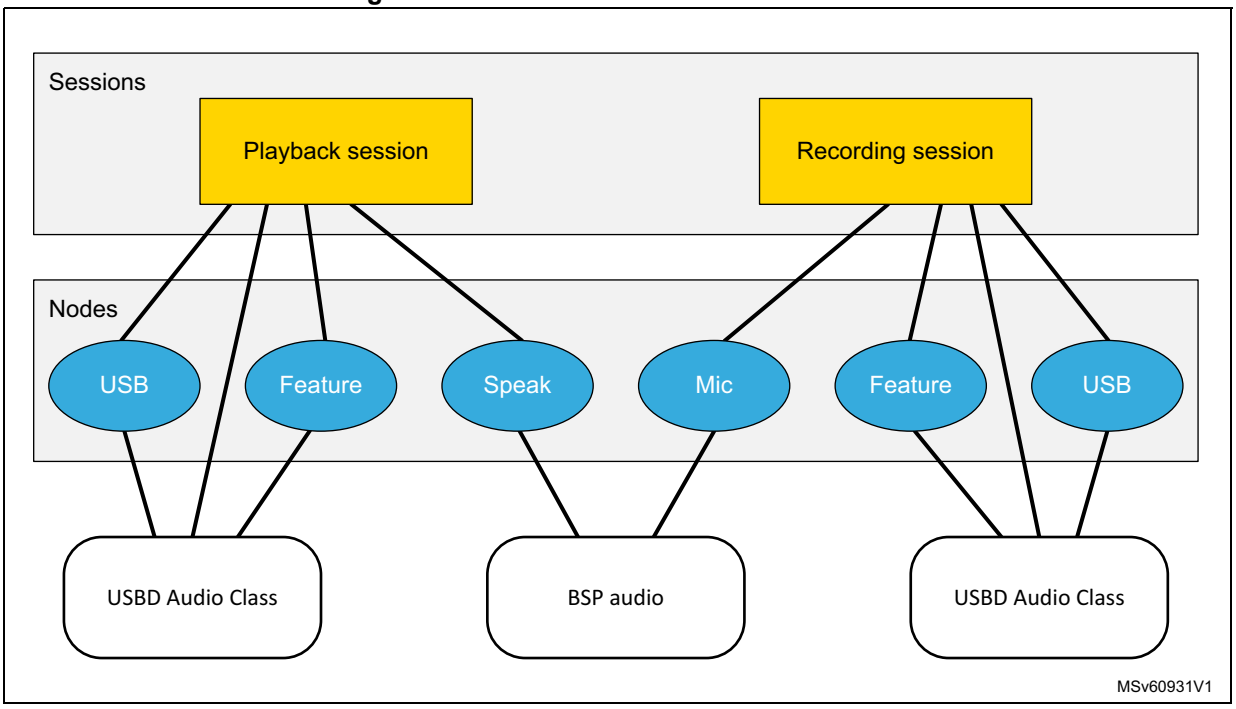


Table 2 describes the sessions an nodes presented in *Figure 10*.

Table 2. Sessions and nodes description

Entity	Description
Playback session	A session that manages playback. It starts, stops, restarts, and executes some commands on a set of nodes. It receives notifications from managed nodes. In the current implementation, the set of playback nodes are: – USB input – Feature unit – Speaker The session has a callback, which is called each time the host sets the playback alternate setting interface.
Recording session	It is similar to the playback session; the set of recording nodes are: – Mic – Feature unit – USB output
USB input (node)	Node responsible of receiving data from the Audio Class. It manages the write pointer of the data buffer.
Feature unit (node)	Node handling Set/Get mute and volume features.
Speaker (node)	Node initializing, starting, and stopping the BSP audio output devices.
USB output (node)	Node responsible of sending data from Audio Class for recording. It manages the read pointer of the data buffer.
Mic (node)	Node initializing, starting and stopping the BSP audio input devices.
USBD Audio Class	Middleware Audio Class implementation.
BSP audio	BSP code to communicate with audio components (codec, microphone).

Table 3 describes the sessions and nodes presented in *Figure 10*.

Table 3. Session and node interaction descriptions

Interaction	Description
Playback session USB input (node)	Playback session may: – Initialize USB input – Start USB input – Stop USB input – Restart USB input USB input may send notification to playback session as: – Begin of streaming, after receiving first packet – Buffer threshold is reached – Overflow of circular buffer – Packet is received – Sampling rate change request
Playback (respectively recording) session Feature unit (node)	Playback (respectively recording) session may: – Initialize FU node (it provides a list of function to control the volume and mute) – Start FU node – Stop FU

Table 3. Session and node interaction descriptions (continued)

Interaction	Description
Playback session Speaker (node)	Playback session may: – Initialize speaker (then speaker plays zeros samples) – Start speaker (then speaker plays real samples) – Stop speaker – Change sampling rate to speaker. – Get played sample count (in order to compute real sampling rate) Speaker notifies playback session for: – underflow of circular buffer – Packet being played
Playback (respectively recording) session USBD Audio Class	Within initialization, session sets streaming interface information, and sets interface alternate setting callback and synchronization callbacks (for playback only). USBD Audio Class calls set interface alternate callback when alternate setting is changed and call SOF received callback for each SOF.
Recording session Microphone (node)	Recording session may: – Initialize Mic (then speaker starts recording samples) – Start Mic (then speaker plays real samples) – Stop Mic – Change sampling rate to Mic. – Get recorded sample count (in order to compute real sampling rate) Mic notifies recording session for: – Overflow of buffer
Recording session USB output (node)	Recording session may: – Initialize USB output – Start USB output – Stop USB output – Restart USB output USB Output may send notification to playback session as: – Underflow of buffer – Sampling rate change request
USB input (node) USBD Audio Class	USB input node sets next callbacks: – Get max packet size – Get buffer – Packet is received – Get current frequency – Set current frequency USBD Audio Class calls: – “Get Max packet size” before opening data EP – “Get Buffer” before calling USB receive next packet – “Packet is received” after transfer is completed of last received packet – “Get/Set Current Frequency” after reception of related control

Table 3. Session and node interaction descriptions (continued)

Interaction	Description
USB output (node) USBD Audio Class	USB output node sets next callbacks: – Get max packet size – Get buffer – Get current frequency – Set current frequency USBD Audio Class calls: – “Get Max packet size” before opening data EP – “Get Buffer” before calling USB transmit of next packet – “Get/Set Current Frequency” after reception of related control
Speaker (node) BSP audio	Speaker node uses BSP audio to configure the codec and transmit data to the codec.
Mic (node) BSP audio	MIC node uses BSP audio to configure the microphone and receive data from the microphone.

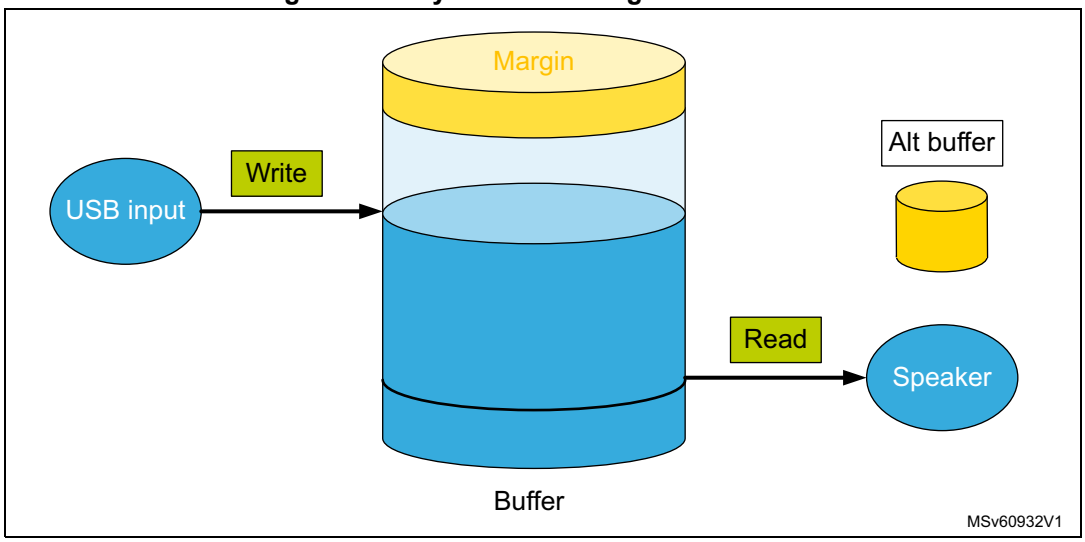
3.5 Buffer model

In order to ensure a regular sampling rate, a buffer is used. *Figure 11 on page 24* shows the buffer structure. As USB streaming is based on a producer-consumer model, the buffer is circular. Read and write pointers respectively provide information about from where to read new packets and where to write them.

In the playback case, the write pointer is managed by the USB-input node and the read pointer is handled by the speaker node. The DMA IP is used to transmit data to the SAI (and then to the codec) each time a packet is written to the codec. Packet size may be increased or decreased by one sample (for example when the sampling rate is 44100, the host sends alternately nine 44-sample packets, then a 45-sample packet.). In order to keep a contiguous memory area, a margin area is added to the buffer (colored in orange in *Figure 11*).

When there is not enough data to write to the SAI, an alternative buffer (Alt buffer) filled with zeros is used by the speaker. In addition, if the application changes samples (for example, resolution modification), it copies the result to the Alt buffer that becomes the DMA source. For recording, a similar buffer is used. The USB output node handles reading while the microphone node handles writing.

Figure 11. Playback streaming buffers model



3.6 Hardware related components

The physical layer and the cable detection are directly related to hardware and need the use of a set of hardware IPs. These IPs are detailed in *Table 4*.

Table 4. Hardware related IPs

IP	Use
USB-OTG	Handles the USB communication.
SAI	Used to transmit data to the codec.
DMA	Used to send data to the SAI, I^2S, and DFSDM.
Codec (WM8994)	Plays PCM samples and converts them to analog samples.
MEMS MIC	Records audio samples and provides PDM bit stream for the STM32446E-EVAL board. Samples are then converted to PCM using the PDM software library.
DFSDM MIC	Records audio samples and converts PDM to PCM. It provides a PCM bit stream for the 32F769IDISCOVERY board.
TIM4	Used to clock the MEMS microphone for STM32446E-EVAL board.

4 USB audio streaming expansion for STM32Cube™ library programming guidelines

The next subsections describe how to initialize the application and provide details about user structures and callbacks.

4.1 Expansion examples initialization

To use and initialize the expansion examples, the execution of the following steps is needed:

1. Initialize HAL: `HAL_Init`
2. Configure MCU clocks: system clock, USB clock 48 kHz
3. Initialize USB: `USBD_Init` & `USBD_RegisterClass` & `USBD_AUDIO_RegisterInterface`
4. Start USB: `USBD_Start`

4.2 USBD-Audio 10 class interface

The middleware class has three interfaces:

1. The first one is with the `USBD_Core` defined by the `USBD_ClassTypeDef` structure
2. The second is with the HAL defined in *usbd_core.h* which is used to open or close EP, transmits or receives data, and replies to the USB IP
3. The third one is with the application

The two first interfaces are similar to other classes and defined in [3]. The current description focuses on the third interface.

In order to communicate with the application, the `USBD_AUDIO_CLASS` uses the `USBD_AUDIO_InterfaceCallbacksfTypeDef` structure.

After calling the `Init` function, the whole audio function with all supported callbacks is defined in `USBD_AUDIO_FunctionDescriptionfTypeDef` parameters

Table 5. `USBD_AUDIO_InterfaceCallbacksfTypeDef` structure

`int8_t (*Init) (USBD_AUDIO_FunctionDescriptionfTypeDef* /* as_desc*/ , uint32_t /*privatedata*/)` `/* Called to initialize USB streaming sessions and nodes , as_desc contains the definition of audio function, it is called just after set configuration command*/`
`int8_t (*DeInit) (USBD_AUDIO_FunctionDescriptionfTypeDef* /* as_desc*/, uint32_t /*privatedata*/);` `/* Called to De-initialize USB streaming sessions and nodes*/`
`int8_t (*GetConfigDesc) (uint8_t ** /*pdata*/, uint16_t * /*psize*/, uint32_t /*private_data*/);` `/* Returns configuration descriptor */`
`int8_t (*GetState) (uint32_t privatedata);` `/* Not used*/`
`uint32_t private_data;` `/* Passed as parameters to all callbacks*/`

Table 6. USBD_AUDIO_FunctionDescriptionfTypeDef

<table>
<tr><td colspan="4">uint8_t as_interfaces_count /* number of supported streaming interfaces*/</td></tr>
<tr><td rowspan="16">USBD_AUDIO_AS_InterfaceTypeDef as_interfaces[USBD_AUDIO_AS_INTERFACE_COUNT]</td><td colspan="3">uint8_t interface_num /* the audio streaming interface number*/</td></tr>
<tr><td colspan="3">uint8_t max_alternate</td></tr>
<tr><td colspan="3">uint8_t alternate /* the current alternate*/</td></tr>
<tr><td rowspan="9">USBD_AUDIO_EP_DataTypeDef data_ep /* data endpoint informations*/</td><td colspan="2">uint8_t ep_num</td></tr>
<tr><td colspan="2">uint16_t control_name_map</td></tr>
<tr><td colspan="2">uint16_t control_selector_map/* define list of supported controls by EP*/</td></tr>
<tr><td colspan="2">uint8_t* buf</td></tr>
<tr><td colspan="2">uint16_t length</td></tr>
<tr><td colspan="2">int8_t (*DataReceived) /* callback after data reception */</td></tr>
<tr><td colspan="2">uint8_t* (*GetBuffer) /* callback to return buffer for data reception*/</td></tr>
<tr><td colspan="2">uint16_t (*GetMaxPacketLength)</td></tr>
<tr><td rowspan="5">USBD_AUDIO_EndPointControlCallbacksTypeDef control_cbk</td><td>int8_t (*GetCurFrequency)</td></tr>
<tr><td>int8_t (*SetCurFrequency)</td></tr>
<tr><td>uint32_t MaxFrequency;</td></tr>
<tr><td>uint32_t MinFrequency;</td></tr>
<tr><td>uint32_t ResFrequency;</td></tr>
<tr><td>uint8_t synch_enabled</td><td colspan="2">/* True when feedback synchronization is enabled */</td></tr>
<tr><td rowspan="3">USBD_AUDIO_EP_SynchTypeDef synch_ep</td><td colspan="2">uint8_t ep_num</td></tr>
<tr><td colspan="2">uint8_t feedback_data[AUDIO_FEEDBACK_EP_PACKET_SIZE]</td></tr>
<tr><td colspan="2">uint32_t (*GetFeedback)</td></tr>
<tr><td>void (*SofReceived)</td><td colspan="2">/* callback when Sof is enabled*/</td></tr>
<tr><td>int8_t (*SetAS_Alternate)</td><td colspan="2">/* callback to start or stop streaming */
uint32_t private_data</td></tr>
<tr><td colspan="4">uint8_t control_count</td></tr>
</table>

Table 6. `USBD_AUDIO_FunctionDescriptionfTypeDef` (continued)

USBD_AUDIO_ControlTypeDef controls[USBD_AUDIO_CONFIG_CONTROL_UNIT_COUNT]	uint8_t id		
	USBD_AUDIO_SpecificACInterfaceDescSubtypeTypeDef type		
	uint16_t control_req_map		
	uint16_t control_selector_map		
	USBD_AUDIO_ControlCallbacksTypeDef Callbacks	USBD_AUDIO_FeatureControlCallbacksTypeDef* feature_control	int8_t (*GetMute)
			int8_t (*SetMute)
			int8_t (*SetCurVolume)
			int8_t (*GetCurVolume)
			uint16_t MaxVolume
			uint16_t MinVolume
			uint16_t ResVolume

4.3 Common USB streaming structures and session structure

Table 7. AUDIO_USB_StreamingSession_t

AUDIO_Session_t session /*default session structure which is common to all sessions */	AUDIO_Node_t * node_list; /* list of session's nodes*/
	AUDIO_SessionState_t state;
	int8_t (*SessionCallback) /** callback called by session's nodes*/
int8_t (*SessionDeInit) /* called when USB is unplugged */	
uint8_t interface_num /* audio streaming interface */	
uint8_t alternate /* current alternate interface*/	
AUDIO_CircularBuffer_t buffer /* the main audio data buffer */	uint8_t* data
	uint16_t rd_ptr
	uint16_t wr_ptr
	uint16_t size

4.3.1 Generic node structure

Table 8. `AUDIO_Node_t`

`AUDIO_NodeState_t state;`	
`AUDIO_Description_t* audio_description;`	`uint32_t frequence; /* current frequency*/`
	`uint8_t channels_count; /* current channel count : 2 */`
	`uint16_t channels_map; /* current channel map */`
	`uint16_t audio_type; /* current channel type: current value PCM */`
	`int audio_volume_db_256; /* current volume*/`
	`uint8_t audio_mute; /* mute state*/`
	`uint8_t resolution; /* audio resolution*/`
`AUDIO_NodeType_t type;`	
`struct AUDIO_Session* session_handle;`	
`struct AUDIO_Node* next;`	

4.3.2 USB input/output node structure

Table 9. `AUDIO_USBInputOutputNode_t`

`AUDIO_Node_t node; /* generic node structure , must be first field */`
`uint8_t flags;`
`AUDIO_CircularBuffer_t* buf; /* buffer to use */`
`uint16_t max_packet_length; /* the packet to read each time from buffer */`
`uint16_t packet_length; /* the packet normallength */`
`int8_t (*IODeInit) (uint32_t /*node_handle*/);`
`int8_t (*IOStart) (AUDIO_BufferTypeDef* buffer, uint16_t thershold, uint32_t /*node handle*/); int8_t (*IOChangeFrequency);`
`int8_t (*IORestart) ( uint32_t /*node handle*/);`
`int8_t (*IOStop) ( uint32_t /*node handle*/);`
`Union { AUDIO_USBInputSpecifcParams_t input; AUDIO_USBOutputSpecifcParams_t output; }specific;`

4.3.3 USB feature unit node structure

Table 10. AUDIO_USB_CF_NodeTypeDef

AUDIO_Node_t node; /* generic node structure , must be first field */	
uint8_t unit_id; /* UNIT ID for usb audio function description and control*/	
USBD_AUDIO_FeatureControlCallbacksTypeDef usb_control_callbacks; /* list of callbacks */	int8_t (*GetMute)
	int8_t (*SetMute)
	int8_t (*SetCurVolume)
	int8_t (*GetCurVolume)
	uint16_t MaxVolume;
	uint16_t MinVolume;
	uint16_t ResVolume;
	int8_t (*GetStatus)
AUDIO_USBFeatureUnitCommands_t control_cbks; /* */	int8_t (*SetMute)
	int8_t (*SetCurrentVolume)
	uint32_t private_data;
int8_t (*CFInit)	
int8_t (*CFDeInit)	
int8_t (*CFStart) int8_t (*CFStop) int8_t (*CFSetMute)	

4.3.4 Speaker node structure

Table 11. AUDIO_SpeakerNode_t

AUDIO_Node_t	node;	/* the structure of generic node*/
AUDIO_CircularBuffer_t*	buf;	/* the audio data buffer*/
uint16_t	packet_length;	/* packet maximal length */
uint16_t	packet_length_max_44_1;	
uint8_t	injection_44_count;	
uint8_t	injection_45_pos;	
int8_t	(*SpeakerDeInit)	

Table 11. `AUDIO_SpeakerNode_t` (continued)

int8_t	(*SpeakerStart)
int8_t	(*SpeakerStop)
int8_t	(*SpeakerChangeFrequency)
int8_t	(*SpeakerMute)
int8_t	(*SpeakerSetVolume)
int8_t	(*SpeakerStartReadCount)
uint16_t	(*SpeakerGetReadCount)

4.3.5 Microphone node structure

Table 12. `AUDIO_MicNode_t`

AUDIO_Node_t node; /* generic node structure*/
AUDIO_CircularBuffer_t *buf;
uint16_t packet_length;
uint8_t volume;
int8_t (*MicDeInit)
int8_t (*MicStart)
int8_t (*MicStop)
int8_t (*MicChangeFrequence)
int8_t (*MicMute)
int8_t (*MicSetVolume)
int8_t (*MicGetVolumeDefaultsValues)
int8_t (*MicStartReadCount)
uint16_t (*MicGetReadCount)
AUDIO_Mic_SpecificTypeDef specific;
AUDIO_MicrophoneSpecificParams_t specific; /*should be defined by user for user speaker */

4.4 Compilation and user flags

In order to support a variety of features, a set of macros and flags must be defined by the user. It comprises compilation flags and a list of defines, which are mainly defined in file *usb_audio_user.h*. *Table 13* and *Table 14* detail the flags and macros.

Table 13. List of compilation flags

Compilation symbols	Description
USE_USB_FS	Used to support FS speed for USB.
USE_USB_FS_INTO_HS	Used to support FS speed in USB OTG-HS IP.

Table 13. List of compilation flags (continued)

Compilation symbols	Description
USE_USB_AUDIO_CLASS_10	Defines the usage of UAC 1.0.
USE_USB_AUDIO_PLAYPBACK=1	Defines that playback function is supported.
USE_USB_AUDIO_RECORDING=1	Defines that playback function is supported
USE_AUDIO_MEMS_MIC=1	Used only when recording function is defined, it activates the support of MEMS microphone (with software PDM to PCM conversion for the STM32446E-EVAL board).
USE_AUDIO_DFSDM_MEMS_MIC=1	Used only when recording function is defined, it activates support of MEMS microphone plus DFSDM IP (with hardware PDM to PCM conversion for the 32F769IDISCOVERY board)
USE_AUDIO_DUMMY_MIC=1	Used only when recording function is defined, it activates support of dummy microphone (which provides “zeros” packets).

Table 14. Main user defined macros defined in *usb_audio_user_cfg.h*

User Defines	Description
USB_AUDIO_CONFIG_PLAY_USE_FREQ_96_K USB_AUDIO_CONFIG_PLAY_USE_FREQ_48_K USB_AUDIO_CONFIG_PLAY_USE_FREQ_44_1_K USB_AUDIO_CONFIG_PLAY_USE_FREQ_32_K USB_AUDIO_CONFIG_PLAY_USE_FREQ_16_K USB_AUDIO_CONFIG_PLAY_USE_FREQ_8_K	Defines the supported frequencies by setting the value to “1” when the frequency is supported.
USE_USB_AUDIO_CLASS_10	USB Audio Class 1.0 support.
USE_AUDIO_PLAYBACK_USB_FEEDBACK	Used only when the playback function is defined; activates the support of feedback synchronization.
USB_AUDIO_CONFIG_PLAY_CHANNEL_COUNT	Playback channels count. Supported value is 2.
USB_AUDIO_CONFIG_PLAY_CHANNEL_MAP	Playback channels spatial mapping. Supported value is 3 (left and right).
USB_AUDIO_CONFIG_PLAY_RES_BIT	Playback audio resolution in bits. Possible values are 16 and 24.
USB_AUDIO_CONFIG_PLAY_RES_BYTE	Playback audio resolution in bytes. Possible values are 2 (for 16-bit audio) and 3 (for 24-bit audio).
USE_AUDIO_TIMER_VOLUME_CTRL	Used only when the playback function is defined. It activates a workaround implementation to change audio using a low-priority timer interrupt. If it is not deployed, the volume is changed inside the USB interrupt handler, which causes glitches when the volume is changed swiftly.
USB_AUDIO_CONFIG_PLAY_BUFFER_SIZE	Size of the playback buffer. The heap size must be changed accordingly when this value is changed.
USB_AUDIO_CONFIG_RECORD_CHANNEL_COUNT	Recording channels count. Supported value is 2.
USB_AUDIO_CONFIG_RECORD_CHANNEL_MAP	Recording channels spatial mapping. Supported value is 3 (left and right).

Table 14. Main user defined macros defined in *usb_audio_user_cfg.h* (continued)

User Defines	Description
USB_AUDIO_CONFIG_RECORD_RES_BIT	Recording audio resolution in bits. Possible values are 16 and 24.
USB_AUDIO_CONFIG_RECORD_RES_BYTE	Recording audio resolution in bytes. Possible values are 2 (for 16-bit audio) and 3 (for 24-bit audio).
USE_AUDIO_RECORDING_USB_IMPLICIT_SYNCHRO	Used only when the recording function is defined; activates implicit feedback.
USE_AUDIO_RECORDING_USB_NO_REMOVE	Used only when the recording function is defined as well as USE_AUDIO_RECORDING_USB_IMPLICIT_SYNCHRO. When it is activated, instead of being removed, samples are sent to the host by augmenting the maximum value of the packet size.
USB_AUDIO_CONFIG_RECORD_USE_FREQ_96_K USB_AUDIO_CONFIG_RECORD_USE_FREQ_48_K USB_AUDIO_CONFIG_RECORD_USE_FREQ_44_1_K USB_AUDIO_CONFIG_RECORD_USE_FREQ_32_K USB_AUDIO_CONFIG_RECORD_USE_FREQ_16_K USB_AUDIO_CONFIG_RECORD_USE_FREQ_8_K	Defines the supported frequencies by setting the value to “1” when the frequency is supported.
USB_AUDIO_CONFIG_RECORD_BUFFER_SIZE	Defines recording streaming buffer size.
USBD_VID	Device vendor ID. Defined in *usbd_desc.c.*
USBD_PID	Device product ID. Defined in *usbd_desc.c.*
USBD_MANUFACTURER_STRING	Device manufacturer name. Defined in *usbd_desc.c.*

5 Example description

The package contains two projects. Each project has four configurations:

1. [BOARDNAME]_UAC10-PLAY
   This configuration is used for standalone playback.
   The board is declared as a USB speaker in the host audio devices panel.
2. [BOARDNAME]_UAC10-REC
   This configuration is used for standalone recording.
   The board is declared as a USB microphone in the host audio devices panel.
3. [BOARDNAME]_UAC10-DUM
   This configuration is used for standalone recording.
   The board is declared as a USB microphone in the host audio devices panel. It is a dummy microphone which sends zeros data. In this case, the 96 kHz frequency is supported while it is not supported with MEMS microphone.
4. [BOARDNAME]_UAC10-ADV
   This configuration is used for simultaneous playback and recording.
   STM32 speaker and STM32 microphone are added to the host audio devices panel.

After choosing a configuration, the user must modify the compilation macro and the defined macro if needed (for example with the proper list of supported frequencies).

5.1 STM32446E-EVAL board standalone playback example

This example is detailed on basis of the sequence diagrams in *Figure 12*, *Figure 13* and *Figure 14*.

Streaming Initialization steps

The steps are illustrated by the sequence diagram in *Figure 12*.

1: The receiving USB sets configuration control request (occurring after plug or USB resume)

2: The request is transmitted from the HAL driver to the `USBD_Core`

3, 4, 5: `USBD_Core` sets configuration

6: The USBD Audio Class Init callback is called.

7, 8: USBD Audio Class handler is allocated, then the application audio interface `Init` callback is called

9, 10, 11: Playback session is initialized, audio buffer is allocated. Next field of `USBD_AUDIO_FunctionDescriptionfTypeDef` parameter (first arguments of `Init` callback) are set:

- `synch_enabled`
- `synch_ep`
- `SofReceived` callback
- `SetAS_Alternate` callback

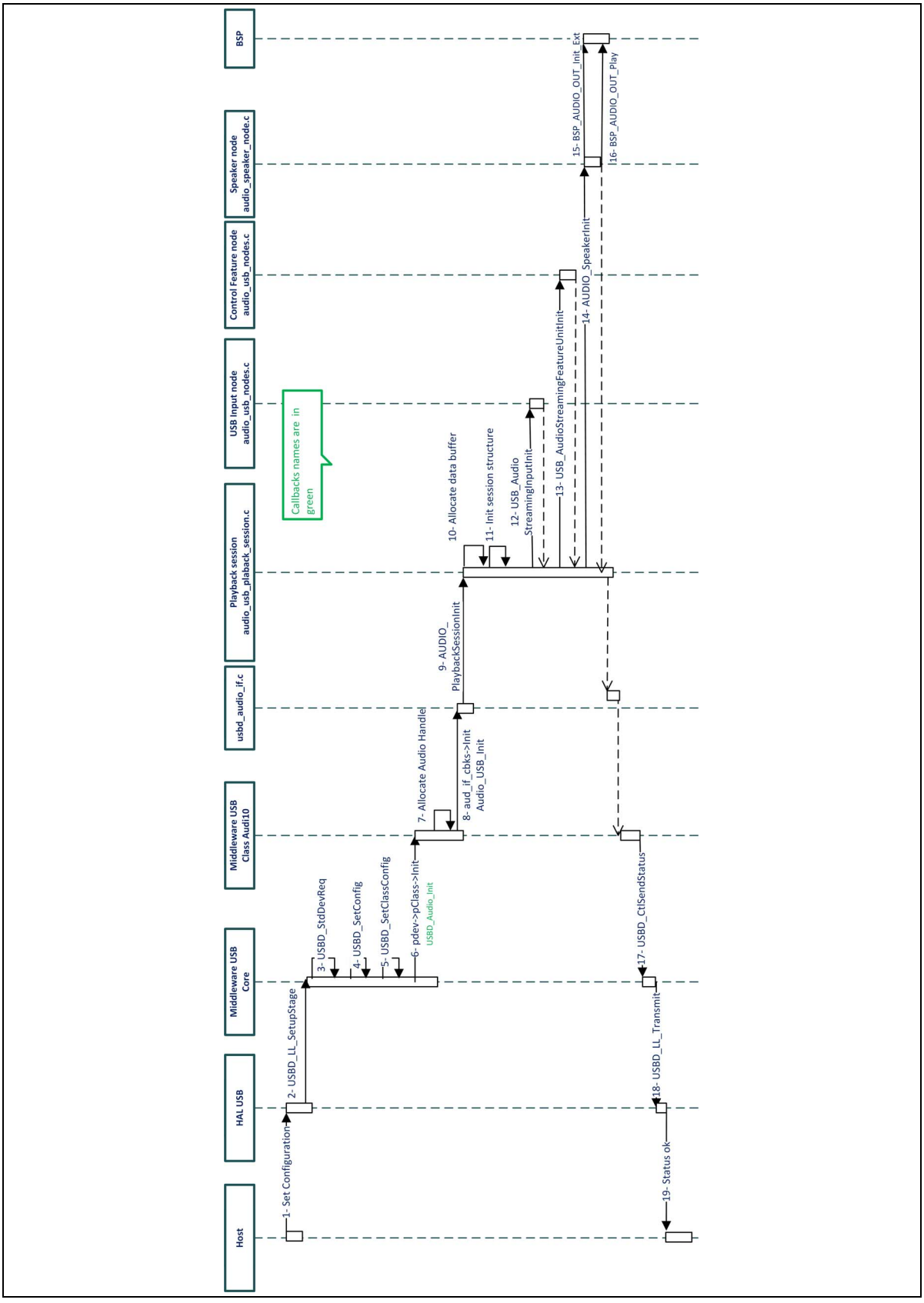
12: USB audio Input node is initialized

13: USB feature unit node is initialized

14, 15, 16: Speaker node is initialized, then SAI is initialized and codec is initialized. The transmission of zero samples is started from the alternate buffer (speaker node) to the SAI IP

17, 18, 19: Positive status is prepared to be sent in handshake stage

Figure 12. Standalone playback: initialization



Playback streaming start steps

The steps are illustrated by the sequence diagram in *Figure 13*.

1: The host sends command set interface alternate setting “1” for the playback audio streaming interface

2: The request is transmitted from the HAL driver to `USBD_Core`

3, 4: `USBD_Core` transmits the request to the Audio Class

5: Set alternate setting function is called

6: `USB_AudioPlaybackSetAudioStreamingInterfaceAlternateSetting` callback is called. It informs the application about the changes in AS interface alternate setting

7: Playback session start is called

8, 9: USB Input node start is called

10, 11: Control feature start is called

12: Playback session is started

13: Middleware Audio Class requests the maximum packet size from the USB Input node

14. Data EP (isochronous) is opened

15: Middleware Audio Class requests the buffer from the USB Input node

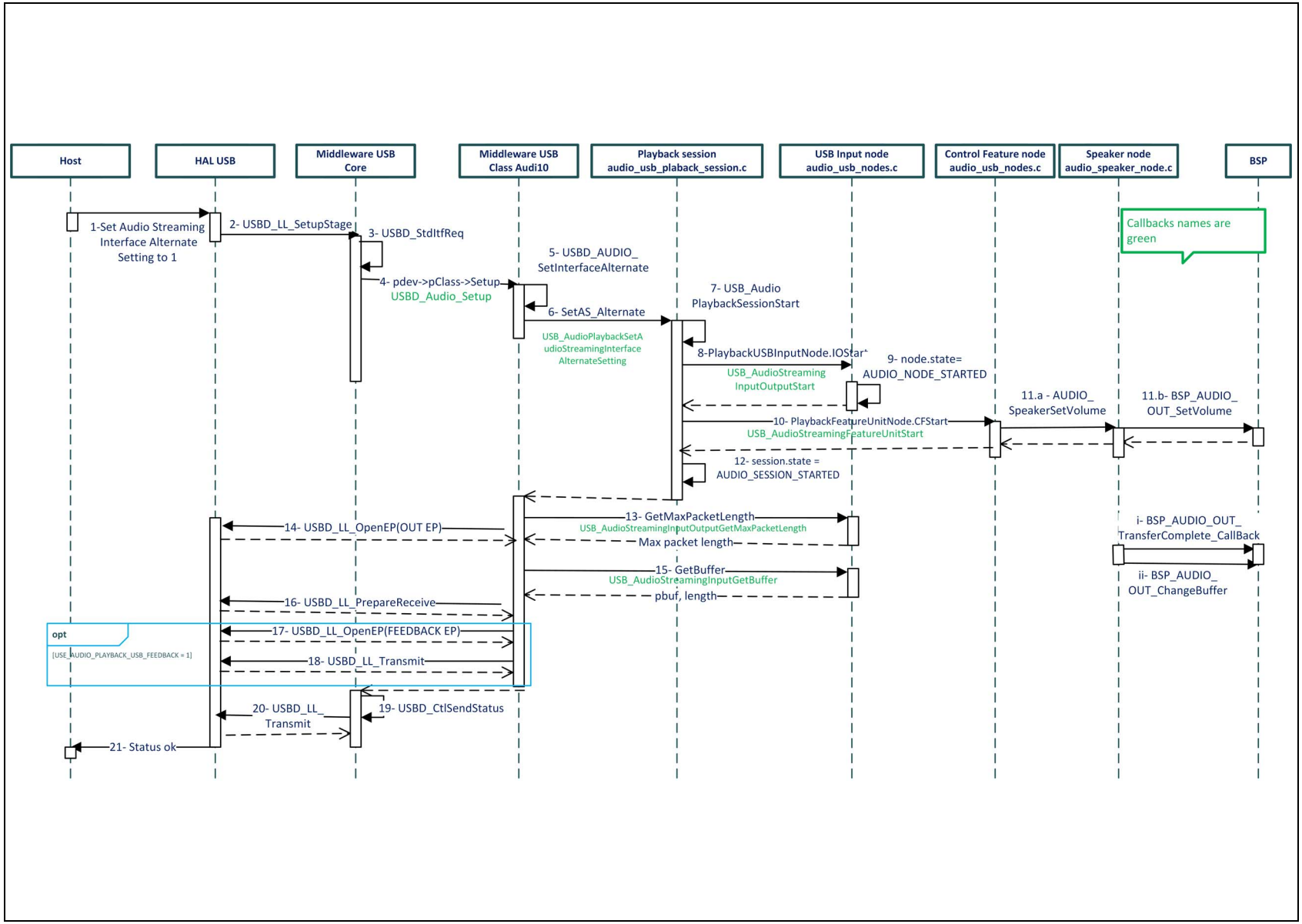
16: Middleware Audio Class starts receiving packets

17, 18: If feedback synchronization is activated, then middleware Audio Class opens feedback EP and starts transmitting data rate

19, 20: Prepare positive handshake

i, ii: As audio SAI DMA is already programmed, it generates an interrupt when the transfer is completed. When this interrupt is received, a new transfer is launched

Figure 13. Standalone playback: starting streaming



After starting the playback session, the USB input writes the received data in the main buffer. When the buffer threshold is reached, the speaker node starts playing real data. *Figure 14* shows the sequence diagram of the speaker start.

1: After setting alternate interface to “1”, the host sends audio sample packets

2: Received data transfer completed event is raised from HAL driver to `USBD_Core`

3: `USBD_Core` transmits the request to the Audio Class

4: The `DataReceived` application is called

5: If it is the first packet, a notification is sent to the session

6: If the buffer filling threshold is reached, a notification is sent to the session

7, 8, 9, 10: The speaker node starts reading data from the main buffer and plays it

11: If 6 and 7 are not present, an `AUDIO RECEIVED PACKET` notification is sent to the session

12: USB Input node increments the write pointer

13, 14: Middleware USB Audio Class prepares the next buffer to receive the next packet

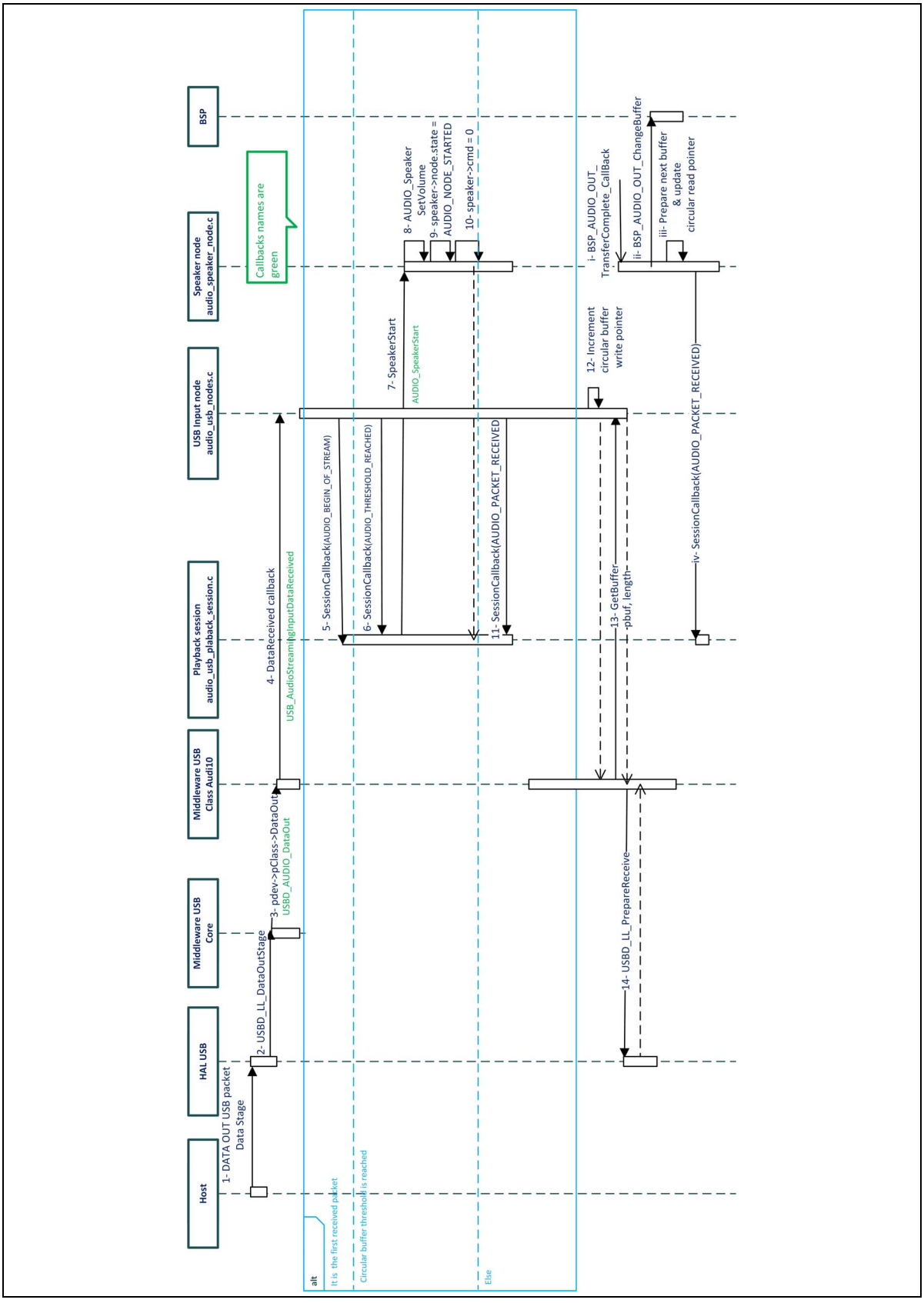


Figure 14. Standalone playback: start `Speaker` node when the circular buffer threshold reached

6 Frequently asked questions

Q: How can I get the USB audio streaming Expansion Package for STM32Cube™?

A: The library sources are provided for free download from the STMicroelectronics web site at *www.st.com*.

Q: Which USB Audio Class is supported by X-CUBE-USB-AUDIO?

A: Currently, the X-CUBE-USB-AUDIO Expansion Package supports UAC 1.0.

Q: Which boards are supported by X-CUBE-USB-AUDIO?

A: The X-CUBE-USB-AUDIO Expansion Package supports STM32446E-EVAL and 32F769IDISCOVERY. It is extensible to other boards and STM32 MCUs.

Q: Which nodes need changing to support a new codec or a microphone?

A: The speaker node, microphone node, and BSP part must be changed.

7 Revision history

Table 15. Document revision history

Date	Revision	Changes
26-Feb-2019	1	Initial release.