UM1897
User manual

Getting started with the OSXBLUEVOICE Bluetooth LE and digital MEMS microphone software expansion for STM32Cube

Introduction

The OSXBLUEVOICE library implements a BlueVoice vendor-specific profile based on the Bluetooth 4.0/4.1 specification and designed for systems adopting the BlueNRG/BlueNRG-MS Bluetooth low energy network processor, digital MEMS microphones and the STM32 MCU.

The OSXBLUEVOICE library (under OPEN.Audio license) is implemented in the BLUEVOICELINK1 sample application; BLUEVOICELINK1 is part of the OPEN.Framework program while OSXBLUEVOICE is part of the OPEN.Audio program available for free source code download. The OSXBLUEVOICE library is also part of the BLUEMICROSYSTEM2 sample application that can stream audio from the ST platform to the BlueMS app, available for Android™ or iOS™.

Information about STM32Cube is available on st.com at: *http://www.st.com/stm32cube*.

Contents

List of tables

List of figures

1 Acronyms and abbreviations

Table 1: Acronyms and abbreviations

Term	Description
MEMS	Micro electro-mechanical systems
PDM	Pulse density modulation
PCM	Pulse code modulation
ADPCM	Adaptive differential pulse code modulation
USB	Universal Serial Bus
BSP	Board support package
HAL	Hardware abstraction layer
BLE	Bluetooth Low Energy
IDE	Integrated development environment
UUID	Universally unique identifier
ATT	Attribute protocol
GATT	Generic attribute profile
GAP	Generic access profile

2 OSXBLUEVOICE library functional description

2.1 Overview

The key features of the OSXBLUEVOICE package are:

- Half-duplex or simplex voice over Bluetooth low energy communication profile.
- Based on the very low power Bluetooth low energy single-mode network processor, compliant with Bluetooth specifications core 4.0 (BlueNRG) and 4.1 (BlueNRG-MS).
- Designed for optimal performance in applications using digital MEMS microphones (e.g., MP34DT01-M).
- Digital audio signal processing.
- Easy portability across different STM32 MCU families thanks to modular architecture and STM32Cube
- Free user-friendly license terms
- BLUEVOICELINK1 sample implementation available on X-NUCLEO-IDB04A1/X-NUCLEO-IDB05A1 and X-NUCLEO-CCA02M1 connected to a NUCLEO-F401RE, NUCLEO-L476RG or NUCLEO-L053R8 board
- BLUEMICROSYSTEM2 sample implementation available on STEVAL-STLKT01V1 or X-NUCLEO-CCA02M1, X-NUCLEO-IKS01A1 and X-NUCLEO-IDB04A1/X-NUCLEO-IDB05A1 boards connected to a NUCLEO-F401RE or NUCLEO-L476RG board

The software is based on the STM32CubeHAL hardware abstraction layer for the STM32 microcontroller.

The OSXBLUEVOICE profile defines a BLE service with a characteristic for audio transmission and one for synchronization. In a half-duplex system, both sides of communication (central and peripheral) can act as information servers. Notifications containing compressed audio data are sent periodically from a server to a client via the selected central-to-peripheral or peripheral-to-central channel. The OSXBLUEVOICE middleware handles audio encoding and periodic data transmission on the server side and decoding received voice data on the client side.

The X-CUBE-BLE1 is an expansion software package for STM32Cube which runs on the STM32 and includes drivers for the BlueNRG/BlueNRG-MS Bluetooth low energy device.

X-CUBE-MEMSMIC1 is an expansion software package for STM32Cube with drivers and middleware for audio data acquisition from MEMS digital microphones (MP34DT01-M) and USB streaming of recorded signals.

BLUEVOICELINK1 is a sample application on www.st.com that the developer can use to start experimenting with the code. It enables the acquisition, compression and transmission of voice data from the acting transmitter module to acting receiver, via the Bluetooth low energy protocol. The receiver handles audio decompression and USB streaming of audio data to a PC.

Any freeware or commercial audio recording software can be used to interface with the system. The peripheral module can also stream audio to an Android™/iOS™ device running the ST BlueMS app v3.0.0 or higher.

3 BlueVoice profile description

3.1 Bluetooth Low Energy

This section describes the design of the application with regard to the Bluetooth Low Energy architecture.

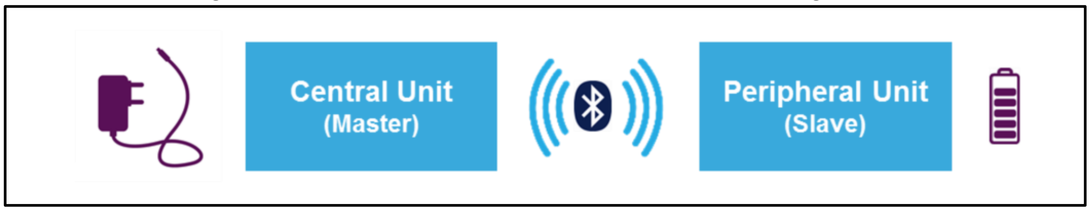
3.1.1 Generic access profile (GAP)

Bluetooth Low Energy 4.0 communications can be either broadcast or connection-based. The BlueVoice application deploys a connection-based communication paradigm insofar as it provides a permanent point-to-point link between two devices. Data sent through a BLE connection is organized through an additional protocol layer, the Generic Attribute Profile (GATT).

Connections involve two separate roles, as specified in Bluetooth Specification v 4.1 [*1*]:

- The **Central** role supports multiple connections and is the initiator for all connections with devices in the peripheral role. Devices supporting the central role require a controller that supports the controller’s master role and generally supports more complex functions compared to the other BLE GAP roles.
- The **Peripheral** role is optimized for devices that support a single connection and are less complex than central devices. Devices supporting the peripheral role may just support the controller’s slave role. They follow the Central’s (master) timing to exchange data with it.

Figure 1: BlueVoice Profile master-slave GAP role assignment



Central and peripheral role assignement is consistent with the asymmetric design concept of BLE where the device with a lower energy source is given less to do: a slave cannot initiate complex procedures, whereas a master manages communication timing, adaptive frequency hopping, sets encryption, and so on. A portable device provided with a coin-size battery is typically suitable to be implemented as a slave. Nevertheless, it must be noted that, according to the specification [*1*] (see also [*3*]), data can be sent independently by either device at each connection event and the roles do not impose restrictions in data throughput or priority. In a half-duplex communication scheme, BlueVoice role assignment is therefore decoupled from "transmitter" or "receiver" functionality.

3.1.2 Generic attribute profile (GATT)

The Bluetooth SIG GATT specification provides a set of predefined profiles to ensure interoperability between devices from different vendors that implement use cases such as Proximity Profile, to detect the presence or absence of nearby devices, Glucose Profile to transfer glucose levels, or Health Thermometer Profile, to transfer body temperature readings. In addition, the Bluetooth specification allows custom definition of so-called vendor-specific profiles for use cases not covered by the standard.

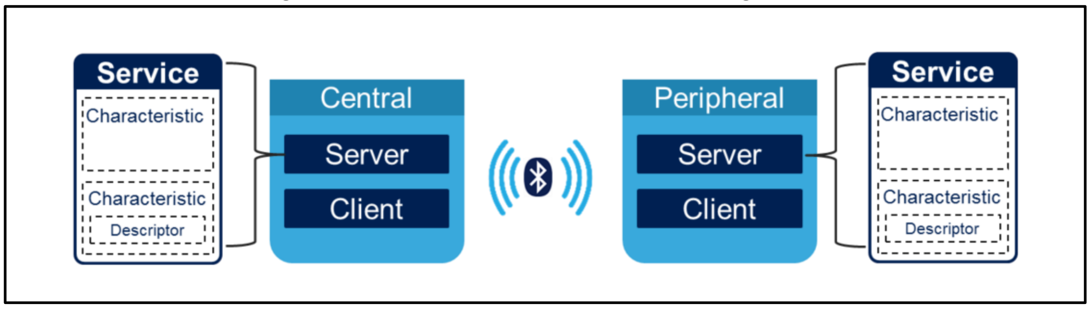
GATT defines client and server roles of interacting devices that are independent of the master-central and slave-peripheral GAP roles described above:

- **Client** performs service discovery to inquire about presence and nature of server attributes. It sends requests to a server and accepts responses and server-initiated updates.
- **Server** accepts requests, commands and confirmations from a client and sends responses back. It can also send server-initiated updates. It stores data organized in attributes as specified in the ATT (Attribute Protocol).

Considering a monodirectional audio streaming asymmetric system, the only device that has voice information is the one provided with a microphone that must therefore be considered the server of the communication. The other device acts as client, sending request to the server and accepting server-initiated updates containing audio data.

In a bidirectional system, where voice signals travel in either direction, the architecture is symmetric: both the central and the peripheral modules are provided with a microphone and may act as servers, exporting audio data organized in attributes. At the same time, they can also act as clients, sending requests and accepting initiated updates from the other module. The illustration below shows this mechanism: both Central and Peripheral act as Server and Client.

Figure 2: BlueVoice Profile GATT role assignment

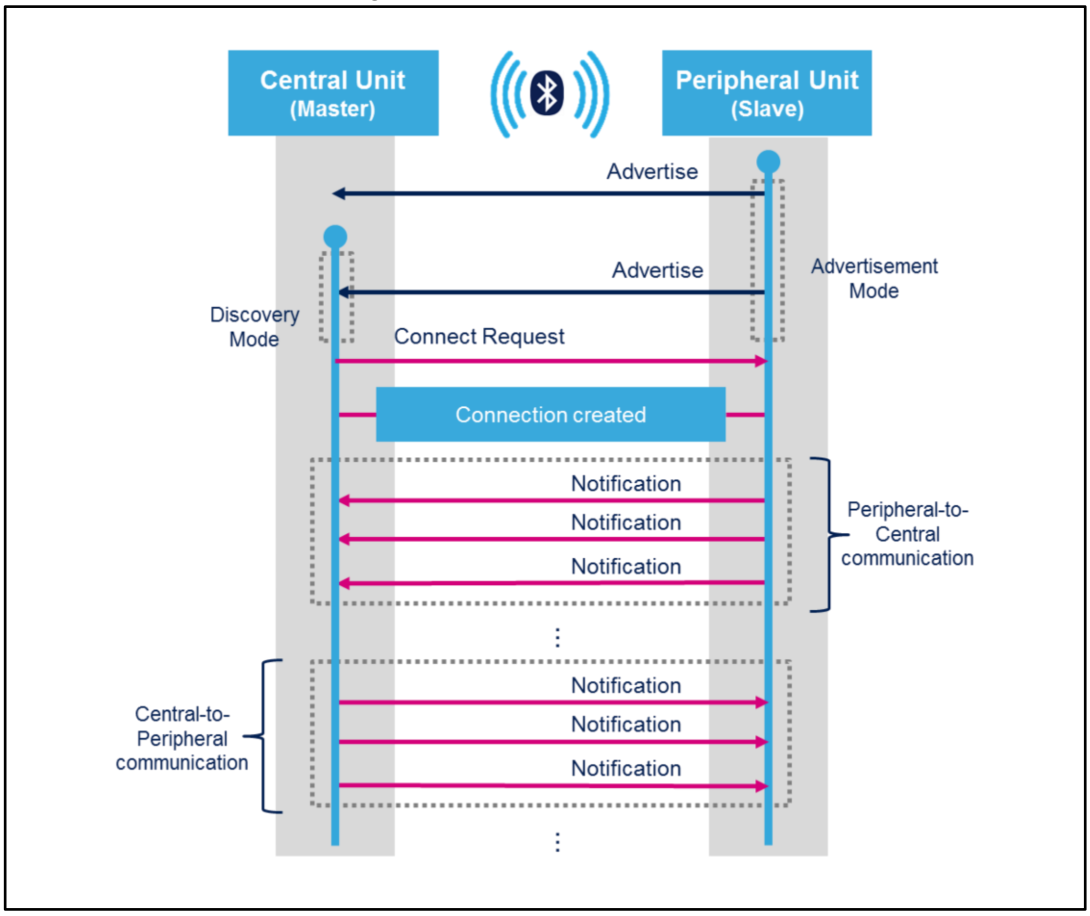


In both directions, streaming audio data transmission is based on periodic server-to-client notifications which do not require a request or response from the receiving device. Server-initiated updates are sent as asynchronous notification packets which include the handle of a characteristic value attribute along with its current value.

3.1.3 Connection establishment

The diagram below shows an example of how a connection between two nodes starts.

Figure 3: BLE connection creation



According to the BLE specification, at power-up the peripheral unit goes into advertising mode and starts sending advertising packets at low frequency. Conversely, the central unit module goes in discovery mode, looking for other devices. As soon as it detects the presence of a slave device by receiving an advertisement packet, it sends a connection request. After the connection establishment phase, notifications carrying audio data are periodically sent from server to client, according to the selected direction: peripheral-to-central or central-to-peripheral.

3.1.4 BlueVoice service

The Attribute Protocol (ATT) is used by GATT as the transport protocol for exchanging data between devices. The smallest entities defined by ATT, named attributes, are addressable pieces of information that may contain user data or meta-information regarding the architecture of the attributes themselves, as stored in the server and as exchanged between client and server.

Attributes are described in the following fields (this section is taken from [4], which includes an in-depth description of the heart rate service):

- **Handle**: a unique 16-bit identifier for each attribute on a particular GATT server; it makes each attribute addressable, and it is guaranteed not to change.
- **Type**: a 16-, 32-, or 128-bit UUID (universally unique identifier) that determines the kind of data present in the value of the attribute. Apart from standard and profile

UUIDs, proprietary and vendor-specific UUIDs can also be used in custom implementations like BlueVoice.

- **Permissions**: metadata describing ATT data access permissions, encryption, and authorization.
- **Value**: the actual data content of the attribute; this is the part of an attribute that a client can access (if permitted) to both read and write.

GATT server attributes are organized as a sequence of services, each one starting with a service declaration attribute marking its beginning. Each service groups one or more characteristics and each characteristic can include zero or more descriptors.

Since audio streaming is not part of the predefined set of profiles, the BlueVoice application defines a vendor-specific service named BlueVoice Service which exposes a user’s voice to a client device.

Table 2: BlueVoice service definition

Type	Handle	UUID	Permissions	Value
Service	Att1	0x2800	READ	audio_adpcm_serv_uuid
Audio characteristic	Att1	0x2803	READ	NOT|0x0012|Audio UUID
	Att2	audio_adpcm_char_uuid	NONE	Audio data
	Att3	0x2902	READ/WRITE	Client characteristic configuration
Sync characteristic	Att1	0x2803	READ	NOT|0x0015|Sync UUID
	Att2	audio_adpcm_synch_char_uuid	NONE	Syn data
	Att3	0x2902	READ/WRITE	Client characteristic configuration

From the table, the BlueVoice service is described by the following attributes:

- Att1 contains the service declaration for the BlueVoice service with:
  - UUID: standard 16-bit UUID for a primary service declaration, UUID primary service (0x2800).
  - Permissions: Read.
  - Value: the value is the proprietary 128-bit UUID for the BlueVoice Service (UUID: 00000000-0001-11e1-9ab4-0002a5d5c51b).

The BlueVoice service: Audio and Sync. Audio characteristic expose actual compressed audio data, while Sync characteristic expose side information used to implement a synchronization mechanism.

They are structured as follows:

Audio Characteristic

- Att1 contains the Audio characteristic declaration. Its attribute fields are:
  - UUID: standard 16-bit UUID for a characteristic declaration, UUID characteristic (0x2803).
  - Permissions: Read.
  - Value: the properties for this characteristic are notify only and the UUID is for Audio Data (UUID: 08000000-0001-11e1-ac36-0002a5d5c51b).
- Att2 contains the characteristic value, in this case audio data. Its attribute fields are:
  - UUID: the same UUID in the last 16 bytes of the characteristic definition attribute value.
  - Permissions: None.
  - Value: the actual audio data.

- Att3 contains the client characteristic configuration, defining how the characteristic may be configured by a specific client. Its attribute fields are:
  - UUID: the UUID is the standard 16-bit UUID for a client characteristic configuration (0x2902).
  - Permissions: Read/Write.
  - Value: Bit 0 Notifications disabled/enabled; Bit 1 Indications disabled/enabled

Sync Characteristic

- Att1 contains the synchronization characteristic declaration. Its attribute fields are the following:
  - UUID: the UUID is the standard 16-bit UUID for a characteristic declaration, UUID characteristic (0x2803).
  - Permissions: Read.
  - Value: the properties for this characteristic are notify only and the UUID is for Sync-Peripheral Data (UUID: 40000000-0001-11e1-ac36-0002a5d5c51b).
- Att2 contains the characteristic value, in this case sync data. Its attribute fields are the following:
  - UUID: the same UUID present in the last 16 bytes of the characteristic definition’s attribute value.
  - Permissions: None.
  - Value: the actual sync data.
- Att3 contains the client characteristic configuration, defining how the characteristic may be configured by a specific client. Its attribute fields are:
  - UUID: standard 16-bit UUID for a client characteristic configuration (0x2902).
  - Permissions: Read/Write.
  - Value: Bit 0 Notifications disabled/enabled; Bit 1 Indications disabled/enabled.

BlueVoice UUIDs are summarized in the following table:

Table 3: BlueVoice UUID

UUID name	UUID
audio_adpcm_serv_uuid	00000000-0001-11e1-9ab4-0002a5d5c51b
audio_adpcm_char_uuid	08000000-0001-11e1-ac36-0002a5d5c51b
audio_adpcm_sync_char_uuid	40000000-0001-11e1-ac36-0002a5d5c51b

Given the hierarchical architecture of BLE services, further characteristics may be added to the BlueVoice service in future releases of the BlueVoice application, such as allowing a client to configure parameters like volume, enabling/disabling of processing algorithms, etc...

The BlueVoice profile exported by the server exposes data type, format and access details to client devices.

3.2 Audio processing

The audio processing component of the osxBlueVoice application is designed to achieve an audio sampling frequency of 8 or 16 kHz at the receiver side, with a trade-off between audio quality and bandwidth occupation for voice signals. Audio signals transmitted over the BLE link is compressed via ADPCM (adaptive differential pulse code modulation) to fit in the available data rate while minimizing radio transmission time and power consumption. On a half-duplex channel (two simplex communication channels in opposite directions), one node acts as the Tx module and the other acts as the Rx module.

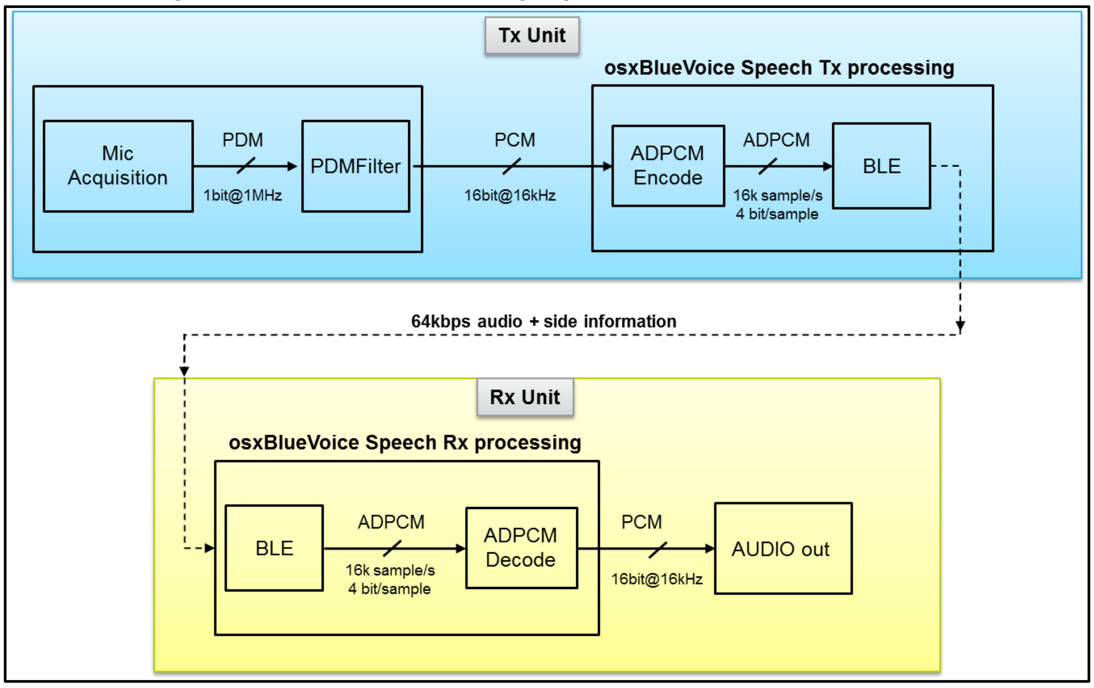
Figure 4: "BlueVoiceLink chain – peripheral to central communication" shows the speech processing chain in a complete communication system with Tx and Rx.

On the Tx side, the library receives an audio signal which is typically acquired by a digital MEMS microphone as a 1-bit PDM signal at 1 MHz and converted by a PDM to PCM conversion filter into 16-bit PCM samples at 16 kHz.

The library can be provided with 1, 2, 5 or 10 ms of audio data. When the compressed output buffer is ready, a flag is set and audio data is streamed via BLE together with collateral ADPCM information.

The resulting communication bandwidth is 64 kbps (with 16 kHz audio sampling frequency) or 32 kbps (with 8 kHz audio sampling frequency) of audio data plus 300 bps of collateral information. On a half-duplex implementation, the same processing chain is implemented and the same bandwidth is used for communication in the opposite direction.

Figure 4: BlueVoiceLink chain – peripheral to central communication



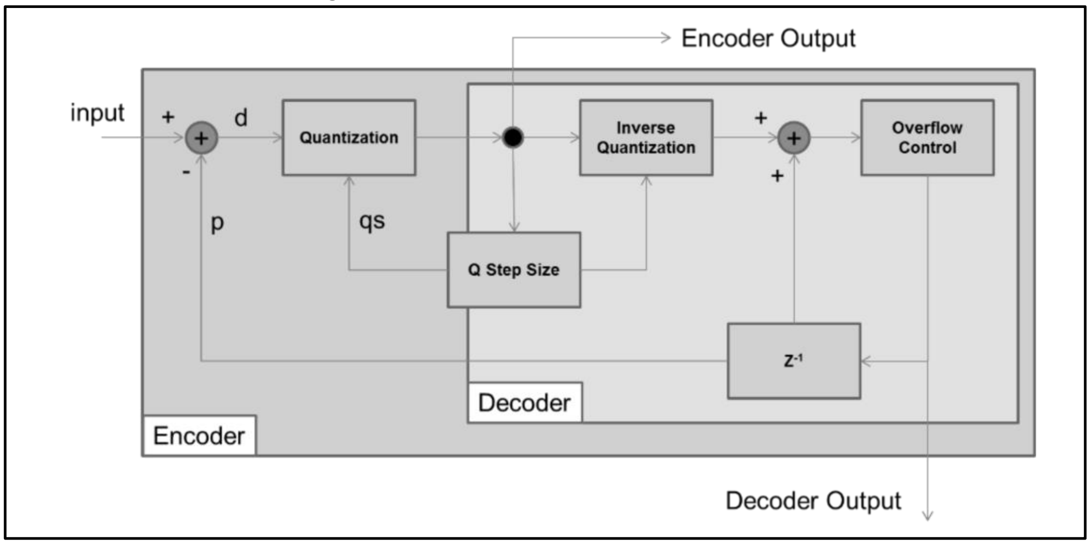
3.2.1 ADPCM compression

The ITU-T G.726 adaptive differential pulse code modulation (ADPCM) standard is applied to save bandwidth. This audio algorithm for lossy waveform coding predicts the current signal value from previous values, and transmits the difference between the real and the predicted value, quantized with an adaptive quantization step.

The ADPCM algorithm used by osxBlueVoice application compresses digital voice signals encoded as:

- Audio format: PCM
- Audio sample size: 16 bits
- Channels: 1 (mono)
- Audio sample rate: 16-8 kHz

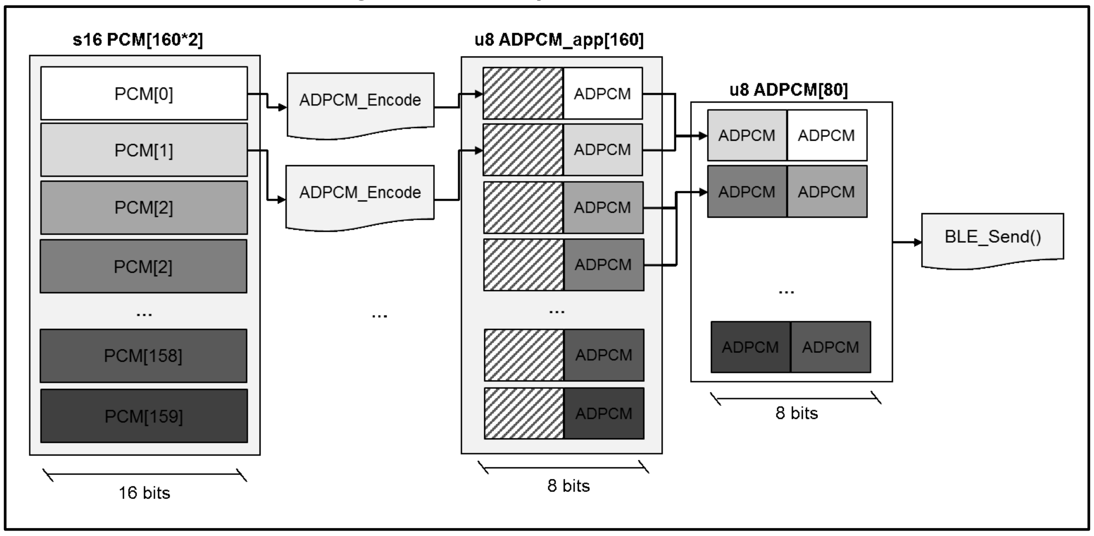
Figure 5: ADPCM encode-decode schema



osxBlueVoice implements a modified version of the compression algorithm with improved communication robustness through an additional low data rate channel with collateral information is added to the ADPCM quantized values, slightly increasing the overall bit-rate to an average 64.3 Kbps.

The internal buffering required by ADPCM compression is shown in *Figure 6: "ADPCM packet mechanism"*. 16-bit input PCM samples are encoded in 8-bit temporary samples with 4-bit actual data (u8 ADPCM_app buffer) and then encapsulated in 8-bit samples containing information of two PCM samples (u8 ADPCM buffer).

Figure 6: ADPCM packet mechanism



ADCPM decoding on the Rx side is performed in a symmetric fashion.

3.3 Packetization

The Tx data rate for streaming data is obtained from:

- the connection interval
- the number of packets per connection interval and user data payload for each packet

Processing in the osxBlueVoice library implements:

- a constant bitrate allocated to audio data through the chosen ADPCM compression
- a small connection interval to minimize the overall audio latency

More specifically, osxBlueVoice is implemented with a maximum packet payload of 20 bytes, while the suggested connection interval is10 ms.

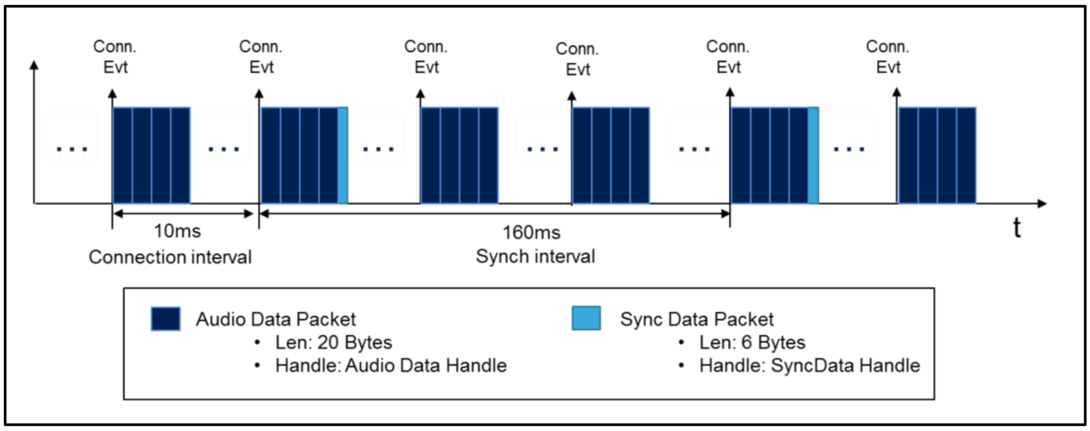
According to the audio sampling frequency chosen:

- With 16 kHz
  - a target of 64 kbps constant data rate is achieved by sending 80 bytes of ADPCM data (640 bits) at each connection event
  - audio data is packetized in 4 packets of 20 bytes per average connection event
- With 8 kHz
  - a target of 32 kbps constant data rate is achieved by sending 40 bytes of ADPCM data (320 bits) at each connection event
  - audio data is packetized in 2 packets of 20 bytes per average connection event

In addition, collateral ADPCM information is sent at a lower frequency via an additional smaller packet sent at regular intervals.

The following figure shows a 16 kHz compressed audio streaming example, where four data packets of 20 bytes are sent at each connection interval of 10 ms, while ADPCM collateral information is sent as an additional packet once every 160 ms.

Figure 7: BLE packets



4 osxBlueVoice library software description

4.1 Overview

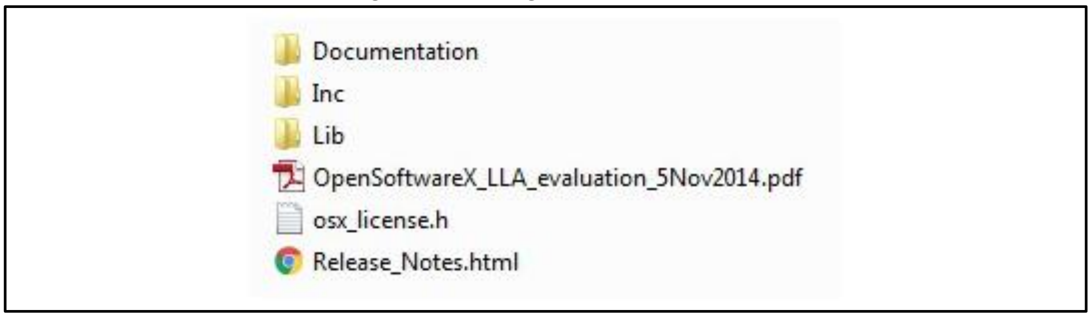
OsxBlueVoice implements a vendor-specific profile that expands on the standard Bluetooth low energy profiles already supported by BlueNRG. The osxBlueVoice engine is provided as a node-locked library which allows derivative firmware images to run on a specific STM32 device only. Licensing activation codes must to be requested from ST and included in the project (and will become part of the build process) prior to attempting its usage. The resulting firmware binary image will therefore be node-locked.

The library is implemented as middleware ready to be integrated in projects based on STM32Cube and the Bluetooth low energy BlueNRG network module. Specifically, osxBlueVoice requires the STM32 Bluetooth low energy middleware component included in the X-CUBE-BLE1 software expansion package.

4.2 Folder structure

The folder structure of the package STM32_OSX_BlueVoice_Library in the Middlewares folder is shown below.

Figure 8: Package folder structure



Documentation
Inc
Lib
OpenSoftwareX_LLA_evaluation_5Nov2014.pdf
osx_license.h
Release_Notes.html

The following files and folders are included in the software package:

- **Documentation**: with a compiled HTML file generated from the source code detailing the software components and APIs.
- **Inc**: contains the header file of the osxBlueVoice library
- **Lib**: contains the osxBlueVoice library binary code compiled for the three different tool-chains
- **OpenSoftwareX_LLA**: limited license agreement document
- **osx_license.h**: the license number received from a license request must be pasted into this file

4.3 Using the osxBlueVoice library

A typical flow of osxBlueVoice library operation can be seen by analyzing the BLUEVOICELINK1 sample firmware.

4.3.1 Initialization and configuration

First call the `osx_BlueVoice_Initialize` API to check if the license number copied in the osx_license.h file is correct. If the return value is 0, the library is unlocked and ready to use.

Afterwards the library must be configured according to the audio acquisition implemented in the application.

```
OSX_BLUEVOICE_Config_t OSX_BLUEVOICE_Config;

OSX_BLUEVOICE_Config.sampling_frequency = FR_16000;

OSX_BLUEVOICE_Config.channel_in = 1;

OSX_BLUEVOICE_Config.channel_tot = 2;

osx_BlueVoice_SetConfig(&OSX_BLUEVOICE_Config);
```

As shown above, a configuration structure must be filled by setting the audio sampling frequency (FR_16000 or FR_8000), the total number of channels given as the input to the library, and which channel (between 1 and channel_tot) shall be used for the voice streaming over BLE. If the return value is OSX_BV_SUCCESS, the library is configured correctly.

The osxBlueVoice can be reset by recalling `osx_BlueVoice_SetConfig`.

The library includes three callbacks that must be called when the corresponding BlueNRG event occurs:

- `osx_BlueVoice_ConnectionComplete_CB` must be called when an EVT_LE_CONN_COMPLETE event occurs; it sets the connection handle.
- `osx_BlueVoice_DisconnectionComplete_CB` must be called when an EVT_DISCONN_COMPLETE event occurs; it resets internal parameters.
- `osx_BlueVoice_AttributeModified_CB` must be called when an EVT_BLUE_GATT_ATTRIBUTE_MODIFIED event occurs; needed when an enable notification is received, it sets the working mode accordingly.

A timeout has been included for correct management of the BlueVoice profile status. When the module is no longer streaming or receiving, the profile status switches to OSX_BLUEVOICE_STATUS_READY as soon as the timer expires. The duration (in ms) of this timeout is defined by the OSX_BLUEVOICE_TIMEOUT_STATUS. To increase this timer, call the `osx_BlueVoice_IncTick` every 1 ms from the SysTick_Handler.

4.3.2 Working mode setting

The library working mode can be configured as NOT_READY (initial setting), TRANSMITTER, RECEIVER or HALF-DUPLEX.

The service and characteristics for the BlueVoice profile can be created by calling the `osx_BlueVoice_AddService` and `osx_BlueVoice_AddChar` functions. Both the APIs require the UUIDs (chosen by the user) as parameters, and they return the relevant handlers. The handle of the service must also be passed to the `osx_BlueVoice_AddChar` API.

Alternately, BlueVoice characteristics can be added to a pre-existing service created in your own application by calling `osx_BlueVoice_AddChar` and passing the handle of that particular service as a parameter.

If both the functions return OSX_BV_SUCCESS, the library is set to TRANSMITTER mode and can stream audio over BLE.

You can also create the characteristics out of the library and pass the relevant handles using a structure of the type `OSX_BLUEVOICE_ProfileHandle_t` to the `osx_BlueVoice_SetTxHandle` API. In the latter case, the following characteristics must be created:

```
aci_gatt_add_char(ServiceHandle,
              UUID_TYPE_128, CharAudioUUID, 20,
              CHAR_PROP_NOTIFY, ATTR_PERMISSION_NONE,
              GATT_DONT_NOTIFY_EVENTS, 16, 1,
              CharAudioHandle);

aci_gatt_add_char(ServiceHandle,
              UUID_TYPE_128, CharAudioSyncUUID, 6,
              CHAR_PROP_NOTIFY, ATTR_PERMISSION_NONE,
              GATT_DONT_NOTIFY_EVENTS, 16, 1,
              CharAudioSyncHandle);
```

The first relates to the compressed audio data; the second sends collateral information used to implement a synchronization mechanism.

You can choose to not create the BlueVoice Service and the module will not be able to transmit audio. This node can still function as a RECEIVER, however, if the connected module exports the BlueVoice profile; you must enable notifications on the other node by calling the `osx_BlueVoice_EnableNotification` function and setting the handle of the BlueVoice service and characteristics exported by the transmitter module through the `osx_BlueVoice_SetRxHandle` API.

If both the TRANSMITTER and RECEIVER procedures are performed, the module acts as transmitter and receiver, and the working mode is set to HALF-DUPLEX, creating a half-duplex link over BLE.

4.3.3 Audio signal injection

The osxBlueVoice library receives audio PCM input samples. The `osx_BlueVoice_AudioIn` function accepts parameters from a PCM buffer, containing all the acquired audio channels (according to the previous library configuration), and the number of PCM samples (for each channel) given as input. An amount of data equal to 1, 2, 5 or 10 ms is accepted, otherwise an OSX_BV_PCM_SAMPLES_ERR is returned.

The library compresses received PCM input samples; when 10 ms of audio is compressed, the `osx_BlueVoice_AudioIn` API returns OSX_BV_OUT_BUF_READY.

4.3.4 Compressed audio streaming

Regarding the transmitter part, the osxBlueVoice library gathers compressed data in an internal double buffer. For every 10 ms of audio, the `osx_BlueVoice_AudioIn` function returns OSX_BV_OUT_BUF_READY to signal that output data can be send via BLE by calling the `osx_BlueVoice_SendData` API.

If the audio sampling frequency is set to 8 kHz, two packets of 20 bytes are sent every 10 ms (according to the connection interval); if the frequency is 16 kHz, four packets of 20 bytes are sent.

On the receiver side, compressed audio is received from a notification through a EVT_BLUE_GATT_NOTIFICATION event and passed to the `osx_BlueVoice_ParseData` function that decompresses the data and returns a PCM buffer. This API is used to parse both audio and collateral information data.

4.4 APIs

Detailed user-API function and parameter descriptions are available in a compiled HTML file in the software package Documentation folder.

4.5 Supported tool-chains and compilers

osxBlueVoice library supports the three following environments:

- IAR Embedded Workbench for ARM® (EWARM) toolchain v7.70 + ST-LINK
- RealView Microcontroller Development Kit (MDK-ARM) toolchain v5.17 + ST-LINK
- Ac6 System Workbench for STM32 toolchain v1.9 + ST-LINK

4.6 Sample application

osxBlueVoice is part of the BLUEVOICELINK1 sample application based on the X-CUBE-BLE1 and X-CUBE-MEMSMIC1 software expansion packages for STM32Cube. BLUEVOICELINK1 is a complete sample application that exploits all the osxBlueVoice functions to demonstrate point-to-point half-duplex voice communication over BLE.

The osxBlueVoice library is also used in the BLUEMICROSYSTEM2 sample application to read and display real-time sensor data, stream audio and run the sensor fusion algorithm.

The BLUEVOICE1 and BLUEMICROSYSTEM2 firmware and related documentation are available on *www.st.com*

5 References

1. MP34DT01-M MEMS audio sensor omnidirectional digital microphone, datasheet. *http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/DM00121815.pdf*
2. PDM audio software decoding on STM32 microcontrollers, Application note AN3998 *http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/DM00040808.pdf*
3. Bluetooth Core Specification 4.1, *https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=282159*
4. Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking: Townsend, K., Cufí, C., Akiba, Davidson, R., O'Reilly Media, 2014.

6 Revision history

Table 4: Document revision history

Date	Version	Changes
06-Aug-2015	1	Initial release.
06-Sep-2016	2	Text and formatting changes throughout document Updated *Section "Introduction"* Updated *Section 2.1: "Overview"* Updated *Section 3.1.4: "BlueVoice service"* Updated *Section 3.2: "Audio processing"* Removed Section: "Application development notes" Added *Section 4: "osxBlueVoice library software description"* Updated *Section 5: "References"*