IoT SDK Bluetooth LE Audio Development Guide

Version:	0.1
Release date:	2022-08-26

Version History

Version	Date	Description
0.1	2022-08-26	Initial Version

Table of Contents

List of Figures

List of Tables

1 Overview

MT793X development platform provides Bluetooth and Bluetooth Low Energy (LE) connectivity support for IoT devices. Bluetooth standard offers basic rate (BR) or enhanced data rate (EDR) and Bluetooth Low Energy (LE) support. Devices that can support BR/EDR and Bluetooth LE are referred to as dual-mode devices. Typically, in a Bluetooth system, a mobile phone or laptop computer acts as a dual-mode device. Devices that only support Bluetooth LE are referred to as single-mode devices where the low power consumption is the primary concern for application development, such as those that run on coin cell batteries.

This document guides you through:

- How to implement your application for Bluetooth music.
- Architecture of Bluetooth music system and folders.
- Introduction of each module related to Bluetooth music.

1.1 Software Architecture

The 793X SDK is built up by four layers, APP, MW, Driver and Hardware. Software developers can just focus on the three modules, BT APP, bt_le_audio and sink.

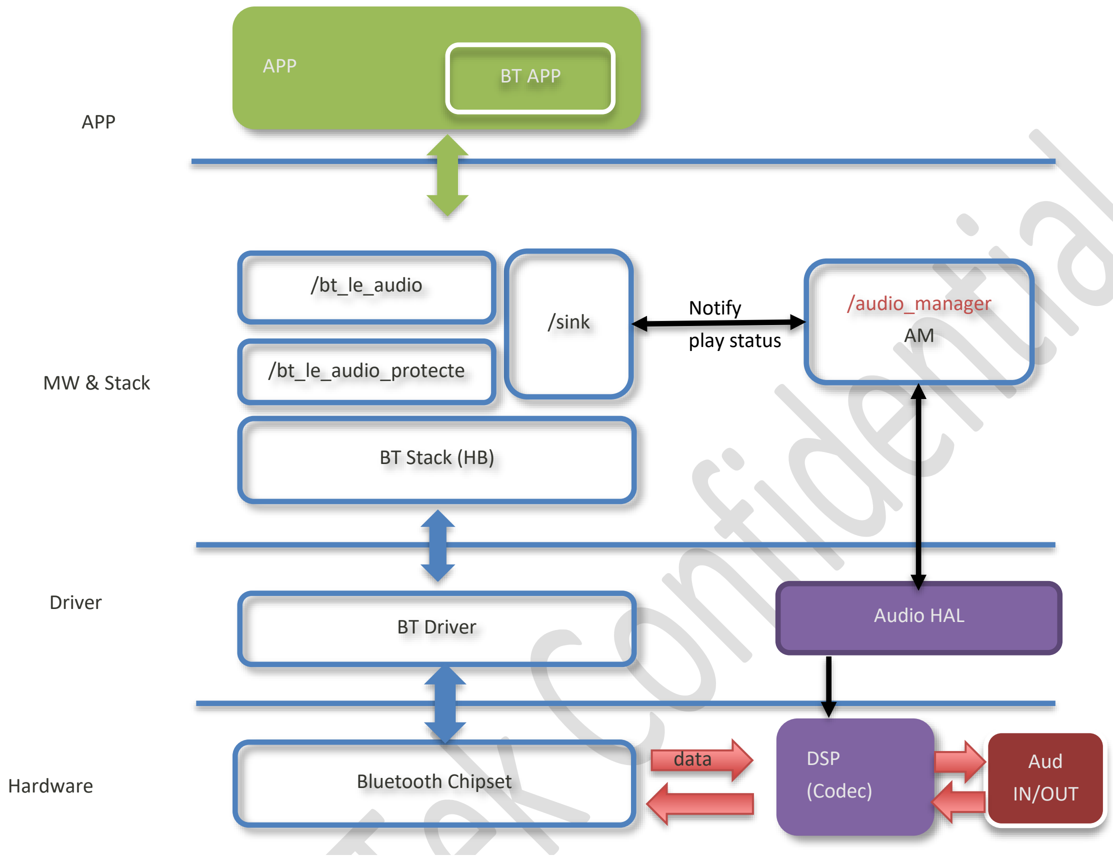


Figure 1 Software Architecture

1.2 Folder Structure

The folders listed below are for your reference. For the detailed design of MediaTek IoT SDK application, refer to the source code.

- APPs
  /project/<board>/apps/<project>/src/apps/app_*
- Configs
  /project/<board>/apps/<project>/src/apps/config
- Event sender
  /project/<board>/apps/<project>/src/apps/events
- Utils
  /project/<board>/apps/<project>/src/apps/utils

2 Module Introduction

As Figure 2 shows, the application layer of MediaTek is composed of three kinds of modules: event senders, APPs, and Config. Event senders send UI shell events to APPs. APPs receive the events and take actions to complete the applications. Configs manage provide key actions and project configuration.

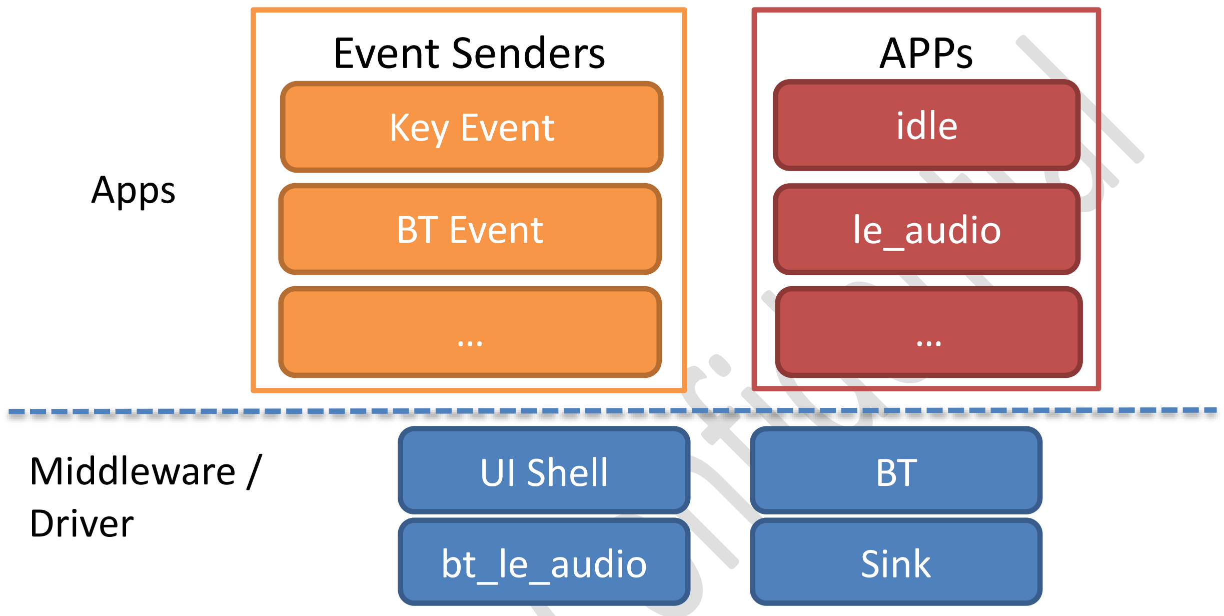


Figure 2 System Architecture

Event senders control key events, interaction events, and others. These modules register callbacks in middleware or hardware abstraction layer modules. When the callbacks are executed, the events are sent to APPs.

Take BT events as an example:

- The event sender sends an event to UI shell when called. After receiving the event sent by the key event sender, the UI shell sends it to APPs.

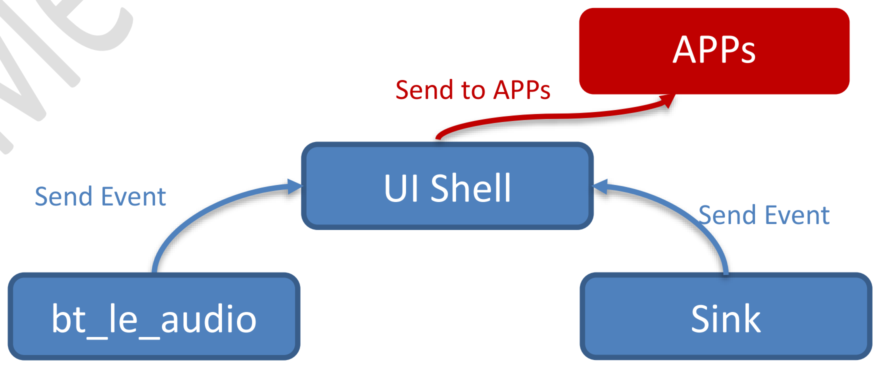


Figure 3 BT Event Sender

Multiple folders named “app_***” are APPs, which implement UI logic. An APP can send events to another one for the request of taking some actions or notifying the happening of some events.

2.1 Event Senders

All the events sent to the APPs layer would be sent via the event sender. Event senders include two modules: BT event and key event. These two modules are responsible for sending different events.

2.1.1 BT Event

This module sends three groups of events. The EVENT_GROUP_UI_SHELL_BT_SINK group is frequently used by APPs to get information about BT. The EVENT_GROUP_UI_SHELL_BT_CONN_MANAGER group and the EVENT_GROUP_UI_SHELL_BT group are helpful when the EVENT_GROUP_UI_SHELL_BT_SINK group cannot provide enough information.

- EVENT_GROUP_UI_SHELL_BT_SINK group events – The event IDs of the group are defined in sink module in middleware. The BT state change and profile connection change are notified by these events.
- EVENT_GROUP_UI_SHELL_BT group events – The event IDs of the group are defined in Bluetooth module in middleware. The event sender callback is registered where the APP needs the BT messages.

2.1.2 Key Event

This module sends EVENT_GROUP_UI_SHELL_KEY group events. This group is used for defining keys. For example, the ID of the power on key is KEY_POWER_ON. Please refer to /project/<board>/apps/<project>/inc/apps/config/apps_config_event_list.h.

2.2 APPs

APPs implement the actions of the application layer and are organized by features and functions. Each APP is composed of one or more activities. The activities are managed by UI shell. Every activity receives events of the EVENT_GROUP_UI_SHELL_SYSTEM group to follow the management to create, destroy, resume, pause, refresh and result.

The following sections introduce the basic APPs in MediaTek IoT SDK. You can refer to the design to add new APPs.

2.2.1 Home Screen APP

Home Screen APP works as settings. The app_home_screen_idle_activity is the only activity of Home Screen APP, which is the top priority when the system is in idle state.

- Home Screen APP processes the common key event requests, such as power_on, power_off.

2.2.2 Music APP

Music APP controls the playback of music. This APP is composed of two activities: app_music_idle_activity and app_music_activity. The idle activity is created when the device is powered on, and the music activity will be started during streaming.

App_music_idle_activity processes events of groups EVENT_GROUP_UI_SHELL_KEY and EVENT_GROUP_UI_SHELL_BT_SINK.

- EVENT_GROUP_UI_SHELL_KEY
  - ■ The key event handled by app_music_idle_activity is shown in Table 1. States and keys are handled by app_music.
- EVENT_GROUP_UI_SHELL_BT_SINK
  - ■ When a BT_SINK_SRV_EVENT_STATE_CHANGE event is received, the APP checks whether the state is changed to streaming and starts app_music_activity.

App_music_activity processes events of groups EVENT_GROUP_UI_SHELL_KEY, EVENT_GROUP_UI_SHELL_BT_SINK and EVENT_GROUP_UI_SHELL_APP_INTERACTION.

- EVENT_GROUP_UI_SHELL_KEY
  - ■ The key event handled by app_music_activity is shown in Table 1. States and keys are handled by app_music.
- EVENT_GROUP_UI_SHELL_BT_SINK
  - ■ When a BT_SINK_SRV_EVENT_STATE_CHANGE event is received, the APP checks whether the streaming is ended and finishes itself.
- EVENT_GROUP_UI_SHELL_APP_INTERACTION
  - ■ When an APPS_EVENTS_INTERACTION_UPDATE_MMI_STATE event is received, the APP sets MMI state.

Table 1 States and Keys Handled by app_music

Activity	State	Key handle
Music idle activity	Power on Connected	Play
Music activity	Streaming	Pause Next / Forward Volume Up / Down

2.2.3 LE Audio APP

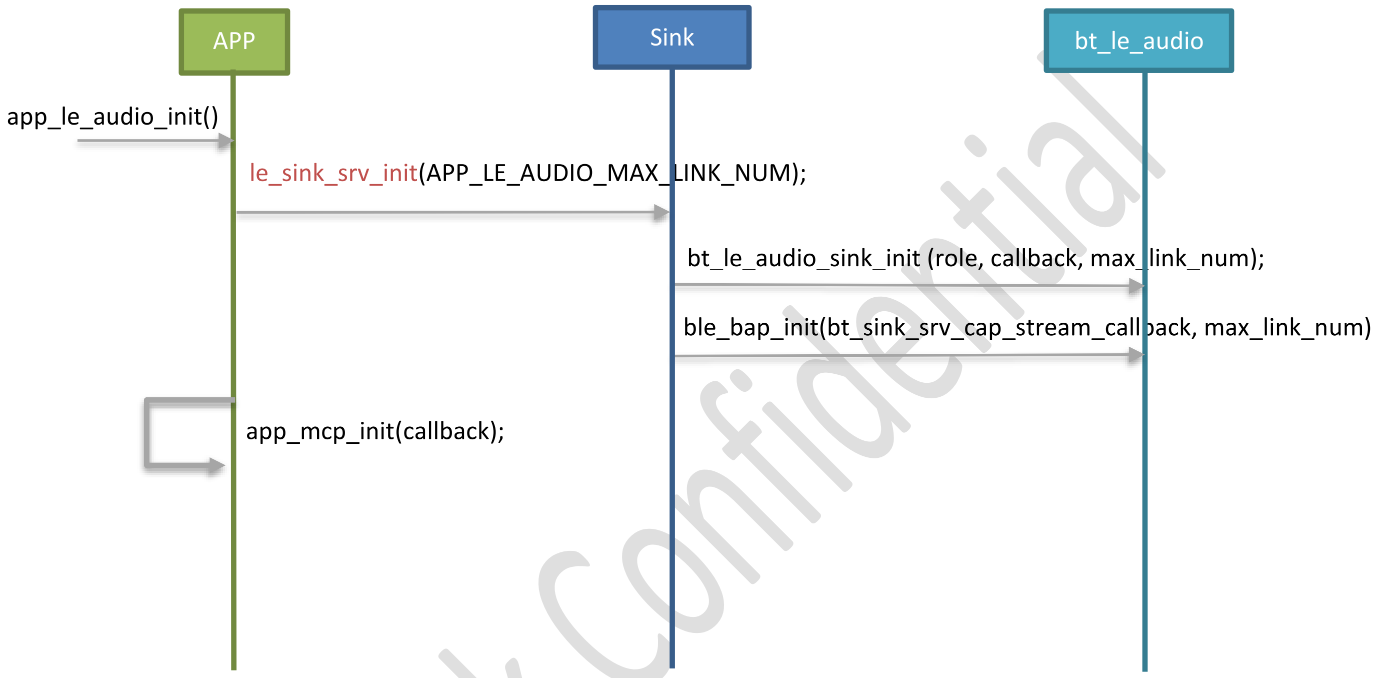
LE Audio APP controls the common key event of LE connection such as start/stop advertising, disconnect, unpair device etc. It also includes the advertising data for LE Audio.

2.2.4 Media Control Profile Client

It is necessary for a media receiver to support Media Control Profile in acting as client. The GATT service registration procedure of MCP and corresponding callback function is declared in projects/mt7933_hdk/apps/bga_sdk_audio/apps/app_le_audio/app_mcp.c. For detail of MCP, please check MCP specification from Bluetooth SIG for reference.

2.2.5 Initialization

For initializing le audio, APP needs to call app_le_audio_init function to enable starting the initialization procedure. The APP would automatically initialize sink module, and then the sink module would initialize the bt_le_audio module.
After that, the APP would also register callback for MCP.


APP
Sink
bt_le_audio
app_le_audio_init()
le_sink_srv_init(APP_LE_AUDIO_MAX_LINK_NUM);
bt_le_audio_sink_init (role, callback, max_link_num);
ble_bap_init(bt_sink_srv_cap_stream_callback, max_link_num)
app_mcp_init(callback);

2.2.5.1 Register GATT Services

Register required GATT Servers in APP layer to provide service for LE Audio Client side.
For example, to allow the device to act as UMR role, certain services are required, for example, ASCS for ASE control, PACS for providing published audio capability, VCS for providing volume control.
For detail of GATT services of UMR, please see task 2.4.2 for reference.

Below is the sample code to register service in APP layer:

```
const bt_gatts_service_t *bt_if_clm_gatt_server[] = {
               :
    &ble_ascs_service,   /**< handle range: 0x1103 to 0x110F. */
    &ble_pacs_service,   /**< handle range: 0x1200 to 0x1212. */
    &ble_vcs_service,    /**< handle range: 0x1301 to 0x1309. */
    &ble_vocs_service_channel_1, /**< handle range: 0x2001 to 0x200C. */
    &ble_aics_service,   /**< handle range: 0x4001 to 0x4010. */
    &ble_csis_service,   /**< handle range: 0x6001 to 0x600A. */
    &ble_cas_service,    /**< handle range: 0x7001 to 0x7002. */
    &ble_tmas_service,   /**< handle range: 0xA301 to 0xA303. */
    NULL
};
```

2.2.5.2 Set up Advertising for Sink Server Role

Before starting advertising, you need to provide advertising data and configure it to controller. You need to provide memory for recording advertising data to Bluetooth Host. The memory can be freed after config function is returned.

```
static int32_t _app_le_audio_get_adv_data(multi_ble_adv_info_t *adv_data)
{
    //LE_AUDIO_MSGLOG_I(LOG_TAG"[APP] app_le_audio_get_adv_data", 0);

    /* ADV DATA */
    if ((NULL != adv_data->adv_data) && (NULL != adv_data->adv_data->data)) {
        uint16_t sink_conent, source_conent;
        uint8_t rsi[6];
        uint8_t len = 0;

        if (adv_data->adv_data->data_length < APP_LE_AUDIO_ADV_DATA_LEN_MAX) {
            APPS_LOG_MSGID_E(APP_LOG_TAG", ERR: pls check adv data len %d < %d", 2,
                             adv_data->adv_data->data_length, APP_LE_AUDIO_ADV_DATA_LEN_MAX);
            return -1;
        }

        adv_data->adv_data->data[len] = 2;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_FLAG;
        adv_data->adv_data->data[len + 2] = BT_GAP_LE_AD_FLAG_BR_EDR_NOT_SUPPORTED |
                                            BT_GAP_LE_AD_FLAG_GENERAL_DISCOVERABLE;
        len += 3;

        /* adv_data: RSI (Resolvable Set Identifier) */
        adv_data->adv_data->data[len] = 7;
        adv_data->adv_data->data[len + 1] = 0x2E;
        ble_csis_get_rsi(rsi);
        memcpy(&adv_data->adv_data->data[len + 2], rsi, sizeof(rsi));
        len += 8;

       /* adv_data: AD_TYPE_SERVICE_DATA (BAP)*/
        adv_data->adv_data->data[len] = 9;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_SERVICE_DATA;
       /* ASCS UUID: 2 bytes */
        adv_data->adv_data->data[len + 2] = (BT_GATT_UUID16_ASCS_SERVICE & 0x00FF);
        adv_data->adv_data->data[len + 3] = ((BT_GATT_UUID16_ASCS_SERVICE & 0xFF00) >> 8);
       /* Announcement Type: 1 bytes */
        adv_data->adv_data->data[len + 4] = ANNOUNCEMENT_TYPE_GENERAL;
       /* Available Audio Contect: 4 bytes */
        ble_pacs_get_available_audio_contexts(&sink_conent, &source_conent);
        memcpy(&adv_data->adv_data->data[len + 5], &sink_conent, 2);
        memcpy(&adv_data->adv_data->data[len + 7], &source_conent, 2);
        adv_data->adv_data->data[len + 9] = 0x00; /* Length of the Metadata field = 0 */
        len += 10;
        /* adv_data: TX_POWER (BAP)*/
        adv_data->adv_data->data[len] = 2;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_TX_POWER;
        adv_data->adv_data->data[len + 2] = 0x7F;
        len += 3;

        /* adv_data: AD_TYPE_APPEARANCE (TMAP) */
        adv_data->adv_data->data[len] = 3;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_APPEARANCE;
        /* value: 2 bytes (EX: 0x0941 = earbud, 0x0841 = audio sink*/
        adv_data->adv_data->data[len + 2] = 0x41;
        adv_data->adv_data->data[len + 3] = 0x08;
        len += 4;

        /* adv_data: AD_TYPE_SERVICE_DATA (TMAS)*/
        adv_data->adv_data->data[len] = 5;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_SERVICE_DATA;
        /* TMAS UUID: 2 bytes */
        adv_data->adv_data->data[len + 2] = (BT_SIG_UUID16_TMAS & 0x00FF);
        adv_data->adv_data->data[len + 3] = ((BT_SIG_UUID16_TMAS & 0xFF00) >> 8);
        /* TMAS Data: 2 bytes */
        ble_tmap_role_t role = ble_tmas_get_role();
        adv_data->adv_data->data[len + 4] = (role & 0x00FF);
        adv_data->adv_data->data[len + 5] = ((role & 0xFF00) >> 8);
        len += 6;

        /* adv_data: AD_TYPE_SERVICE_DATA (CAS)*/
        adv_data->adv_data->data[len] = 4;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_SERVICE_DATA;
        /* CAS UUID: 2 bytes */
        adv_data->adv_data->data[len + 2] = (BT_SIG_UUID16_CAS & 0x00FF);
        adv_data->adv_data->data[len + 3] = ((BT_SIG_UUID16_CAS & 0xFF00) >> 8);
        adv_data->adv_data->data[len + 4] = ANNOUNCEMENT_TYPE_GENERAL;
        len += 5;

        uint16_t device_name_len = 0;
        char device_name[BT_GAP_LE_MAX_DEVICE_NAME_LENGTH] = {0};
        bt_bd_addr_t *local_addr = bt_device_manager_get_local_address();

        snprintf(device_name, BT_GAP_LE_MAX_DEVICE_NAME_LENGTH, "MTK_LEA_%.2X%.2X%.2X%.2X%.2X%.2X",
                 (*local_addr)[5], (*local_addr)[4], (*local_addr)[3],
                 (*local_addr)[2], (*local_addr)[1], (*local_addr)[0]);

        device_name_len = strlen((char *)device_name);

        /* scan_rsp: AD_TYPE_NAME_COMPLETE*/
        adv_data->adv_data->data[len] = device_name_len + 1;
        adv_data->adv_data->data[len + 1] = BT_GAP_LE_AD_TYPE_NAME_COMPLETE;
        memcpy(&adv_data->adv_data->data[len + 2], device_name, device_name_len);

        len += 2 + device_name_len;

        if (len > APP_LE_AUDIO_ADV_DATA_LEN_MAX) {
            //serious error
            print_error("ERR: pls check code, data may overlapped len %d > %d", 2,
                              len, APP_LE_AUDIO_ADV_DATA_LEN_MAX);
            return -2;
        }
        adv_data->adv_data->data_length = len;
    }

    if (NULL != adv_data->adv_param) {
        //AE -- shall not be both connectable and scannable
        adv_data->adv_param->advertising_event_properties =
                                          BT_HCI_ADV_EVT_PROPERTIES_MASK_CONNECTABLE;

        /* Interval should be no larger than 100ms when discoverable */
        adv_data->adv_param->primary_advertising_interval_min = g_le_audio_adv_interval_min;
        adv_data->adv_param->primary_advertising_interval_max = g_le_audio_adv_interval_max;
        adv_data->adv_param->primary_advertising_channel_map = 0x07;
        adv_data->adv_param->own_address_type = BT_ADDR_RANDOM;
        adv_data->adv_param->advertising_filter_policy = 0;
        adv_data->adv_param->advertising_tx_power = 0x7F;
        adv_data->adv_param->primary_advertising_phy = BT_HCI_LE_ADV_PHY_1M;
        adv_data->adv_param->secondary_advertising_phy = BT_HCI_LE_ADV_PHY_1M;
    }
    return 0;
}

bt_status_t _app_le_audio_config_adv_data(uint8_t adv_hdl)
{
    uint8_t adv_raw[APP_LE_AUDIO_ADV_DATA_LEN_MAX] = {0};
    uint8_t scan_raw[APP_LE_AUDIO_ADV_SCAN_RSP_LEN_MAX] = {0};

    bt_hci_le_set_ext_advertising_parameters_t ext_adv_para = {0};

    bt_gap_le_set_ext_advertising_data_t ext_data = {0};
    ext_data.data_length = sizeof(adv_raw);
    ext_data.data = adv_raw;
    ext_data.fragment_preference = 0;

    bt_gap_le_set_ext_scan_response_data_t ext_scan = {0};
    ext_scan.data_length = sizeof(scan_raw);
    ext_scan.data = scan_raw;
    ext_scan.fragment_preference = 0;

    multi_ble_adv_info_t adv_info;
    adv_info.adv_param = &ext_adv_para;
    adv_info.adv_data = &ext_data;
    adv_info.scan_rsp = &ext_scan;

    if (_app_le_audio_get_adv_data(&adv_info) != 0) {
        return BT_STATUS_FAIL;
    }

    return bt_gap_le_config_extended_advertising(adv_hdl, bt_gap_le_get_random_address(),
                                             &ext_adv_para, &ext_data, &ext_scan);
}
```

2.3 Project Config

Settings are in /project/<board>/apps/<project>/inc/project_config.h

2.3.1 BT_LE_CONNECTION_MAX

This variable is used for indicating the maximum number of LE connections. This variable would also affect BT_LE_CONNECTION_BUF_SIZE. Both are related to LE connection limitation.

2.3.2 BT_CONNECTION_MAX

This variable is used for indicating the maximum number of BR/EDR connections. This variable would also affect BT_CONNECTION_BUF_SIZE. Both are related to BR/EDR connection limitation.

2.4 Middleware

2.4.1 Sink Service

Integrated with audio manager (AM) and LE audio MW, the application layer can send action to sink service and then wait for the corresponding event from sink service. That is the interface between sink service and application layer.

2.4.1.1 Initialization

As mentioned in 2.2.5, the application needs to call le_sink_srv_init() function to initialize sink service. This function will call initialization function of bt_le_audio middleware.

2.4.1.2 Action and Event

The interface of sink service is Action and Event. Below is a list of the actions and events.

Action	Corresponding Event	Description
BT_SINK_SRV_ACTION_PLAY	BT_SINK_SRV_EVENT_LE_MCP_STATUS_CHANGE	Play music
BT_SINK_SRV_ACTION_PAUSE	BT_SINK_SRV_EVENT_LE_MCP_STATUS_CHANGE	Pause music
BT_SINK_SRV_ACTION_NEXT_TRACK	BT_SINK_SRV_EVENT_LE_MCP_EVENT_IND	Next track
BT_SINK_SRV_ACTION_PREV_TRACK	BT_SINK_SRV_EVENT_LE_MCP_EVENT_IND	Previous Track
BT_SINK_SRV_ACTION_PLAY_PAUSE	BT_SINK_SRV_EVENT_LE_MCP_STATUS_CHANGE	Toggle play/pause state
BT_SINK_SRV_ACTION_FAST_FORWARD	N/A	Fast Forward
BT_SINK_SRV_ACTION_REWIND	N/A	Rewind
BT_SINK_SRV_ACTION_GET_PLAY_STATUS	BT_SINK_SRV_EVENT_LE_MCP_STATUS_CHANGE	Complete event of get play status
BT_SINK_SRV_ACTION_GET_CAPABILITY	BT_SINK_SRV_EVENT_LE_MCP_EVENT_IND	Complete event of get capability
N/A	BT_SINK_SRV_EVENT_LE_VCP_VOLUME_STATE_CHANGE	Event that notifies that the volume setting has been changed by remote device

2.4.1.3 Action

Action is a function supported by the sink service. The application can send an action request to the sink service by using `bt_sink_srv_send_action`. For example, if you want to play music, you can send the action BT_SINK_SRV_ACTION_PLAY to the sink service.

```
{
    bt_sink_srv_action_t sink_action = BT_SINK_SRV_ACTION_NONE;
    bt_status_t bt_status;

    sink_action = BT_SINK_SRV_ACTION_PLAY;
    bt_status = le_sink_srv_send_action (sink_action, op);
    if (bt_status != BT_STATUS_SUCCESS)
        // print error message
}
```

2.4.1.4 Event

2.4.1.4.1 LE_SINK_SRV_EVENT_STATE_CHANGE

This event indicates the state of the sink service.

```
typedef struct {
    bt_sink_srv_state_t previous;
    bt_sink_srv_state_t current;
} bt_sink_srv_state_change_t;
```

Below is the definition of bt_sink_srv_state_t.

```
#define BT_SINK_SRV_STATE_NONE                  (0x0000)
#define BT_SINK_SRV_STATE_POWER_ON              (0x0001)
#define BT_SINK_SRV_STATE_CONNECTED             (0x0002)
#define BT_SINK_SRV_STATE_STREAMING             (0x0004)
```

2.4.1.4.2 LE_SINK_SRV_EVENT_REMOTE_INFO_UPDATE

This event is used for notifying APP layer of the connection state of remote device. The data structure of this event is as below.

```
typedef struct {
    bt_addr_t address;
    bt_le_link_state_t pre_state;
    bt_le_link_state_t state;
    bt_le_service_mask_t pre_connected_service;
    bt_le_service_mask_t connected_service;
    bt_status_t reason;
} bt_le_sink_srv_event_remote_info_update_t;
```

You can check state element to know the current connection state.

```
#define BT_BLE_LINK_DISCONNECTED            (0x00)
#define BT_BLE_LINK_DISCONNECTING           (0x01)
#define BT_BLE_LINK_CONNECTING              (0x02)
#define BT_BLE_LINK_CONNECTED               (0x03)
typedef uint8_t bt_le_link_state_t;
```

You can check connected service to know which service is connected.

```
#define BT_SINK_SRV_PROFILE_NONE            (0x00)
#define BT_SINK_SRV_PROFILE_CALL            (0x01)
#define BT_SINK_SRV_PROFILE_MUSIC           (0x02)
typedef uint8_t bt_le_service_mask_t;
```

2.4.1.4.3 BT_SINK_SRV_EVENT_LE_MCP_STATUS_CHANGE

This event is used for notifying the application of the change of music status. The data structure of this event is bt_sink_srv_event_param_t. You need to pare the element named mcp_status_change, which includes the information on the Bluetooth address of remote device and the media state.

```
typedef union {
    .
    .
    bt_sink_srv_event_mcp_status_changed_ind_t mcp_status_change;
    .
    .
} bt_sink_srv_event_param_t;
```

```
typedef struct {
    bt_bd_addr_t   address;
    ble_mcs_media_state_t    mcp_status;
} bt_sink_srv_event_mcp_status_changed_ind_t;
```

```
#define BLE_MCS_MEDIA_STATE_INACTIVE                0x00
#define BLE_MCS_MEDIA_STATE_STOPED                  BLE_MCS_MEDIA_STATE_INACTIVE
#define BLE_MCS_MEDIA_STATE_PLAYING                 0x01
#define BLE_MCS_MEDIA_STATE_PAUSED                  0x02
#define BLE_MCS_MEDIA_STATE_SEEKING                 0x03
typedef uint8_t ble_mcs_media_state_t;
```

2.4.1.4.4 BT_SINK_SRV_EVENT_LE_MCP_EVENT_IND

This event is used for numerous types of events about MCP operation. There are numerous definitions of param. You need to parser param by using correct data structure chosen according to event type.

```
typedef struct {
    bt_msg_type_t event_type;
    bt_status_t   status;
    bool is_param_valid;
    bt_sink_srv_event_mcp_ind_param_t param;
} bt_sink_srv_event_mcp_ind_t;
```

Definition of event_type.

```
#define BLE_MCP_MEDIA_PLAYER_NAME_IND
#define BLE_MCP_TRACK_CHANGED_IND
#define BLE_MCP_TRACK_TITLE_IND
#define BLE_MCP_TRACK_DURATION_IND
#define BLE_MCP_TRACK_POSITION_IND
#define BLE_MCP_PLAYBACK_SPEED_IND
#define BLE_MCP_SEEKING_SPEED_IND
#define BLE_MCP_PLAYING_ORDER_IND
#define BLE_MCP_MEDIA_STATE_IND
#define BLE_MCP_MEDIA_CONTROL_POINT_IND
#define BLE_MCP_MEDIA_CONTROL_OPCODES_SUPPORTED_IND
#define BLE_MCP_READ_MEDIA_PLAYER_NAME_CNF
#define BLE_MCP_READ_MEDIA_PLAYER_ICON_URL_CNF
#define BLE_MCP_READ_TRACK_TITLE_CNF
#define BLE_MCP_READ_TRACK_DURATION_CNF
#define BLE_MCP_READ_TRACK_POSITION_CNF
#define BLE_MCP_READ_PLAYBACK_SPEED_CNF
#define BLE_MCP_READ_SEEKING_SPEED_CNF
#define BLE_MCP_READ_PLAYING_ORDER_CNF
#define BLE_MCP_READ_PLAYING_ORDERS_SUPPORTED_CNF
#define BLE_MCP_READ_MEDIA_STATE_CNF
#define BLE_MCP_READ_MEDIA_CONTROL_OPCODES_SUPPORTED_CNF
#define BLE_MCP_READ_CONTENT_CONTROL_ID_CNF
#define BLE_MCP_READ_CURRENT_TRACK_OBJECT_INFORMATION_CNF
#define BLE_MCP_SET_MEDIA_PLAYER_NAME_NOTIFICATION_CNF
#define BLE_MCP_SET_TRACK_CHANGED_NOTIFICATION_CNF
#define BLE_MCP_SET_TRACK_TITLE_NOTIFICATION_CNF
#define BLE_MCP_SET_TRACK_DURATION_NOTIFICATION_CNF
#define BLE_MCP_SET_TRACK_POSITION_NOTIFICATION_CNF
#define BLE_MCP_SET_PLAYBACK_SPEED_NOTIFICATION_CNF
#define BLE_MCP_SET_SEEKING_SPEED_NOTIFICATION_CNF
#define BLE_MCP_SET_PLAYING_ORDER_NOTIFICATION_CNF
```

```
#define BLE_MCP_SET_MEDIA_STATE_NOTIFICATION_CNF
#define BLE_MCP_SET_MEDIA_CONTROL_POINT_NOTIFICATION_CNF
#define BLE_MCP_SET_MEDIA_CONTROL_OPCODES_SUPPORTED_NOTIFICATION_CNF
#define BLE_MCP_PLAY_CURRENT_TRACK_CNF
#define BLE_MCP_PAUSE_CURRENT_TRACK_CNF
#define BLE_MCP_STOP_CURRENT_TRACK_CNF
#define BLE_MCP_MOVE_TO_NEXT_TRACK_CNF
#define BLE_MCP_MOVE_TO_PREVIOUS_TRACK_CNF
#define BLE_MCP_FAST_FORWARD_CNF
#define BLE_MCP_FAST_REWIND_CNF
#define BLE_MCP_WRITE_PLAYBACK_SPEED_CNF
#define BLE_MCP_WRITE_PLAYING_ORDER_CNF
#define BLE_MCP_MOVE_TO_FIRST_TRACK_CNF
#define BLE_MCP_MOVE_TO_LAST_TRACK_CNF
```

Definition of bt_sink_srv_event_mcp_ind_param_t.

```
typedef union {
    ble_mcp_event_parameter_t                   event_param;
    ble_mcp_read_media_player_name_cnf_t        player_name_cnf;
    ble_mcp_read_media_player_icon_url_cnf_t    player_icon_url_cnf;
    ble_mcp_read_track_title_cnf_t              track_title_cnf;
    ble_mcp_read_track_duration_cnf_t           track_dur_cnf;
    ble_mcp_read_track_position_cnf_t           track_pos_cnf;
    ble_mcp_read_playback_speed_cnf_t           playback_speed_cnf;
    ble_mcp_read_seeking_speed_cnf_t            seeking_speed_cnf;
    ble_mcp_read_playing_order_cnf_t            playing_order_cnf;
    ble_mcp_read_playing_orders_supported_cnf_t playing_order_sup_cnf;
    ble_mcp_read_media_state_cnf_t              media_state_cnf;
    ble_mcp_read_media_control_opcodes_supported_cnf_t  ctrl_op_sup_cnf;
    ble_mcp_read_content_control_id_cnf_t       content_ctrl_id_cnf;
    ble_mcp_media_player_name_ind_t             player_name_ind;
    ble_mcp_track_changed_ind_t                 track_changed_ind;
    ble_mcp_track_title_ind_t                   track_title_ind;
    ble_mcp_track_duration_ind_t                track_dur_ind;
    ble_mcp_track_position_ind_t                track_pos_ind;
    ble_mcp_playback_speed_ind_t                playback_speed_ind;
    ble_mcp_seeking_speed_ind_t                 seeking_speed_ind;
    ble_mcp_playing_order_ind_t                 playing_order_ind;
    ble_mcp_media_state_ind_t                   media_state_ind;
    ble_mcp_media_control_opcodes_supported_ind_t   ctrl_op_sup_ind;
    ble_mcp_media_control_point_ind_t           ctrl_point_ind;
} bt_sink_srv_event_mcp_ind_param_t;
```

2.4.1.4.5 BT_SINK_SRV_EVENT_LE_VCP_VOLUME_STATE_CHANGE

This event is used for notifying APP layer that the volume setting has been changed by remote device. The data structure of this event is as below.

```
typedef struct {
    ble_vcs_volume_t volume;
    ble_vcs_mute_t mute;
} ble_vcs_volume_state_t;
```

Range of volume is 0x00 to 0xFF.

```
#define BLE_VCS_VOLUME_MIN          0
#define BLE_VCS_VOLUME_MAX          0xFF
typedef uint8_t ble_vcs_volume_t;
```

The value of mute is 0 or 1.

```
#define BLE_VCS_UNMUTE      0
#define BLE_VCS_MUTE        1
typedef uint8_t ble_vcs_mute_t;
```

2.4.2 bt_le_audio

There are numerous GATT services defined for LE Audio. We declare those services in the bt_le_audio layer.

2.4.2.1 CAS

Common Audio Service (CAS) is declared in middleware/MTK/bt_le_audio/cap/ble_cas_service.c. For detail of CAS, please check CAS specification from Bluetooth SIG for reference.

2.4.2.2 PACS

Published Audio Capabilities Service (PACS) declared in
middleware/MTK/bt_le_audio/pacs/ble_pacs_service.c.
The codec specific capabilities (like sample rate, frame duration, audio channel counts etc.) are declared here. If you want to customize the codec specific capabilities, you can modify this file. For detail of PACS, please check ASCS specification from Bluetooth SIG for reference.

The PACS initial flow is in included in ble_bap_init(), so Sink MW or upper layer doesn’t need to take care of initialization.

You could set up Published Audio Capabilities Service (PACS) according to your requirement.
In be_le_audio MW, the PACS attribute should be checked.

In bt_le_audio MW (e.g. `bt_le_audio/src/ascs/ble_pacss_service.c`), the "void ble_pacs_init_parameter(void)" should be implemented for setting PAC data.

```
#define PACS_SINK_PAC_1_RECORD_NUM      0x02

static uint8_t g_pacs_codec_capabilities_16k[] = {
    CODEC_CAPABILITY_LEN_SUPPORTED_SAMPLING_FREQUENCY,
    CODEC_CAPABILITY_TYPE_SUPPORTED_SAMPLING_FREQUENCY,
    (uint8_t)SUPPORTED_SAMPLING_FREQ_16KHZ,
(uint8_t)(SUPPORTED_SAMPLING_FREQ_16KHZ >> 8),

    CODEC_CAPABILITY_LEN_SUPPORTED_FRAME_DURATIONS,
    CODEC_CAPABILITY_TYPE_SUPPORTED_FRAME_DURATIONS,
    SUPPORTED_FRAME_DURATIONS_7P5_MS | SUPPORTED_FRAME_DURATIONS_10_MS,

    CODEC_CAPABILITY_LEN_AUDIO_CHANNEL_COUNTS,
    CODEC_CAPABILITY_TYPE_AUDIO_CHANNEL_COUNTS,
    AUDIO_CHANNEL_COUNTS_1,

    CODEC_CAPABILITY_LEN_SUPPORTED_OCTETS_PER_CODEC_FRAME,
    CODEC_CAPABILITY_TYPE_SUPPORTED_OCTETS_PER_CODEC_FRAME,
    (uint8_t)SUPPORTED_OCTETS_PER_CODEC_FRAME_30_40,
  (uint8_t)(SUPPORTED_OCTETS_PER_CODEC_FRAME_30_40 >> 8),
    (uint8_t)(SUPPORTED_OCTETS_PER_CODEC_FRAME_30_40 >> 16),
  (uint8_t)(SUPPORTED_OCTETS_PER_CODEC_FRAME_30_40 >> 24),

    CODEC_CAPABILITY_LEN_SUPPORTED_MAX_CODEC_FRAMES_PER_SDU,
    CODEC_CAPABILITY_TYPE_SUPPORTED_MAX_CODEC_FRAMES_PER_SDU,
    MAX_SUPPORTED_LC3_FRAMES_PER_SDU_1,
};

static uint8_t g_pacs_codec_capabilities_24k[] = {
    :
}
:
static ble_pacs_pac_record_t g_pacs_pac_1[] = {
    {
        CODEC_ID_LC3,
        PACS_CODEC_CAPABLITIES_LEN,
        &g_pacs_codec_capabilities_16k[0],
        PACS_METADATA_LEN,
        &g_pacs_metadata[0],
    },
    :
};
```

```
ble_pacs_pac_t g_pacs_sink_pac_1 = {
    PACS_SINK_PAC_1_RECORD_NUM,
    &g_pacs_pac_1[0],
};

void ble_pacs_init_parameter(void)
{
    :
    ble_pacs_set_pac(AUDIO_DIRECTION_SINK, BLE_PACS_SINK_PAC_1,
&g_pacs_sink_pac_1);
    :
}
```

2.4.2.3 ASCS

Audio Stream Configuration Service (ASCS) is declared in
middleware/MTK/bt_le_audio/ascs/ble_ascs_service.c.
The Audio Stream Endpoint (ASE) is declared here. If you want to customize the number of ASEs, you can modify this file. For detail of ASCS, please check ASCS specification from Bluetooth SIG for reference.

The ASCS initial flow is included in ble_bap_init(), so Sink MW or upper layer doesn't need to call API to perform initialization.

If you want to change ASCS, the ASCS related code should be modified. For example, depending on product scenario, define the number of Audio Stream Endpoint (ASE).
Here is an example on how to define our ASEs for UMR audio product.

In ble_le_audio MW: (e.g. bt_le_audio/src/ascs/ble_ascs_service.c)

```
ble_ascs_attribute_handle_t g_ascs_att_handle_tbl[] = {
    {BLE_ASCS_UUID_TYPE_ASCS_SERVICE,        ASCS_START_HANDLE},
    /* BLE_ASCS_SINK_ASE_1 */
    {BLE_ASCS_UUID_TYPE_SINK_ASE,            ASCS_VALUE_HANDLE_SINK_ASE_1},
    /* BLE_ASCS_SINK_ASE_2 */
    {BLE_ASCS_UUID_TYPE_SINK_ASE,            ASCS_VALUE_HANDLE_SINK_ASE_2},
  :
    /* ASE Control Point */
    {BLE_ASCS_UUID_TYPE_ASE_CONTROL_POINT,
ASCS_VALUE_HANDLE_ASE_CONTROL_POINT},
    {BLE_ASCS_UUID_TYPE_INVALID,             ASCS_END_HANDLE},
};

bt_gatts_service_rec_t *ble_ascs_service_rec[] = {
    (const bt_gatts_service_rec_t *) &ble_ascs_primary_service,
    /* BLE_ASCS_SINK_ASE_1 */
    (const bt_gatts_service_rec_t *) &ble_ascs_char4_ascs_sink_ase_1,
    (const bt_gatts_service_rec_t *) &ble_ascs_sink_ase_value_1,
    (const bt_gatts_service_rec_t *) &ble_ascs_sink_ase_config_1,

    /* BLE_ASCS_SINK_ASE_2 */
    (const bt_gatts_service_rec_t *) &ble_ascs_char4_ascs_sink_ase_2,
    (const bt_gatts_service_rec_t *) &ble_ascs_sink_ase_value_2,
    (const bt_gatts_service_rec_t *) &ble_ascs_sink_ase_config_2,
    :
    /* BLE_ASCS_ASE_CONTROL_POINT  */
    (const bt_gatts_service_rec_t *) &ble_ascs_char4_ascs_ase_control_point,
    (const bt_gatts_service_rec_t *) &ble_ascs_ase_control_point_value,
    (const bt_gatts_service_rec_t *) &ble_ascs_ase_control_point_config,
};
```

2.4.2.4 VCS

Volume Control Service (VCS) is declared in middleware/MTK/bt_le_audio/vcp/ble_vcs_service.c. For detail of VCS, please check VCS specification from Bluetooth SIG for reference.

Since VSC includes secondary services called VOCS and AICS, if the secondary services exist, the content of VCS also needs to be modified.

```
#ifdef MTK_LEA_VOCS_ENABLE
#define BLE_VOCS_SECOND_SERVICE_COUNT 1
#else
#define BLE_VOCS_SECOND_SERVICE_COUNT 0
#endif

#ifdef MTK_LEA_AICS_ENABLE
#define BLE_AICS_SECOND_SERVICE_COUNT 1
#else
#define BLE_AICS_SECOND_SERVICE_COUNT 0
#endif

#define BLE_VCS_CHAR_START_HANDLE              (BLE_VCS_START_HANDLE + BLE_VOCS_SECOND_SERVICE_COUNT
+ BLE_AICS_SECOND_SERVICE_COUNT + 2)
#define BLE_VCS_VALUE_HANDLE_VOLUME_STATE         (BLE_VCS_CHAR_START_HANDLE)
#define BLE_VCS_VALUE_HANDLE_VOLUME_CONTROL_POINT  (BLE_VCS_CHAR_START_HANDLE + 3)
#define BLE_VCS_VALUE_HANDLE_VOLUME_FLAGS         (BLE_VCS_CHAR_START_HANDLE + 5)
#define BLE_VCS_END_HANDLE                        (BLE_VCS_CHAR_START_HANDLE + 6)
:
const bt_gatts_service_rec_t *ble_vcs_service_rec[] = {
    (const bt_gatts_service_rec_t *) &ble_vcs_primary_service,
#ifdef MTK_LEA_VOCS_ENABLE
    (const bt_gatts_service_rec_t *) &ble_vocs_secondary_service_included,
#endif
#ifdef MTK_LEA_AICS_ENABLE
    (const bt_gatts_service_rec_t *) &ble_aics_secondary_service_included,
#endif
    (const bt_gatts_service_rec_t *) &ble_vcs_char4_volume_state,
    (const bt_gatts_service_rec_t *) &ble_vcs_volume_state,
    (const bt_gatts_service_rec_t *) &ble_vcs_volume_state_client_config,
    (const bt_gatts_service_rec_t *) &ble_vcs_charc4_control_point,
    (const bt_gatts_service_rec_t *) &ble_vcs_control_point,
    (const bt_gatts_service_rec_t *) &ble_vcs_char4_volume_flags,
    (const bt_gatts_service_rec_t *) &ble_vcs_volume_flags,
    (const bt_gatts_service_rec_t *) &ble_vcs_volume_flags_client_config,
};
```

2.4.2.5 VOCS

Volume Offset Control Service (VOCS) is declared in middleware/MTK/bt_le_audio/vcp/ble_vocs_service.c
This GATT service is a secondary service of VCS and it is by default disabled in MediaTek SDK. If you want to enable this service, please define MTK_LEA_VOCS_ENABLE in your Makefile. For detail of VOCS, please check VOCS specification from Bluetooth SIG for reference.

2.4.2.6 AICS

Audio Input Control Service (AICS) is declared in middleware/MTK/bt_le_audio/vcp/ble_aics_service.c
This GATT service is a secondary service of VCS and it is by default disabled in MediaTek SDK. If you want to enable this service, please define MTK_LEA_AICS_ENABLE in your Makefile. For detail of AICS, please check AICS specification from Bluetooth SIG for reference.

2.4.3 Audio Manager

Audio Manager is a module responsible for managing audio related components such as audio driver and DSP.

2.5 Streaming

2.5.1 Media Control

Remote device will send corresponding media control command to local device. Since the media data will be decoded by DSP, SDK needs to configure or operate DSP when receiving codec configuration or media control from remote device.

Figure 4 shows how SDK controls DSP.

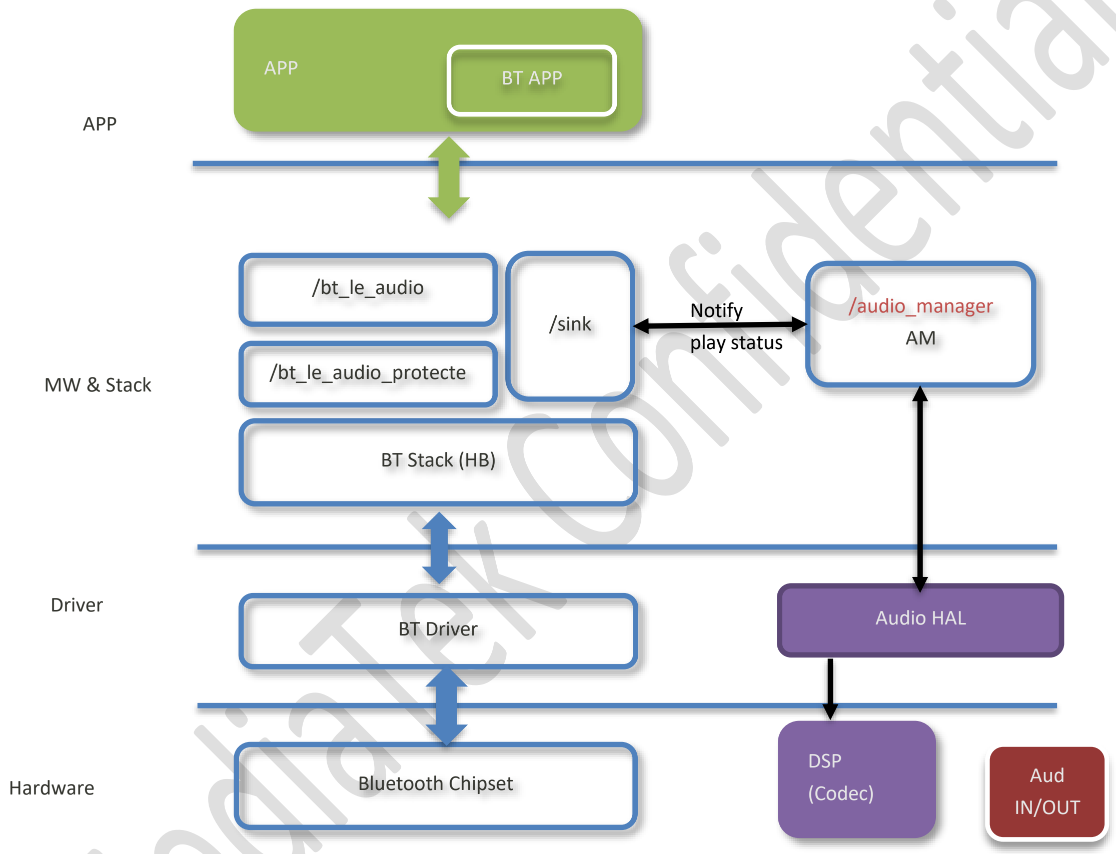


Figure 4 Flow of Media Control

2.5.2 Streaming Data

After streaming starts, remote will send media data to controller. Since DSP is used to decode it, the controller will send the media data to DSP directly. Then, DSP will send PCM data to Audio Driver.

Figure 5 shows how the sink module operates with AM module to configure and play DSP.

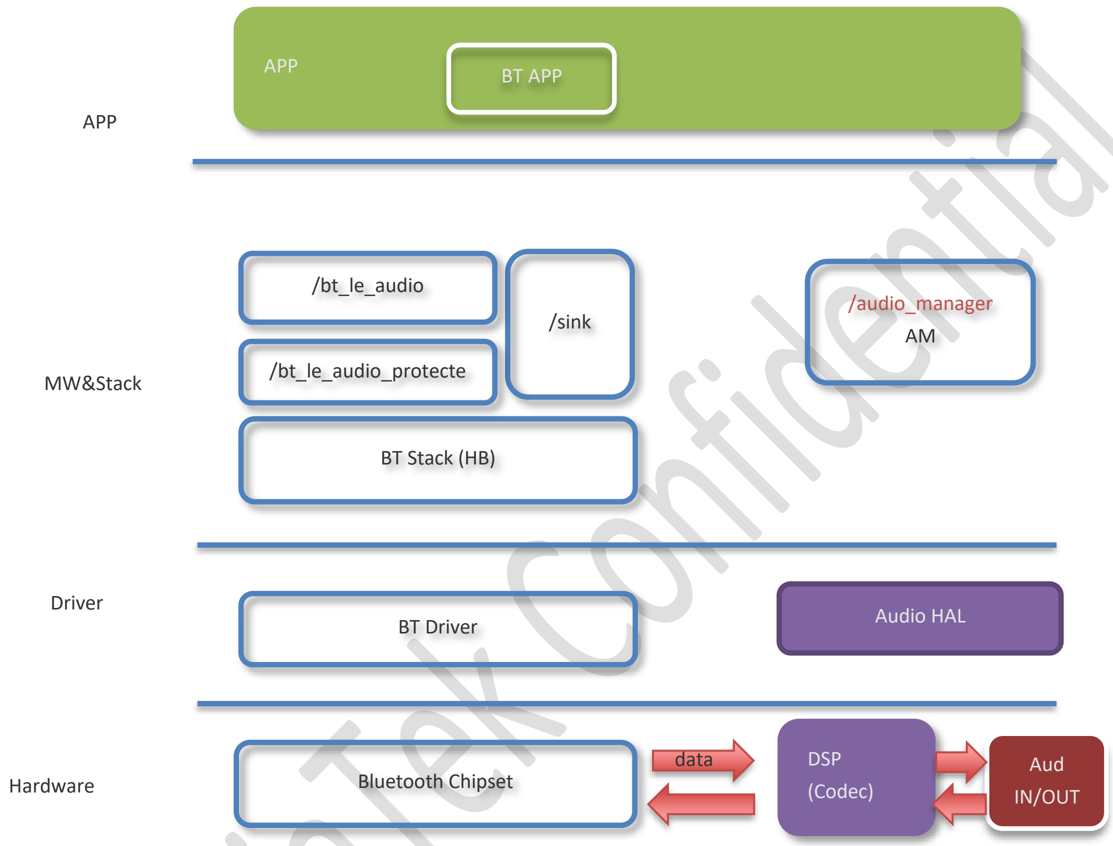


Figure 5 Flow of LC3 Streaming Data

3 Appendix A: CLI Usage Guide

In the demo project folder, CLI commands are available to demonstrate BT features. For example, in the bga_sdk_audio, we integrate the MCP feature as sample code for developing application.

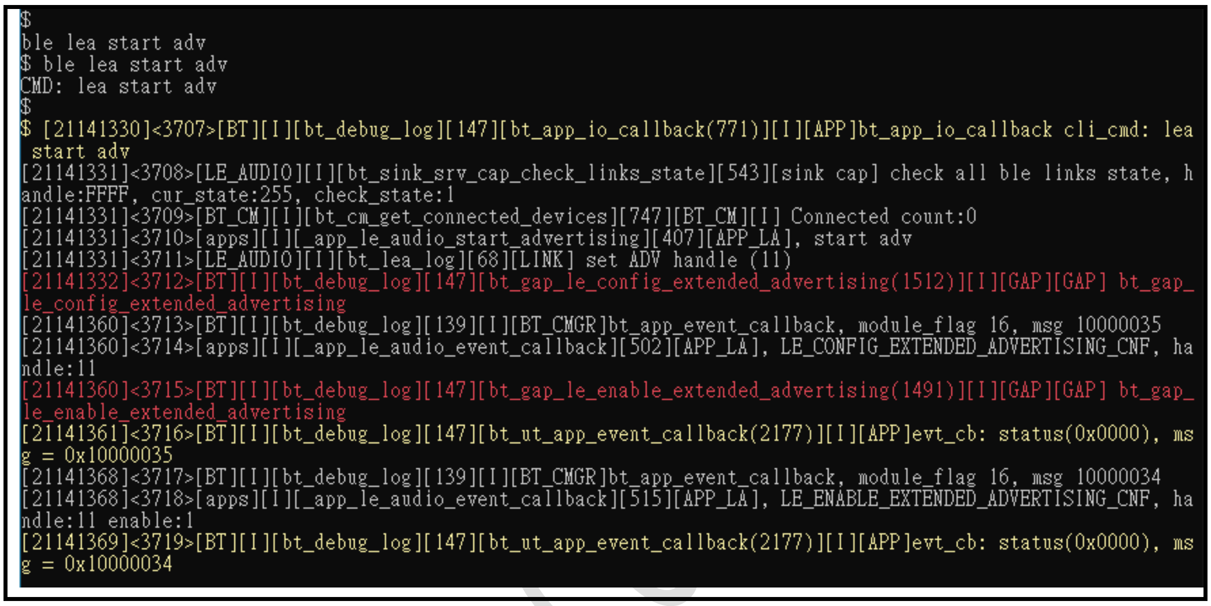
Before developing application, you can use the following CLI commands to check BT feasibility.

CLI Command	Description
ble init	Initialize BT host stack and power on BT module. If power-on is successful, you can see success log, [APP]BT_POWER_ON_CNF Success, in the debug port. /* Please make sure you do 'ble init' before using the supported CLI commands below*/
ble lea start adv	This command is used for manually starting advertising for LE Audio.
ble lea stop adv	This command is used for manually stopping advertising for LE Audio.
ble lea disconnect <type> <addr>	This command is used for disconnecting LE link.
ble lea unpair <type> <addr>	This command is used for unpairing bonded devices.
ble lea show paired list	This command is used for showing information of the bonded devices.
ble lea show conn list	This command is used for showing information of the connected LE devices.
ble lea get sup op	This command is used for showing the supported operation of remote device.
ble lea play	This command is used for starting to play music via LE Audio.
ble lea pause	This command is used for stopping playing music via LE Audio.
ble lea fwd	This command is used for switching to next track.
ble lea bwd	This command is used for switching to previous track.
ble lea get media state	This command is used for getting media state like play, pause etc.
ble lea get media player name	This command is used for showing the media player name of remote device.
ble lea get track title	This command is used for showing the track title of the current song.
ble lea get track duration	This command is used for showing the track duration of the current song.
ble lea get track position	This command is used for showing the track position of the current song.

ble lea get vol info	This command is used for getting current volume setting information.

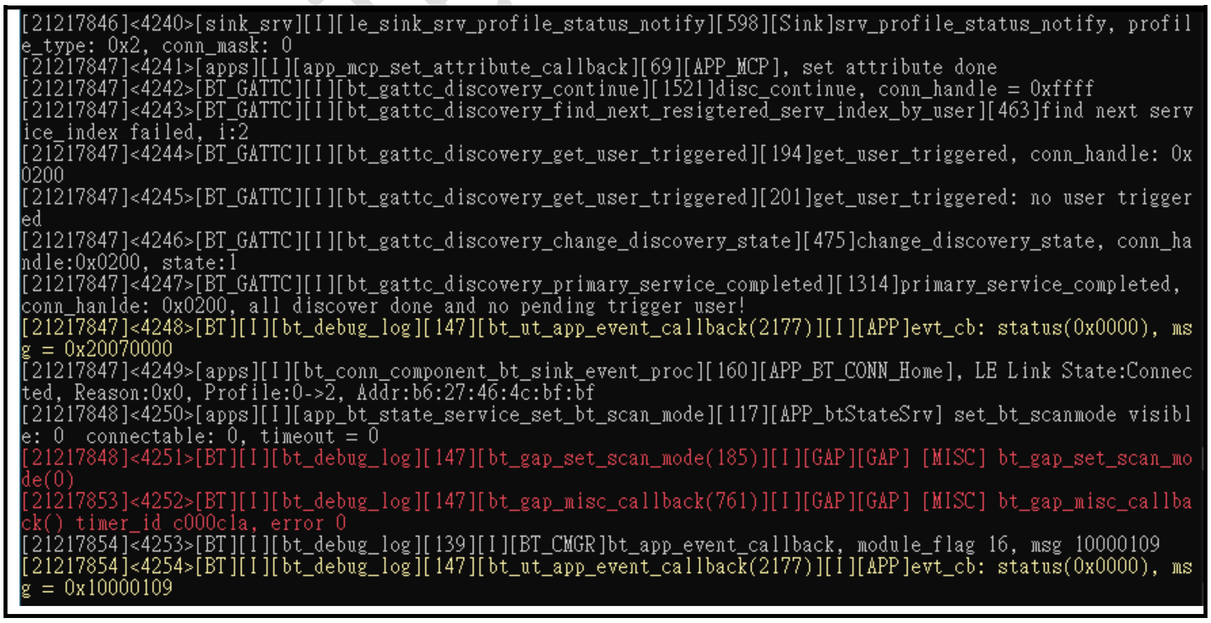
3.1 CLI Example of Connecting or Disconnecting Device

You can manually enable advertising for LE Audio. If successful, you can find LE_ENABLE_EXTENDED_ADVERTISING_CNF message in log.



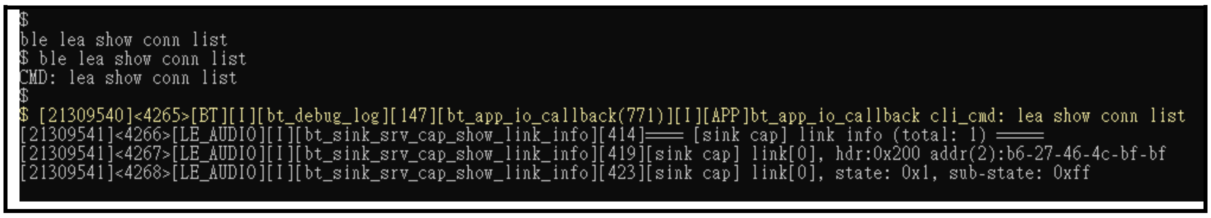
```
$
ble lea start adv
$ ble lea start adv
CMD: lea start adv
$
$ [21141330]<3707>[BT][I][bt_debug_log][147][bt_app_io_callback(771)][I][APP]bt_app_io_callback cli_cmd: lea
 start adv
[21141331]<3708>[LE_AUDIO][I][bt_sink_srv_cap_check_links_state][543][sink cap] check all ble links state, h
andle:FFFF, cur_state:255, check_state:1
[21141331]<3709>[BT_CM][I][bt_cm_get_connected_devices][747][BT_CM][I] Connected count:0
[21141331]<3710>[apps][I][_app_le_audio_start_advertising][407][APP_LA], start adv
[21141331]<3711>[LE_AUDIO][I][bt_lea_log][68][LINK] set ADV handle (11)
[21141332]<3712>[BT][I][bt_debug_log][147][bt_gap_le_config_extended_advertising(1512)][I][GAP][GAP] bt_gap_
le_config_extended_advertising
[21141360]<3713>[BT][I][bt_debug_log][139][I][BT_CMGR]bt_app_event_callback, module_flag 16, msg 10000035
[21141360]<3714>[apps][I][_app_le_audio_event_callback][502][APP_LA], LE_CONFIG_EXTENDED_ADVERTISING_CNF, ha
ndle:11
[21141360]<3715>[BT][I][bt_debug_log][147][bt_gap_le_enable_extended_advertising(1491)][I][GAP][GAP] bt_gap_
le_enable_extended_advertising
[21141361]<3716>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), ms
g = 0x10000035
[21141368]<3717>[BT][I][bt_debug_log][139][I][BT_CMGR]bt_app_event_callback, module_flag 16, msg 10000034
[21141368]<3718>[apps][I][_app_le_audio_event_callback][515][APP_LA], LE_ENABLE_EXTENDED_ADVERTISING_CNF, ha
ndle:11 enable:1
[21141369]<3719>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), ms
g = 0x10000034
```

Then, you can use remote device which supports Unicast Media Receiver to search for MTK_LEA_XXXXXXXXX and then connect it. After connection is set up successfully, you can find **LE Link State:Connected** message in log.



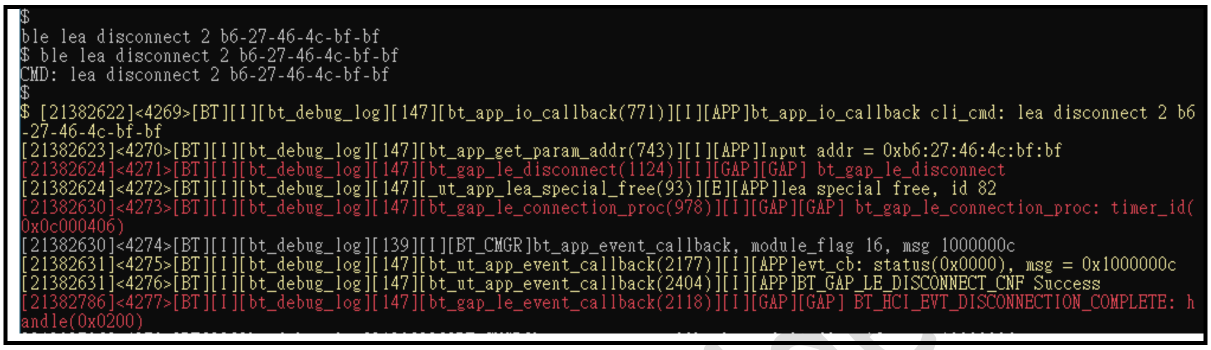
```
[21217846]<4240>[sink_srv][I][le_sink_srv_profile_status_notify][598][Sink]srv_profile_status_notify, profil
e_type: 0x2, conn_mask: 0
[21217847]<4241>[apps][I][app_mcp_set_attribute_callback][69][APP_MCP], set attribute done
[21217847]<4242>[BT_GATTC][I][bt_gattc_discovery_continue][1521]disc_continue, conn_handle = 0xffff
[21217847]<4243>[BT_GATTC][I][bt_gattc_discovery_find_next_resigtered_serv_index_by_user][463]find next serv
ice_index failed, i:2
[21217847]<4244>[BT_GATTC][I][bt_gattc_discovery_get_user_triggered][194]get_user_triggered, conn_handle: 0x
0200
[21217847]<4245>[BT_GATTC][I][bt_gattc_discovery_get_user_triggered][201]get_user_triggered: no user trigger
ed
[21217847]<4246>[BT_GATTC][I][bt_gattc_discovery_change_discovery_state][475]change_discovery_state, conn_ha
ndle:0x0200, state:1
[21217847]<4247>[BT_GATTC][I][bt_gattc_discovery_primary_service_completed][1314]primary_service_completed,
conn_hanlde: 0x0200, all discover done and no pending trigger user!
[21217847]<4248>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), ms
g = 0x20070000
[21217847]<4249>[apps][I][bt_conn_component_bt_sink_event_proc][160][APP_BT_CONN_Home], LE Link State:Connec
ted, Reason:0x0, Profile:0->2, Addr:b6:27:46:4c:bf:bf
[21217848]<4250>[apps][I][app_bt_state_service_set_bt_scan_mode][117][APP_btStateSrv] set_bt_scanmode visibl
e: 0  connectable: 0, timeout = 0
[21217848]<4251>[BT][I][bt_debug_log][147][bt_gap_set_scan_mode(185)][I][GAP][GAP] [MISC] bt_gap_set_scan_mo
de(0)
[21217853]<4252>[BT][I][bt_debug_log][147][bt_gap_misc_callback(761)][I][GAP][GAP] [MISC] bt_gap_misc_callba
ck() timer_id c000c1a, error 0
[21217854]<4253>[BT][I][bt_debug_log][139][I][BT_CMGR]bt_app_event_callback, module_flag 16, msg 10000109
[21217854]<4254>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), ms
g = 0x10000109
```

Then, you can check connection list by CLI command.



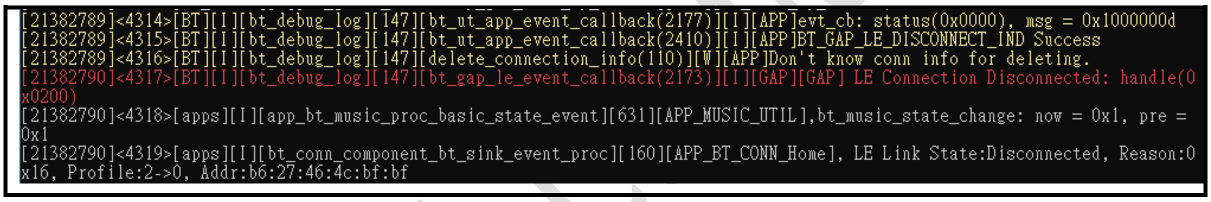
```
$
ble lea show conn list
$ ble lea show conn list
CMD: lea show conn list
$
$ [21309540]<4265>[BT][I][bt_debug_log][147][bt_app_io_callback(771)][I][APP]bt_app_io_callback cli_cmd: lea show conn list
[21309541]<4266>[LE_AUDIO][I][bt_sink_srv_cap_show_link_info][414]=== [sink cap] link info (total: 1) ===
[21309541]<4267>[LE_AUDIO][I][bt_sink_srv_cap_show_link_info][419][sink cap] link[0], hdr:0x200 addr(2):b6-27-46-4c-bf-bf
[21309541]<4268>[LE_AUDIO][I][bt_sink_srv_cap_show_link_info][423][sink cap] link[0], state: 0x1, sub-state: 0xff
```

With the information, you can use it to disconnect or unpair remote device.



```
$
ble lea disconnect 2 b6-27-46-4c-bf-bf
$ ble lea disconnect 2 b6-27-46-4c-bf-bf
CMD: lea disconnect 2 b6-27-46-4c-bf-bf
$
$ [21382622]<4269>[BT][I][bt_debug_log][147][bt_app_io_callback(771)][I][APP]bt_app_io_callback cli_cmd: lea disconnect 2 b6
-27-46-4c-bf-bf
[21382623]<4270>[BT][I][bt_debug_log][147][bt_app_get_param_addr(743)][I][APP]Input addr = 0xb6:27:46:4c:bf:bf
[21382624]<4271>[BT][I][bt_debug_log][147][bt_gap_le_disconnect(1124)][I][GAP][GAP] bt_gap_le_disconnect
[21382624]<4272>[BT][I][bt_debug_log][147][_ut_app_lea_special_free(93)][E][APP]lea special free, id 82
[21382630]<4273>[BT][I][bt_debug_log][147][bt_gap_le_connection_proc(978)][I][GAP][GAP] bt_gap_le_connection_proc: timer_id(
0x0c000406)
[21382630]<4274>[BT][I][bt_debug_log][139][I][BT_CMGR]bt_app_event_callback, module_flag 16, msg 1000000c
[21382631]<4275>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), msg = 0x1000000c
[21382631]<4276>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2404)][I][APP]BT_GAP_LE_DISCONNECT_CNF Success
[21382786]<4277>[BT][I][bt_debug_log][147][bt_gap_le_event_callback(2118)][I][GAP][GAP] BT_HCI_EVT_DISCONNECTION_COMPLETE: h
andle(0x0200)
```

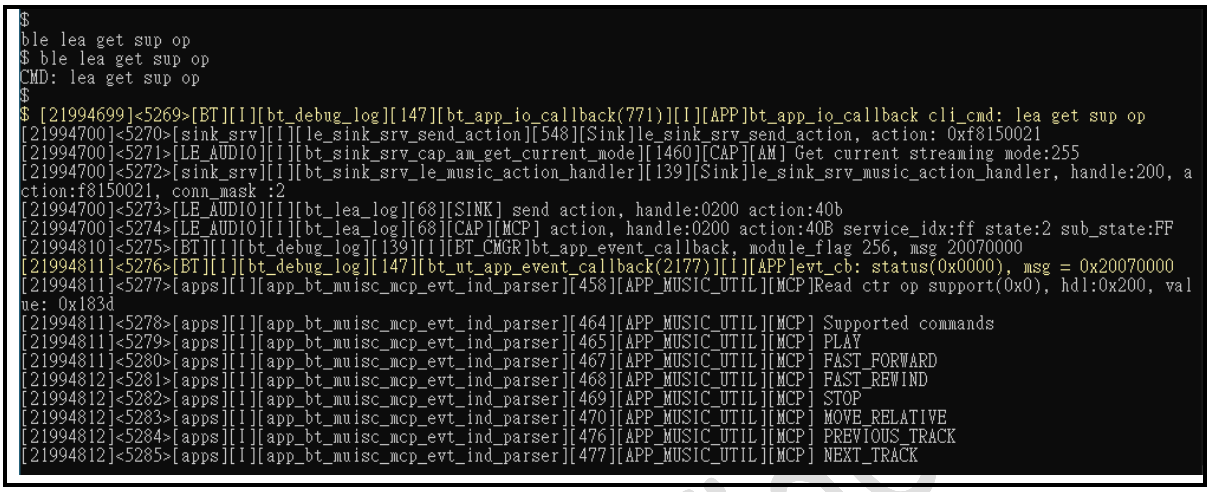
After disconnection is successful, you can find **LE Link State:Disconnected** message in log.



```
[21382789]<4314>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), msg = 0x1000000d
[21382789]<4315>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2410)][I][APP]BT_GAP_LE_DISCONNECT_IND Success
[21382789]<4316>[BT][I][bt_debug_log][147][delete_connection_info(110)][W][APP]Don't know conn info for deleting.
[21382790]<4317>[BT][I][bt_debug_log][147][bt_gap_le_event_callback(2173)][I][GAP][GAP] LE Connection Disconnected: handle(0
x0200)
[21382790]<4318>[apps][I][app_bt_music_proc_basic_state_event][631][APP_MUSIC_UTIL],bt_music_state_change: now = 0x1, pre =
0x1
[21382790]<4319>[apps][I][bt_conn_component_bt_sink_event_proc][160][APP_BT_CONN_Home], LE Link State:Disconnected, Reason:0
x16, Profile:2->0, Addr:b6:27:46:4c:bf:bf
```

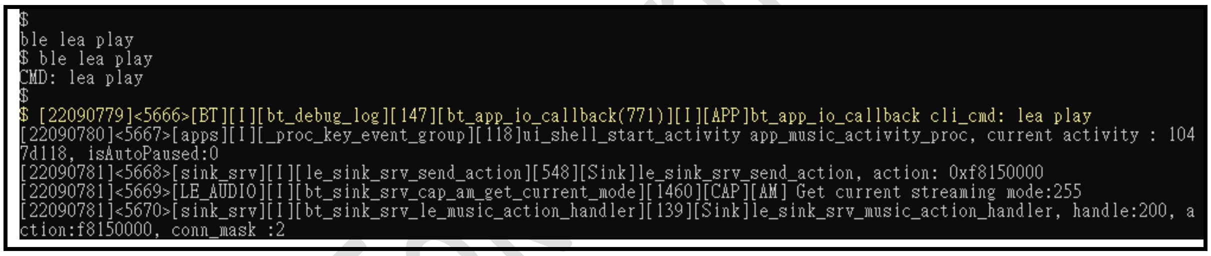
3.2 CLI Example of Controlling Music

You can use CLI command to check supported operation of remote device. Log will show all the supported commands.



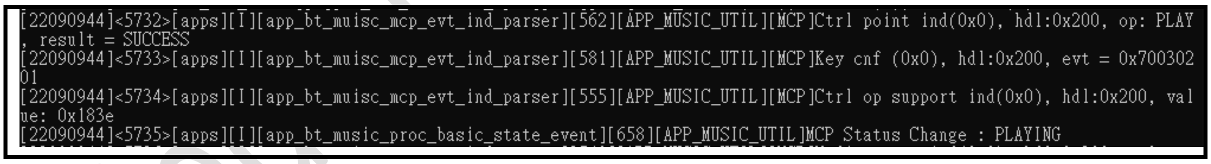
```
$
ble lea get sup op
$ ble lea get sup op
CMD: lea get sup op
$
$ [21994699]<5269>[BT][I][bt_debug_log][147][bt_app_io_callback(771)][I][APP]bt_app_io_callback cli_cmd: lea get sup op
[21994700]<5270>[sink_srv][I][le_sink_srv_send_action][548][Sink]le_sink_srv_send_action, action: 0xf8150021
[21994700]<5271>[LE_AUDIO][I][bt_sink_srv_cap_am_get_current_mode][1460][CAP][AM] Get current streaming mode:255
[21994700]<5272>[sink_srv][I][bt_sink_srv_le_music_action_handler][139][Sink]le_sink_srv_music_action_handler, handle:200, a
ction:f8150021, conn_mask :2
[21994700]<5273>[LE_AUDIO][I][bt_lea_log][68][SINK] send action, handle:0200 action:40b
[21994700]<5274>[LE_AUDIO][I][bt_lea_log][68][CAP][MCP] action, handle:0200 action:40B service_idx:ff state:2 sub_state:FF
[21994810]<5275>[BT][I][bt_debug_log][139][I][BT_CMGR]bt_app_event_callback, module_flag 256, msg 20070000
[21994811]<5276>[BT][I][bt_debug_log][147][bt_ut_app_event_callback(2177)][I][APP]evt_cb: status(0x0000), msg = 0x20070000
[21994811]<5277>[apps][I][app_bt_muisc_mcp_evt_ind_parser][458][APP_MUSIC_UTIL][MCP]Read ctr op support(0x0), hdl:0x200, val
ue: 0x183d
[21994811]<5278>[apps][I][app_bt_muisc_mcp_evt_ind_parser][464][APP_MUSIC_UTIL][MCP] Supported commands
[21994811]<5279>[apps][I][app_bt_muisc_mcp_evt_ind_parser][465][APP_MUSIC_UTIL][MCP] PLAY
[21994811]<5280>[apps][I][app_bt_muisc_mcp_evt_ind_parser][467][APP_MUSIC_UTIL][MCP] FAST_FORWARD
[21994812]<5281>[apps][I][app_bt_muisc_mcp_evt_ind_parser][468][APP_MUSIC_UTIL][MCP] FAST_REWIND
[21994812]<5282>[apps][I][app_bt_muisc_mcp_evt_ind_parser][469][APP_MUSIC_UTIL][MCP] STOP
[21994812]<5283>[apps][I][app_bt_muisc_mcp_evt_ind_parser][470][APP_MUSIC_UTIL][MCP] MOVE_RELATIVE
[21994812]<5284>[apps][I][app_bt_muisc_mcp_evt_ind_parser][476][APP_MUSIC_UTIL][MCP] PREVIOUS_TRACK
[21994812]<5285>[apps][I][app_bt_muisc_mcp_evt_ind_parser][477][APP_MUSIC_UTIL][MCP] NEXT_TRACK
```

If remote device is supported, you could use CLI command to send operation to remote device. For example, if you want to play music, you could use 'ble lea play'.



```
$
ble lea play
$ ble lea play
CMD: lea play
$
$ [22090779]<5666>[BT][I][bt_debug_log][147][bt_app_io_callback(771)][I][APP]bt_app_io_callback cli_cmd: lea play
[22090780]<5667>[apps][I][_proc_key_event_group][118]ui_shell_start_activity app_music_activity_proc, current activity : 104
7d118, isAutoPaused:0
[22090781]<5668>[sink_srv][I][le_sink_srv_send_action][548][Sink]le_sink_srv_send_action, action: 0xf8150000
[22090781]<5669>[LE_AUDIO][I][bt_sink_srv_cap_am_get_current_mode][1460][CAP][AM] Get current streaming mode:255
[22090781]<5670>[sink_srv][I][bt_sink_srv_le_music_action_handler][139][Sink]le_sink_srv_music_action_handler, handle:200, a
ction:f8150000, conn_mask :2
```

After play operation is successful, you can find MCP Status Change : PLAYING message in log.



```
[22090944]<5732>[apps][I][app_bt_muisc_mcp_evt_ind_parser][562][APP_MUSIC_UTIL][MCP]Ctrl point ind(0x0), hdl:0x200, op: PLAY
, result = SUCCESS
[22090944]<5733>[apps][I][app_bt_muisc_mcp_evt_ind_parser][581][APP_MUSIC_UTIL][MCP]Key cnf (0x0), hdl:0x200, evt = 0x700302
01
[22090944]<5734>[apps][I][app_bt_muisc_mcp_evt_ind_parser][555][APP_MUSIC_UTIL][MCP]Ctrl op support ind(0x0), hdl:0x200, val
ue: 0x183e
[22090944]<5735>[apps][I][app_bt_music_proc_basic_state_event][658][APP_MUSIC_UTIL]MCP Status Change : PLAYING
```

Exhibit 1 Terms and Conditions