Airoha IoT SDK MCSync Developer's Guide

Version: 2.0

Release date: 20 November 2020

Document Revision History

Revision	Date	Description
1.0	20 May 2019	Initial release
2.0	20 November 2020	Add MCSync Share mode

Table of Contents

Lists of Tables and Figures

1. Introduction

Multi Cast Synchronization (MCSync) is an Airoha proprietary profile to support voice/audio over multiple Bluetooth Audio devices. MCSync Link is an extension for the Bluetooth link with a MCSync packet/role extension mechanism. There is no extra cost for a piconet switch and less overhead compared with other solutions because only one Wireless Link is used in MCSync.

There are three modes in MCSync: Earbuds Mode , Speaker Mode and MCsync share Mode. Earbuds Mode is currently available.

About MCSync share is a special design to extend number of devices from 2 in Earbuds Mode to 4 in MCSync share Mode.

This document guides you through:

- Support for Bluetooth with the library description and supported reference examples.
- Detailed descriptions of the BR/EDR profiles.
- Bluetooth LE profiles.
- Custom application development and debugging logs.

1.1. Profile overview

Figure 1 shows the protocols and entities used in this profile.

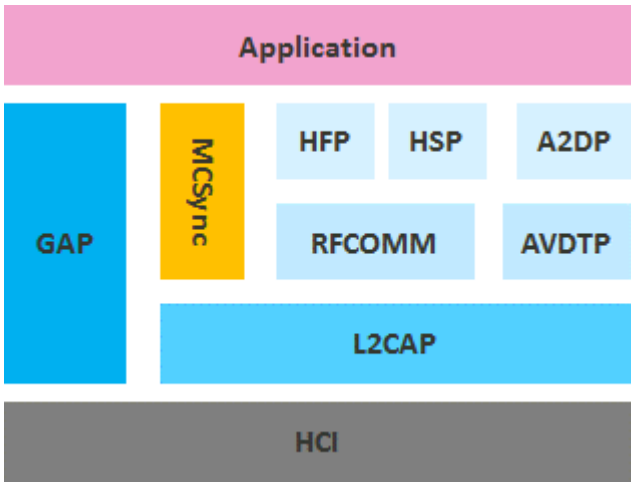

Figure 1. Protocol Model

The HCI, L2CAP, GAP, RFCOMM, AVDTP, HFP, HSP, and A2DP protocols are described in the *Airoha_IoT_SDK_Bluetooth_Developers_Guide.docx* document. MCSync communicates with GAP and Application before more than one audio device can connect (i.e. link establishment), and communicates with L2CAP and Application if data which is in the same format as ACL data is transferred between or among the multiple devices more than one device is connected. The profile information related to audio (i.e. HFP, HSP, and A2DP) are passed to MCSync via Application.

1.2. Configurations and Roles

MCSync defines three roles for smart devices: Agent; Client; and Partner.

Agent connects with the smart device and acts as a delegator to the other device(s) in different roles. It shares important link information with the other smart devices.

Client passively receives the data from either the smart device or from Agent. It does not send data to the smartphone or Agent.

Partner is same as Client, but it can communicate with Agent.

MCSync defines two links: the MCSync link, and the MCSync special link. There is one Agent and one Partner on the MCSync special link. The Agent is not connected to the smartphone or any other smart device. For the MCSync link, Agent is connected with the smartphone or smart device, and Partner is connected to the Agent. The MCSync roles and link are shown in Figure 2.

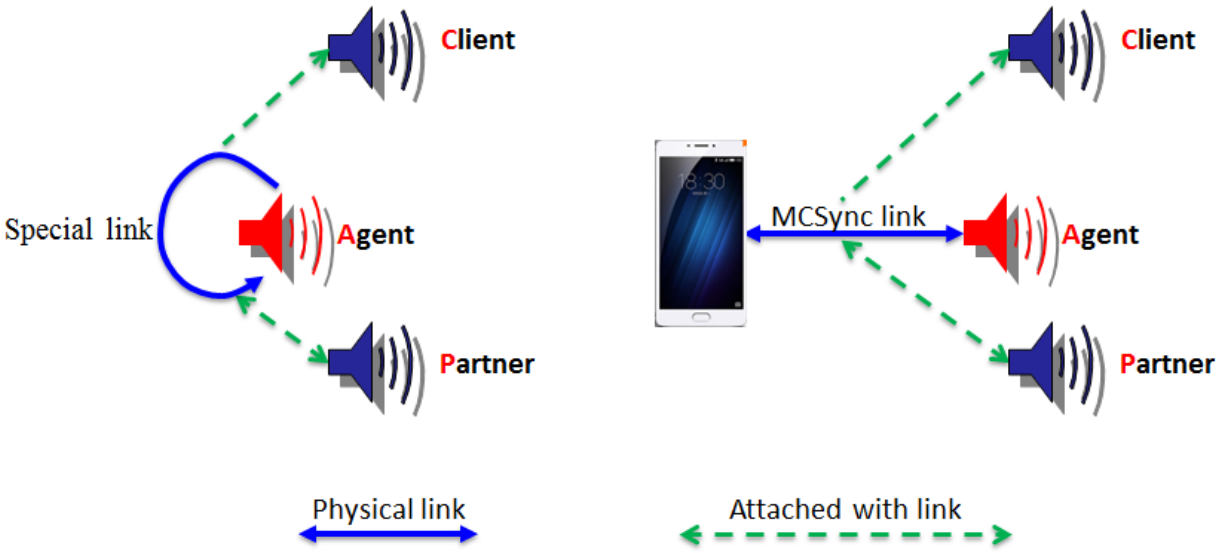

Figure 2. MCSync roles and link

1.3. Related SDK library requested

MCSync can only be run on IoT SDK for BT-Audio platform with the requested library files to interface the Bluetooth with C source and header files related to the platform, as shown in Table 1.

Table 1. Airoha IoT SDK library support for MCSync

Module	File Name	Location	Function
Bluetooth	libbt.a	/middleware/bluetooth/lib/	BR/EDR and Bluetooth LE stack library
	libbtdriver_[chip].a		Bluetooth driver library
	libbt_hfp.a		HFP library
	libbt_hsp.a		HSP library
	libbt_a2dp.a		A2DP library
	libbt_avrcp.a		AVRCP library, including the PASS THROUGH command
	libbt_aws_mce.a		MCSync library, including MCSync implementation
	bt_platform.h		Interface for Bluetooth tasks

Module	File Name	Location	Function
	bt_type.h	/middleware/bluetooth/inc/	Common data types
	bt_system.h		Interface for the system, such as power on or off, memory initiation and callback APIs for event handling
	bt_uuid.h		Interface for the UUID
	bt_a2dp.h		Interface for the A2DP
	bt_codec.h		Interface for the codec
	bt_hfp.h		Interface for the HFP
	bt_hsp.h		Interface for the HSP
	bt_avrcp.h		Interface for the AVRCP
	bt_aws_mce.h		Interface for the MCSync
	bt_gap.h		Interface for the GAP
	bt_sdp.h		Interface for the SDP
	bt_os_layer_api.h		Wrapper APIs for RTOS, memory, advanced encryption standard (AES) and rand
	bt_debug.h		Encapsulated debugging interface
	bt_hci_log.h		Encapsulated interface for the HCI logging
	bt_os_layer_api.c	/middleware/bluetooth/src/	Encapsulated interface for system, memory or AES. Developers can replace the implementation when porting to other platforms
	bt_debug.c		Encapsulated debugging interface. Developers can replace the implementation when porting to other platforms
	bt_hci.c		Encapsulated interface for the HCI logging. Developers can replace the implementation when porting to other platforms
	bt_task.c		The default Bluetooth task entry function
	bt_log.c		The definition for the debugging string used in BT Stack library

2. The MCSync Profile

2.1. The state machine for Agent

There are only two states for Partner: Connected and Disconnected. There are three states defined for Agent: Inactive, Connectable, and Attached. The Inactive state is the primary state in which the partner cannot successfully connect to the Agent. The Agent changes to the Inactive state when Partner disconnects from Agent. The Agent must change to the Connectable state when it wants to connect with Partner. Agent changes to the Attached state when it is connected to Partner. Please refer Figure 3 for the state diagram.

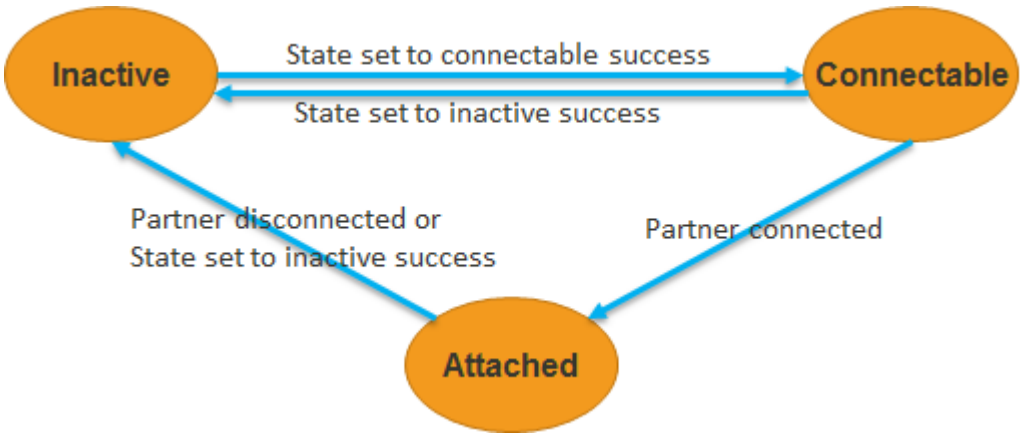

Figure 3. MCSync state diagram

2.2. The MCSync message sequence

There are four operations available with MCSync: connection establishment, connection release, send information, and role handover. The message sequence for each procedure is described in the following sections.

2.2.1. Connection establishment

Use the connection establishment operation to establish a connection between Agent and Partner. There are two different flows for the connections (depending on the connection with Phone) as shown in Figure 4 and Figure 5. The SDK provides two different message sequences: one for Agent (left) and another for Partner (right). For more details, refer to `bt_aws_mce.h`.

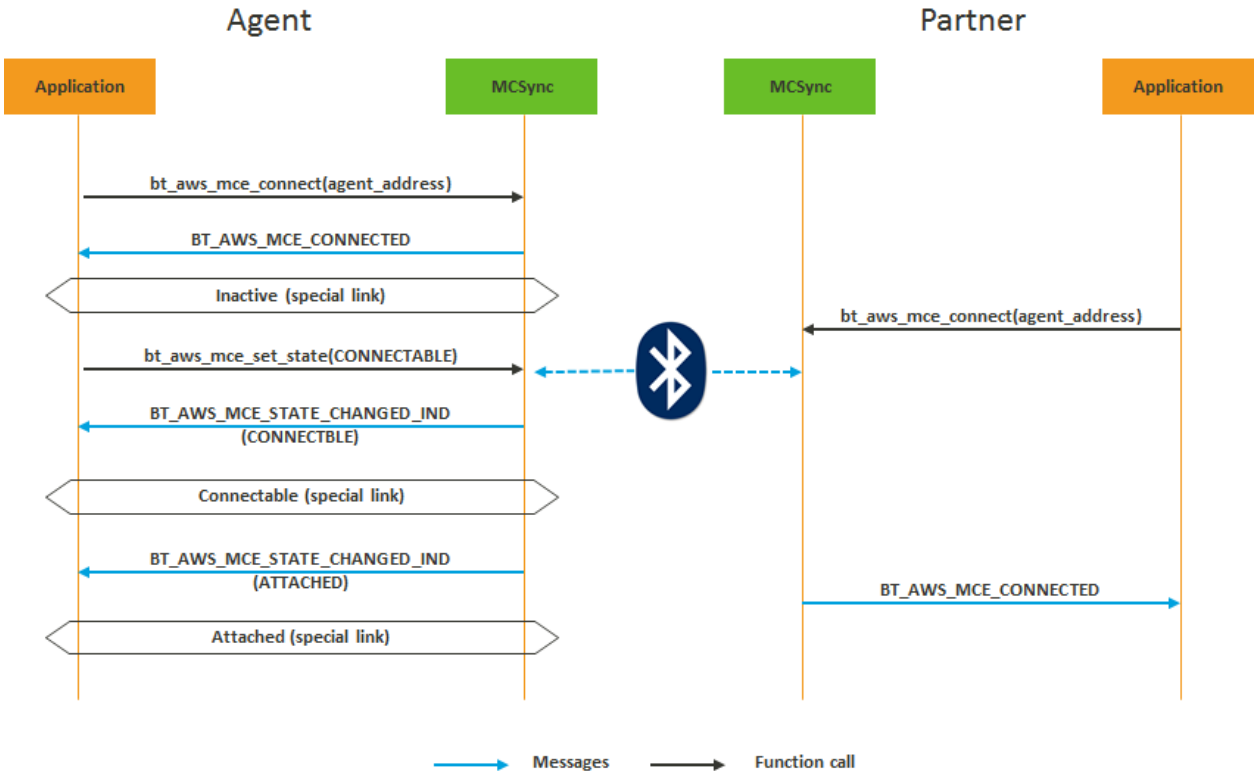

Figure 4. MCSync connection establishment message sequence without a smartphone

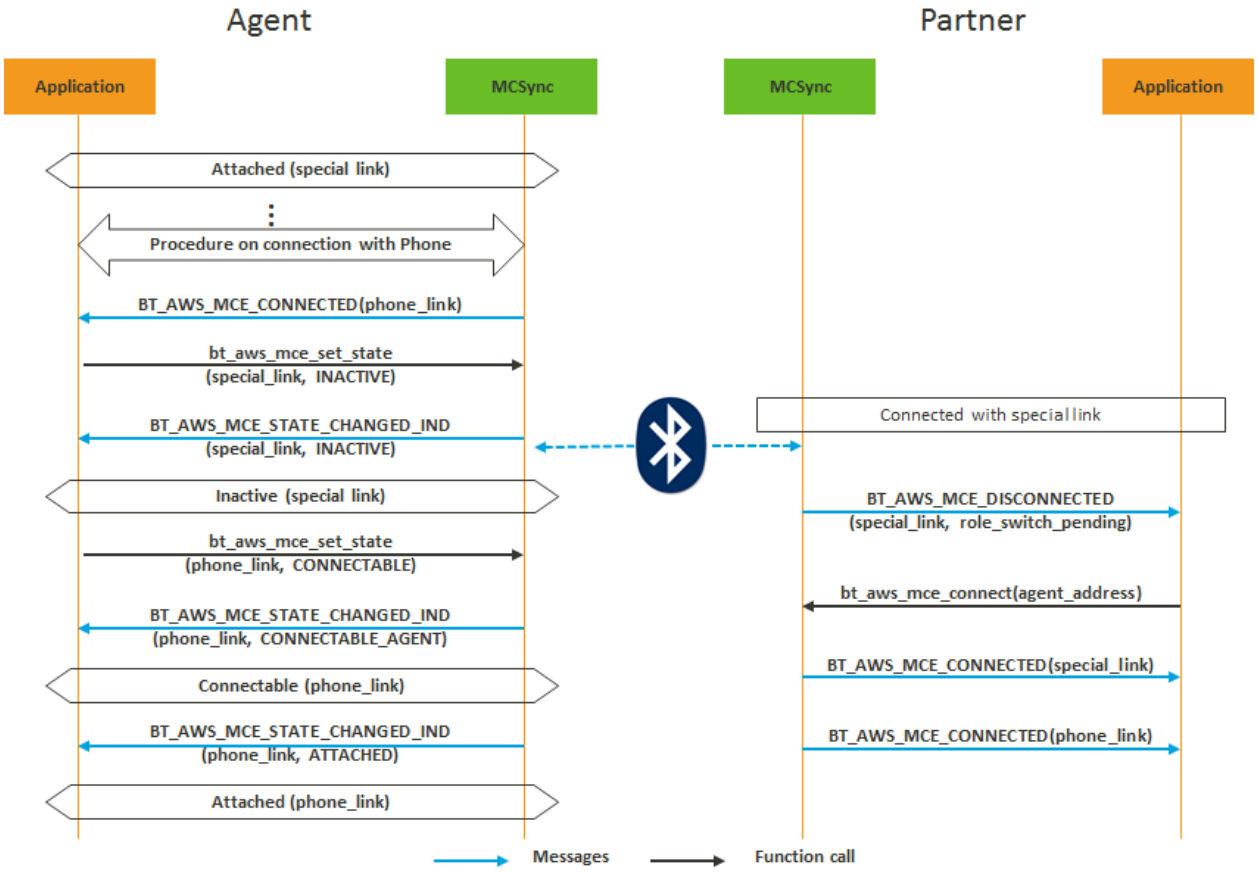

Figure 5. MCSync connection establishment message sequence with a smartphone

2.2.2. Connection release

The connection release procedure disconnects the MCSync link. There are three different flows for connection release (depending the initiator and connection status with phone) as shown in Figure 6, Figure 7, and Figure 8. For more details, refer to `bt_aws_mce.h`.

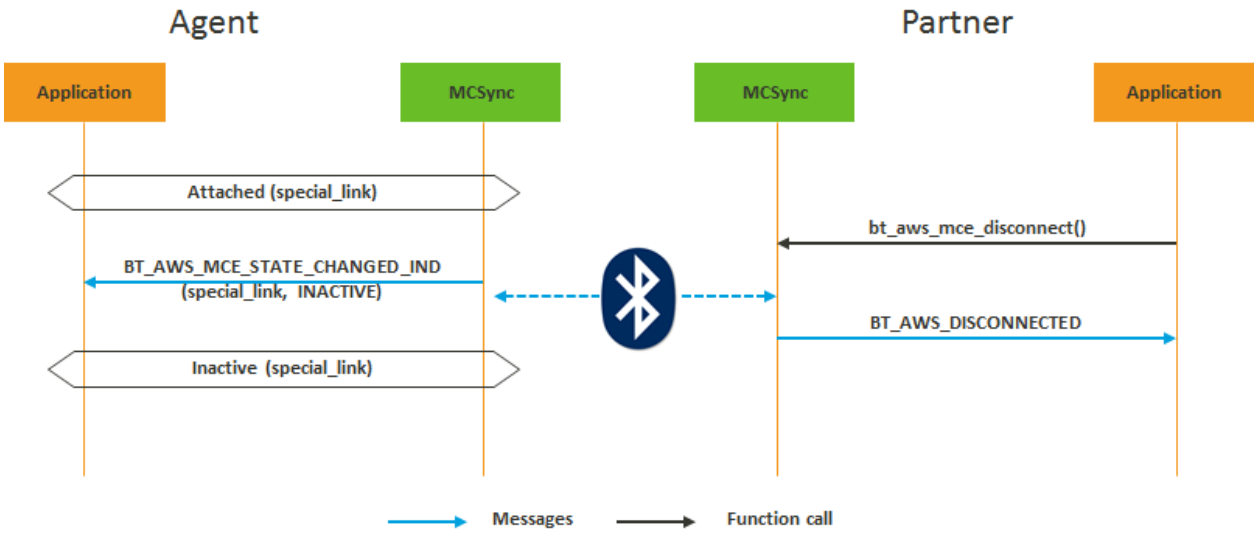

Figure 6. Message sequence when the MCSync special link is released from Partner

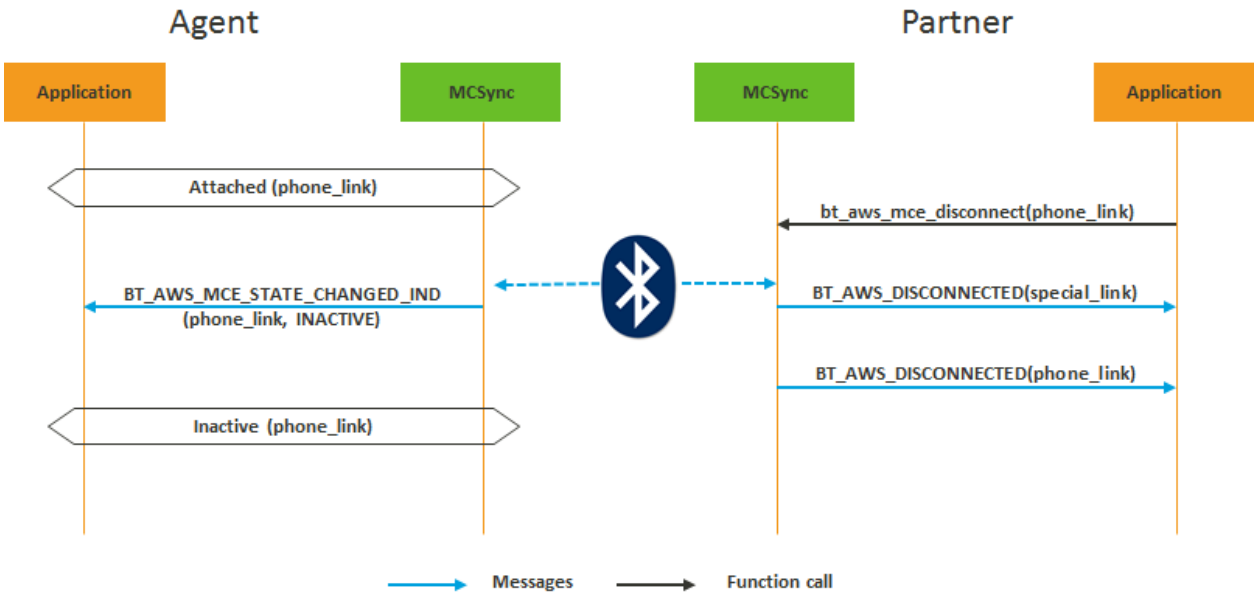

Figure 7. Message sequence when the MCSync link is released from Partner

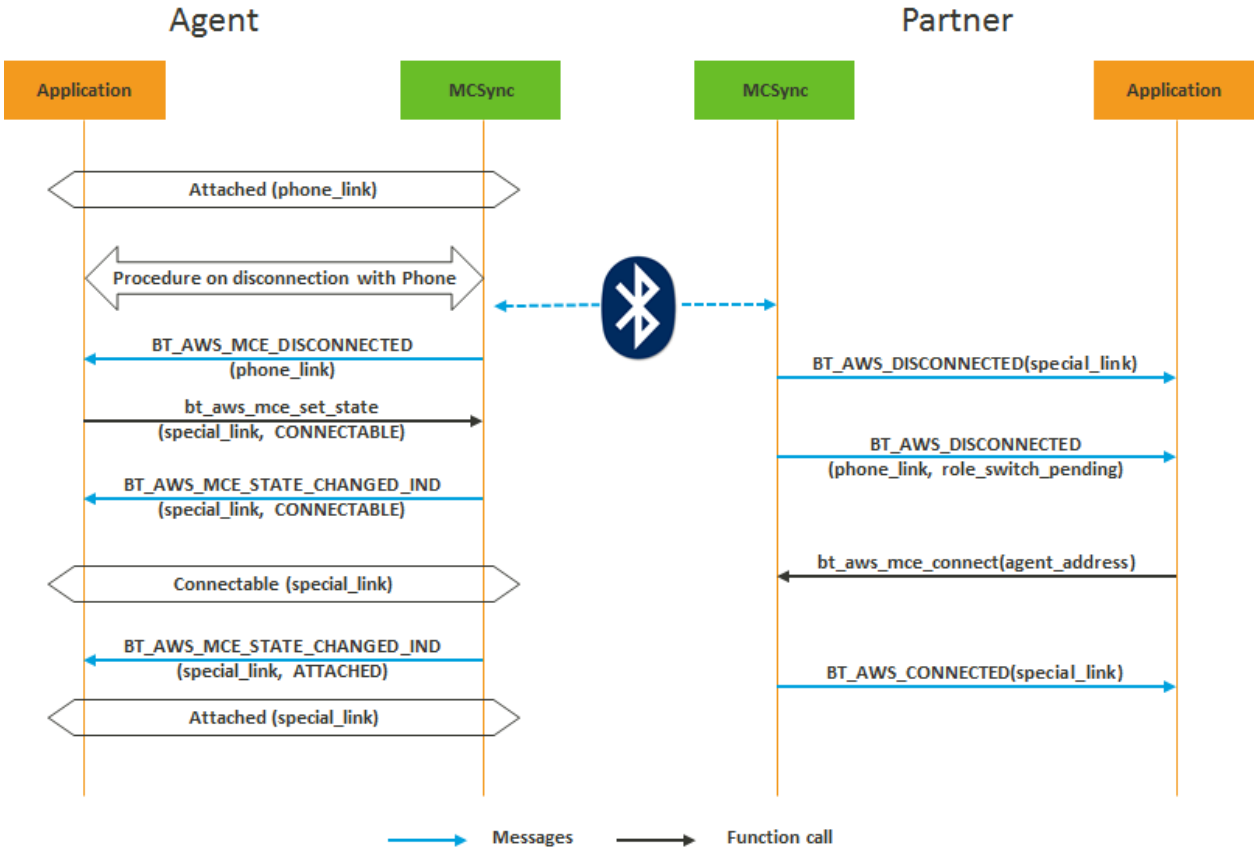

Figure 8. Message sequence when the MCSync link is released from the smartphone

2.2.3. Send information

When the connection between Agent and Partner is established, the application on Agent can send some information with the corresponding application on Partner, and vice versa. For instance, Agent can send music

playing information to Partner if the user plays music on the smartphone, or Partner can send volume change info to Agent if the user adjusts the music volume on Partner. The message flow is shown in Figure 9. For more details, refer to `bt_aws_mce.h`.

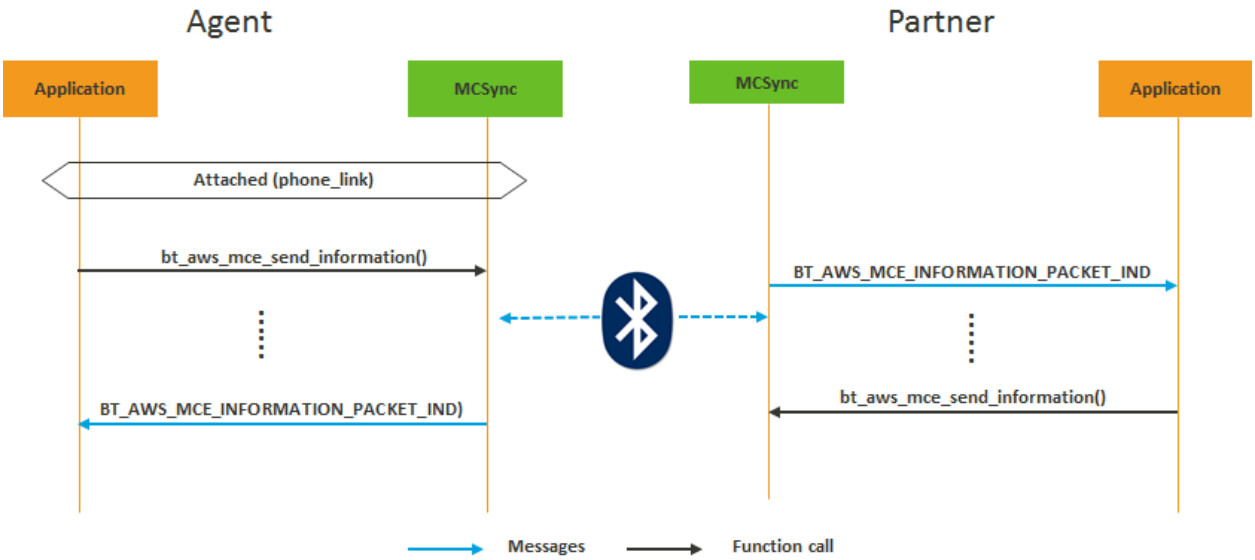

Figure 9. Send information message sequence

2.2.4. Role handover

Role handover is the process in which Agent and Partner change roles. Usually, the first earbud taken out of the charger case acts as the Agent, and the other earbud acts as Partner. The role handover process is triggered if the first earbud taken out of the case is already in the Partner role. The information on the Agent side (including profiles connection context and the important application data) must be sent to the Partner during the Role handover process. The message flow is shown in Figure 10. For more details, refer to `bt_aws_mce.h`.

Please note that this procedure cannot be performed if a SPP connection or BLE connection has been made; the length of the application data sent must be less than 100 Bytes.

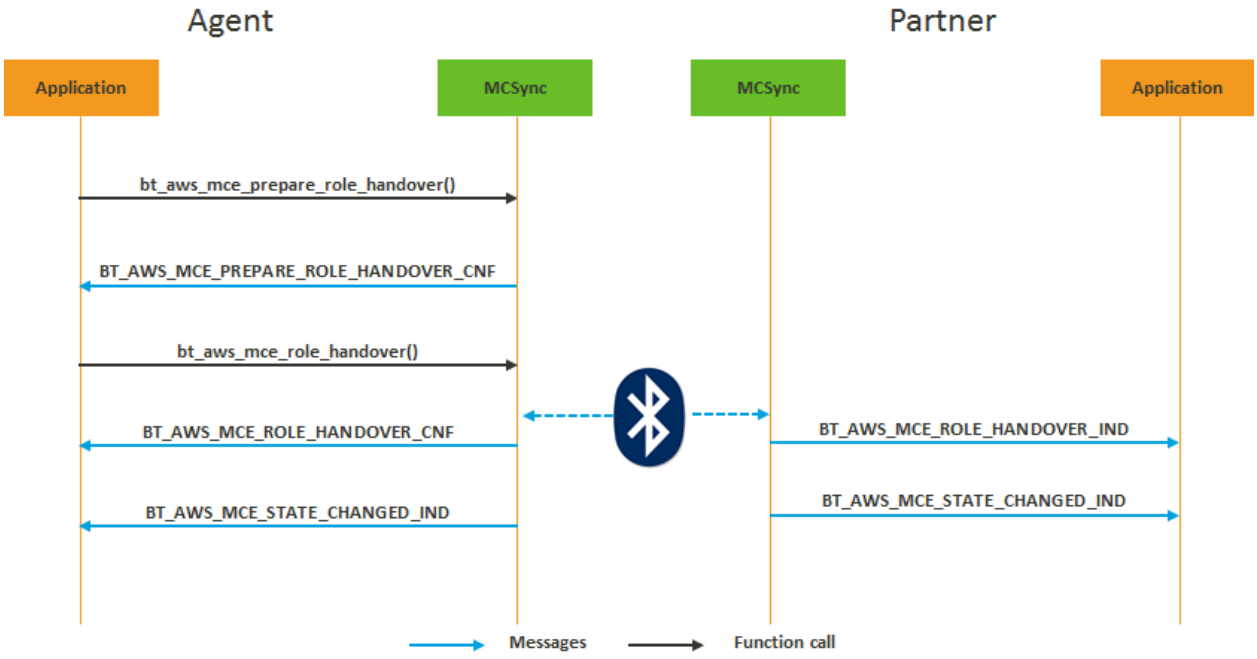

Figure 10. Role handover message sequence

2.3. Information packet format

The application can transfer information between Agent and Partner via the MCSync profile, information such as the volume value, LED flash signal, key code, etc. MCSync provides the bt_aws_mce_send_information API to support the information synchronization. The information is clarified as a different type, and defined with a tuple containing type, length, and value. For the detailed information types, please refer Table 2.

Please note that only the payload types listed in the table can be used. Otherwise, the data cannot be successfully transferred.

Table 2. MCSync Information Type

IF Type	Payload Type	Note
0x81	A2DP Information	Information for Client to Sync A2DP Status
0x82	Call Information	Information for Client to Sync Call Status
0x85	Application Report	Agent or Partner's Application Data (e.g. FOTA or VP)
0x87	Role Handover	Action to ask Partner to handover to Agent (refer to section 2.2.4)

2.4. Using the MCSync APIs

This section describes how to use the MCSync APIs for application development. The functionality of the MCSync APIs is implemented in the library, but related APIs can be found in `bt_aws_mce.h`.

1) Call the `bt_aws_mce_init_role()` function to start the MCSync role during the initiation process when the system powers on.

```
bt_aws_mce_init_role(aws_mce_role);
```

2) Partner must call the `bt_aws_mce_connect()` function to establish a link with Agent, and expects to receive the `BT_AWS_MCE_CONNECTED` event as shown in `bt_app_event_callback(bt_msg_type_t msg_id, bt_status_t status, void *buff)`. For Agent, this function should be also be called to set a special link when the role is set so that Partner can connect to Agent and data can be transferred

between them, even if Agent is not connected with the smartphone. To make Partner attach to the link, the Agent state must be switched to CONNECTABLE.

```
bt_aws_mce_init_role(aws_mce_role);
bt_aws_mce_connect(&aws_handle, agent_address);
```

```
bt_app_event_callback(bt_msg_type_t msg_id, bt_status_t status, void *buff)
{
switch (msg_id)
{
    case BT_AWS_MCE_CONNECTED:
{
    bt_aws_mce_connected_t *conn = (bt_aws_mce_connected_t *)buff;
    uint32_t aws_handle = conn->handle;
    if (aws_role == BT_AWS_MCE_ROLE_AGENT) {
bt_aws_mce_set_state(aws_handle,
BT_AWS_MCE_STATE_CONNECTABLE_AGENT);
    }
    break;
}
case BT_AWS_MCE_STATE_CHANGED_IND:
{
      bt_aws_mce_state_change_ind_t *changed_data =
(bt_aws_mce_state_change_ind_t *)buff;
    break;
}
default:
    break;
}
}
```

3) At the Partner end, call the following API to disconnect from Agent and then expect to receive the BT_AWS_DISCONNECTED event as shown in `bt_app_event_callback(bt_msg_type_t msg_id, bt_status_t status, void *buff)`.

```
bt_aws_disconnect(aws_handle);
```

At the Agent end, the state must be switched to INACTIVE if Agent wants to disconnect from Partner.

```
bt_aws_mce_set_state(aws_handle, BT_AWS_MCE_STATE_INACTIVE_AGENT);
```

4) When the user plays music or makes a call, Agent must send audio information (such as playing time, codec, or volume) to Partner. If the user presses a button on the Partner end, Partner sends key press information to Agent. Therefore, `bt_aws_mce_send_information()` should be called. Usually, the `urgent` argument in `bt_aws_mce_send_information()` is `true` if the type is A2DP or CALL.

```
//Agent
//get current bt clock
bt_get_bt_clock(gap_handle, &current_clock);
// Calculate the playing time.
// Fill the A2DP information into a2dp data(state, volume, playing time, //
media packet inforamtion, codec)
bt_aws_mce_information_t a2dp_infomation;
a2dp_infomation.type = BT_AWS_MCE_INFORMATION_A2DP;
a2dp_infomation.data_length = sizeof(a2dp_data);
a2dp_infomation.data = (uint8_t *)&a2dp_data;
bt_aws_mce_send_information(aws_handle, &a2dp_infomation, true);
```

```
// Set the audio playtime, and correspoding media packet.
a2dp_codec_handle->play(a2dp_audio_id);
//-------------------------------------------------------------------------
// Partner
// After receiving the notify ID BT_AWS_MCE_INFOMATION_PACKET_IND.
// Extract the A2DP information from the AWS_MCE information packet.
// Set the audio playtime, and correspoding media packet.
a2dp_codec_handle->play(a2dp_audio_id);
```

5) The role handover procedure can occur when the user takes the earbuds out of the charging case. Usually, there is some preparation before the handover (e.g., a check is performed to see if the connection establishment is complete or ongoing) to check if Bluetooth is receiving the signal related to control. If the related procedures are not ongoing, then the handover process can start. The role handover process must be initiated by Agent.

```
bt_aws_mce_send_information(aws_handle, partner_address);
```

```
bt_app_event_callback(bt_msg_type_t msg_id, bt_status_t status, void *buff)
{
switch (msg_id)
{
    case BT_AWS_MCE_PREPARE_ROLE_HANDOVER_CNF:
{
    if (status == BT_STATUS_SUCCESS){
        // Collect the important data passed to Partner. The data length
should be less than 100 Bytes.
        ret = bt_aws_mce_role_handover(aws_handle, data, length);
}
    break;
}
case BT_AWS_MCE_ROLE_HANDOVER_IND:
{
      // Partner end received the Role handover request.
      // Set the role as Agent, apply the Agent related data, for example,
      // Sink A2DP context, Call context, etc.
    break;
}
case BT_AWS_MCE_ROLE_HANDOVER_CNF:
{
      if (status == BT_STATUS_SUCCESS) {
    // Role handover success.
    // Clear the context related Agent role, example, sink A2DP, CALL.
    // And set the role as Partner, set related data.
} else {
    // Role handover failed.
}
    break;
}
case BT_AWS_MCE_STATE_CHANGED_IND:
{
    // State will be changed after Role handover.
      bt_aws_mce_state_change_ind_t *changed_data =
(bt_aws_mce_state_change_ind_t *)buff;
    break;
}
default:
    break;
```

```
    }
    }
```

3. The MCSync Share Mode

3.1. Introduction

MCSync Share allows two sets of earbuds to share the same music stream (only a2dp streaming) from one smartphone. A new role Follower are defined as the Finger 11, it does not connect to the smartphone directly, only sniffing the data stream between smartphone and the other set of earbuds. After Followers leave the MCSync Share scenario, they will change back to normal MCSync (Agent + Partner).

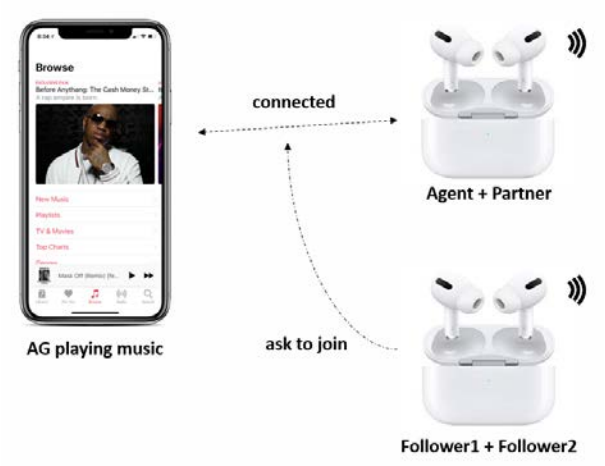

Figure 11. MCSync Share roles and link status

3.2. The state machine

There are four states for Agent, Partner, Follower1 and Follower2 as shown in Figure 12. The state change can be triggered by KEY or connection and disconnection event. In default the MCSync share are in *Normal state*, and Agent and Follower1 will change to *Preparing state* when user clicks the key to enable Share mode. The Agent will change to *Sharing state* when one of the followers connects and the followers will change to *Sharing state* when it connects agent. If user clicks the key in *Sharing state*, all devices change to *Leaving state*.

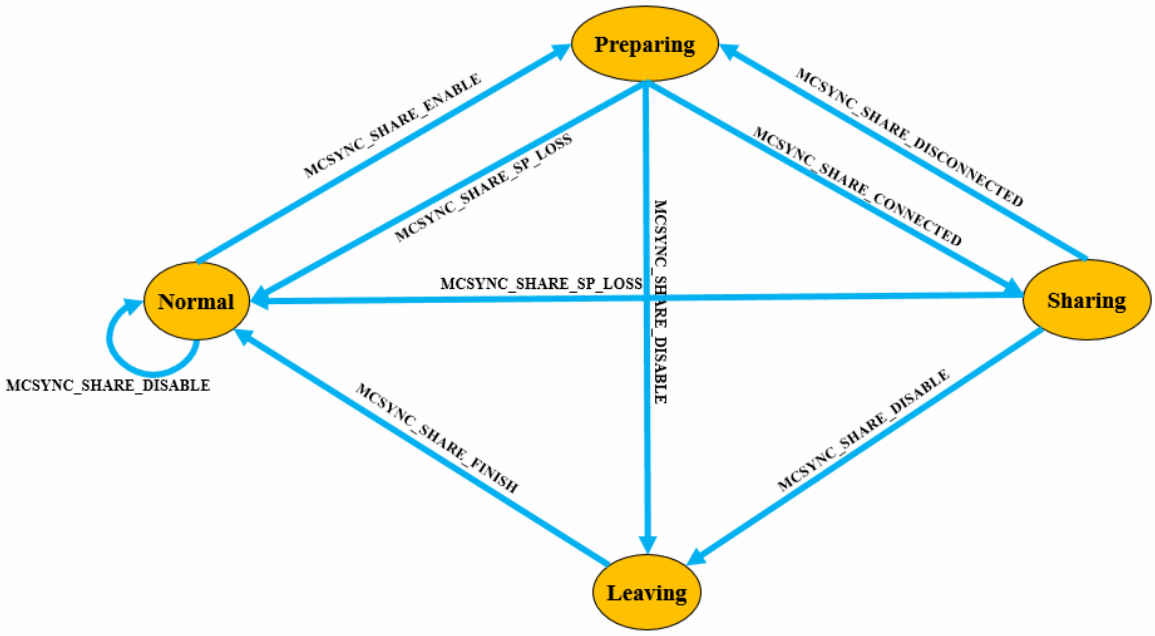

Figure 12. mcsync share state machine

3.3. The MCSync share message sequence

There are four operations in MCSync share: Share Pairing, connection establishment, disable Share mode and send information. The message sequence for each procedure is described in the following sections.

3.3.1. Share pairing

The purpose of Share pairing is to establish a connection between the agent and the followers. In Share pairing, the agent and the followers will find the address of each other. Agent and follower will send the address to Partner. For more details, refer to **bt_mcsync_share.h**

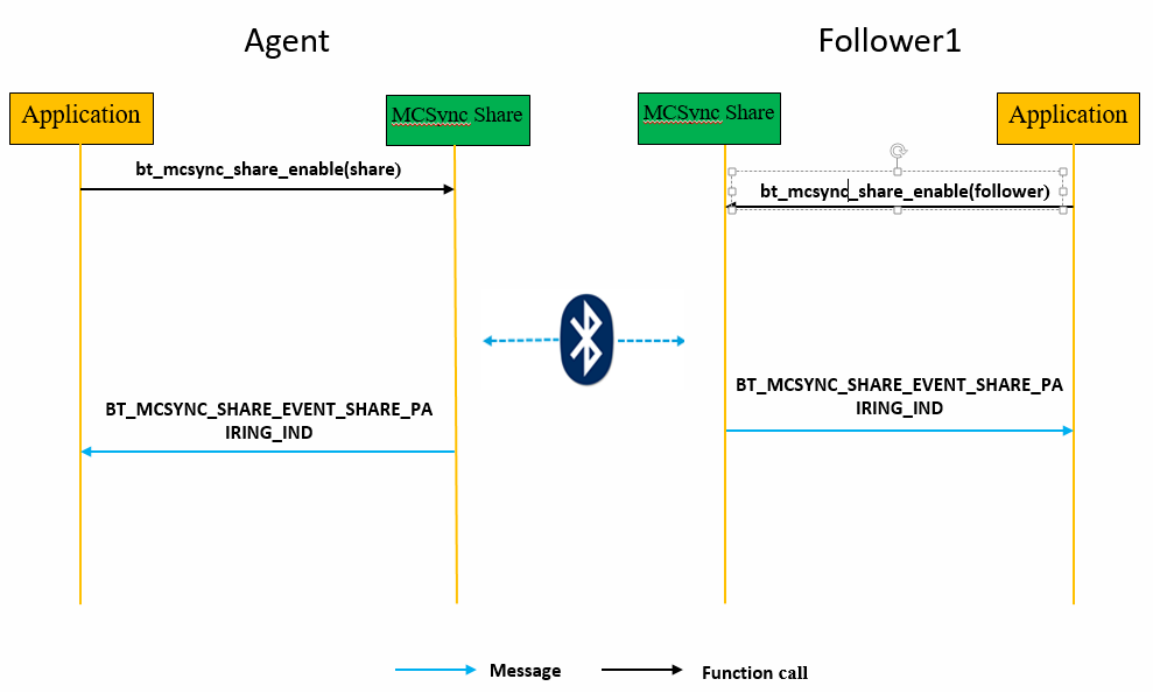

Figure 13. Message sequence when share pairing

3.3.2. Connection establishment

After share pairing the connection process will start. The agent and partner have the same behavior and the follower1 and follower2 have the same behavior. The partner will not do any things and it only update state from the agent.

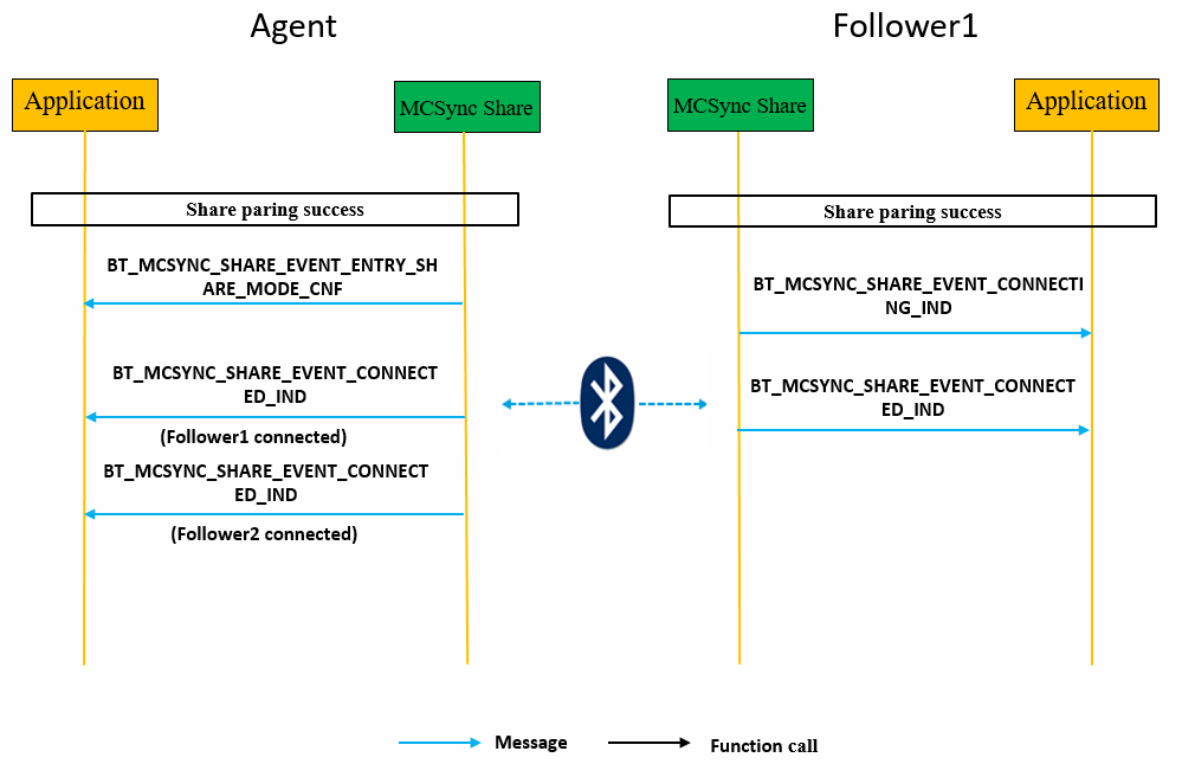

Figure 14. Message sequence when connection establish between agent and follower1/2

3.3.3. Disable Share mode

Device will leave the Share mode when user clicks the key as the app is configured by the API **bt_mcsync_share_init()**. The default behavior is *exit_all*, which means the four devices will exit Share mode at the same time. If the configuration is *self_only*, only the current device will exit Share mode, and the other devices will be kept in the Share mode, but if the device is agent, all devices will exit the Share mode. The follower will return to the previous state after the Share mode is exited.

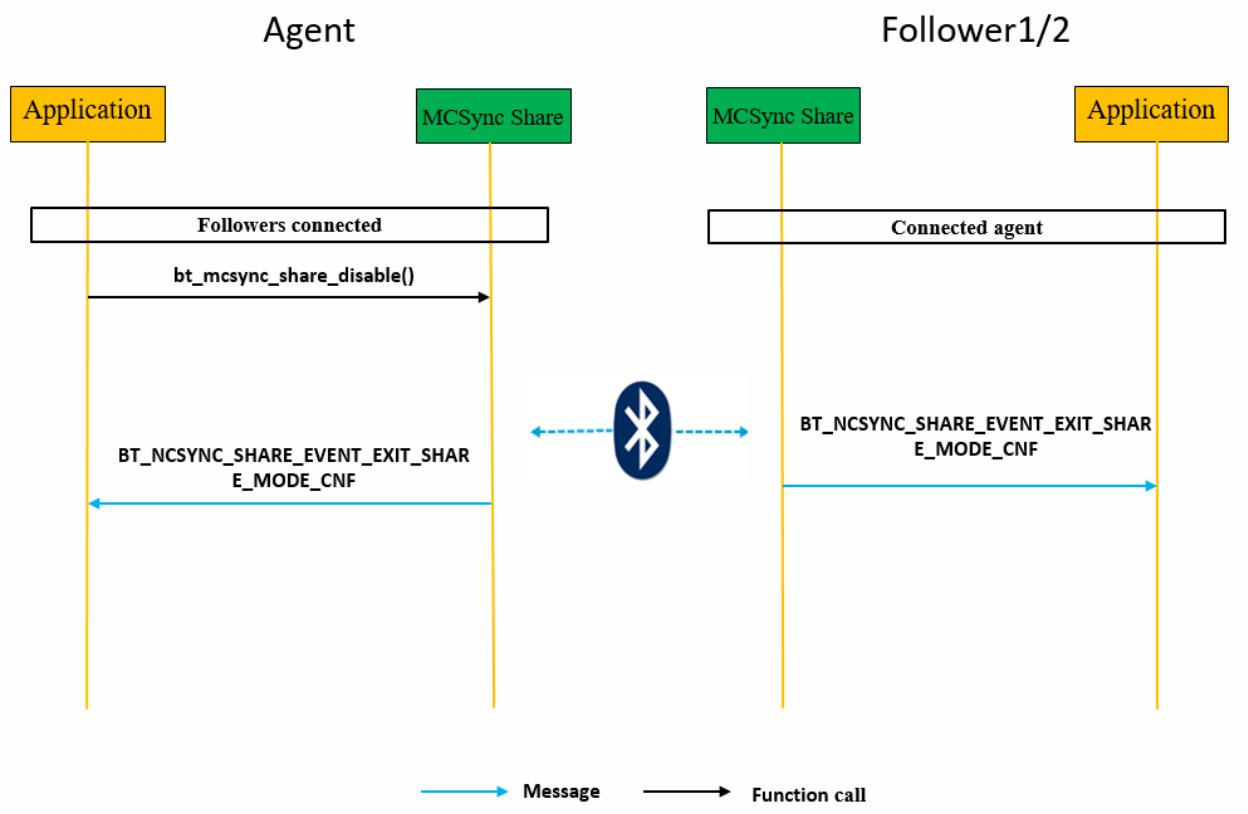

Figure 15. Message sequence when disable mode from agent

3.3.4. Send information

When Share mode is established, four devices can send messages to each other with the API bt_mcsync_share_send_action(). The message total size including the message header must be less than or equal to 4 bytes. On the partner side, the bt_aws_mce_report module cannot send messages in this state, but it can receive messages. The partner can also use bt_mcsync_share_send_action() to send message.

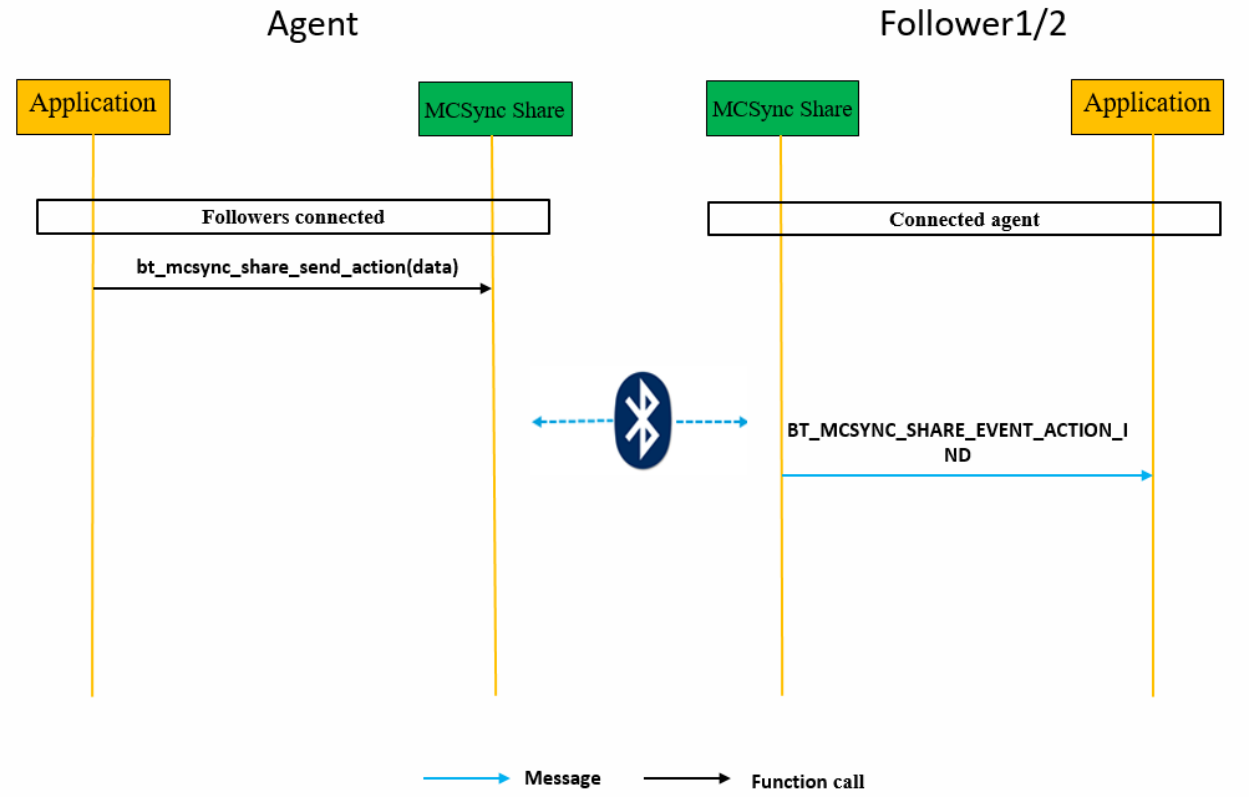

Figure 16. Message sequence send message after Share mode is established

3.4. Using the MCSync share APIs

This section describes how to use the MCSync share APIs for application development. The functionality of the MCSync APIs is implemented in the module bt_mcsync_share, related APIs can be found in `bt_mcsync_share.h`, the other header files are used internally, application cannot call them at any time.

1) Call the `bt_mcsync_share_init()` function to initialize the MCSync share module.

```
bt_mcsync_share_init(configure);
```

2) All device must register callback to receive message from bt_mcsync_share module.

```
bt_mcsync_share_register_callback(modile_id, callback);
```

```
bt_status_t bt_mcsync_share_event_callback(bt_mcsync_share_event_t event,
                                           bt_status_t status,
                                           void* data)
{
    switch(event){
        case BT_MCSYNC_SHARE_EVENT_SHARE_PAIRING_IND:
            if(status == BT_STATUS_SUCCESS){
                //do something
            }else{
                //
            }
        break;
        case BT_MCSYNC_SHARE_EVENT_ENTRY_SHARE_MODE_CNF:
```

```
            break;
            case BT_MCSYNC_SHARE_EVENT_DISCONNECTED_IND:
            break;
            case BT_MCSYNC_SHARE_EVENT_CONNECTING_IND:
            break;
           case BT_MCSYNC_SHARE_EVENT_CONNECTED_IND:
            break;
            case BT_NCSYNC_SHARE_EVENT_EXIT_SHARE_MODE_CNF:
            break;
            case BT_MCSYNC_SHARE_EVENT_ACTION_IND:
                bt_mcsync_share_action_info_t* info =
                                            (bt_mcsync_share_action_info_t*)data;
                if(info->action == xx){
                    //do some thing
                }
            break;
           default:
            break;

        }

    }
```

3) If user clicks the predefined key pattern, the app will enable the Share mode; if the key action is on partner, the app should send the action to the agent and then enable Share mode.

```
bt_status_t app_share_mode_key_action(bt_key_action_t key,
                                      bt_mcsync_share_role_t share_role)
  {
      switch(key){
          case BT_KEY_ACTION_SHARE_MODE_ENABLE:{
              bt_mcsync_share_role_t aws_role = bt_mcsync_share_get_role();
              if(aws_role == BT_MCSYNC_SHARE_ROLE_AGENT){
                  bt_mcsync_share_role_t tar_role =
                     share_role ==  BT_MCSYNC_SHARE_ROLE_SHARE ?
                                    BT_MCSYNC_SHARE_ROLE_SHARE:
                                    BT_MCSYNC_SHARE_ROLE_FOLLOWER;
                  bt_mcsync_share_enable(BT_MCSYNC_SHARE_ROLE_AGENT|tar_role);
              } else if(aws_role == BT_MCSYNC_SHARE_ROLE_PARTNER){
                  char packet[2] = {0};
                  packet[0] = 1; //enable share mode
                  packet[1] = share_role;
                  bt_aws_mce_report_info_t info;
                  info.module_id =  BT_AWS_MCE_REPORT_MODULE_APP_ACTION;
                  info.is_sync = false;
                  info.param_len = 2;
                  info.param = packet;
                  bt_aws_mce_report_send_event(&info);
              }
          }
          break;
          case  BT_KEY_ACTION_SHARE_MODE_DISABLE:
               bt_mcsync_share_disable();
          break;
          default:
          break;
      }
```

```
}
```