GR551x AMS Profile Example Application

Version: 1.6

Release Date: 2020-06-30

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

GOODiX and other Goodix trademarks are trademarks of Shenzhen Goodix Technology Co., Ltd. All other trademarks and trade names mentioned in this document are the property of their respective holders.

Disclaimer

Information contained in this document is intended for your convenience only and is subject to change without prior notice. It is your responsibility to ensure its application complies with technical specifications.

Shenzhen Goodix Technology Co., Ltd. (hereafter referred to as “Goodix”) makes no representation or guarantee for this information, express or implied, oral or written, statutory or otherwise, including but not limited to representation or guarantee for its application, quality, performance, merchantability or fitness for a particular purpose. Goodix shall assume no responsibility for this information and relevant consequences arising out of the use of such information.

Without written consent of Goodix, it is prohibited to use Goodix products as critical components in any life support system. Under the protection of Goodix intellectual property rights, no license may be transferred implicitly or by any other means.

Shenzhen Goodix Technology Co., Ltd.

Headquarters: 2F. & 13F., Tower B, Tengfei Industrial Building, Futian Free Trade Zone, Shenzhen, China

TEL: +86-755-33338828 FAX: +86-755-33338099

Website: www.goodix.com

Preface

Purpose

This document introduces how to use and verify an AMS Client example in GR551x SDK, to help users quickly get started with secondary development.

Audience

This document is intended for:

- GR551x user
- GR551x developer
- GR551x tester
- iOS engineer
- Hobbyist developer
- Technical writer

Release Notes

This document is the fourth release of *GR551x AMS Profile Example Application*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Modified the name of Track-Artist entity in “Section 3.6 Test and Verification”.
1.5	2020-05-30	Modified the code format in “Chapter 4 Application Details”.
1.6	2020-06-30	Updated the document version based on SDK changes.

Contents

1 Introduction

The Apple Media Service (AMS) is applied to intelligent Bluetooth-enabled devices such as wristbands and smart watches that connect to iOS devices. Through a Bluetooth Low Energy (Bluetooth LE) link, the Bluetooth devices can access media notifications from iOS devices and send AMS-related control commands to iOS devices.

This document introduces the approaches to implementing AMS Client based on a GR551x SoC.

Before getting started, it is recommended to refer to the following documents.

Table 1-1 Reference documents

Name	Description
Apple Media Service Reference	Offers Apple Media Service specification. Available at https://developer.apple.com/library/archive/documentation/CoreBluetooth/Reference/AppleMediaService_Reference/Specification/Specification.html#//apple_ref/doc/uid/TP40014716-CH1-SW7.
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
Bluetooth Core Spec v5.1	Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at https://www.bluetooth.com/specifications/bluetooth-core-specification/.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at www.bluetooth.com/specifications/gatt.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/.

2 Profile Overview

The AMS Profile defines two device roles:

- Server: iOS devices serve as the Central, providing services and data sources.
- Client: Bluetooth devices serve as the Peripheral capable of detecting services from iOS devices (the Central) as well as reading and writing data after being connected to an iOS device.

The interaction process between the Server and the Client is illustrated in Figure 2-1.

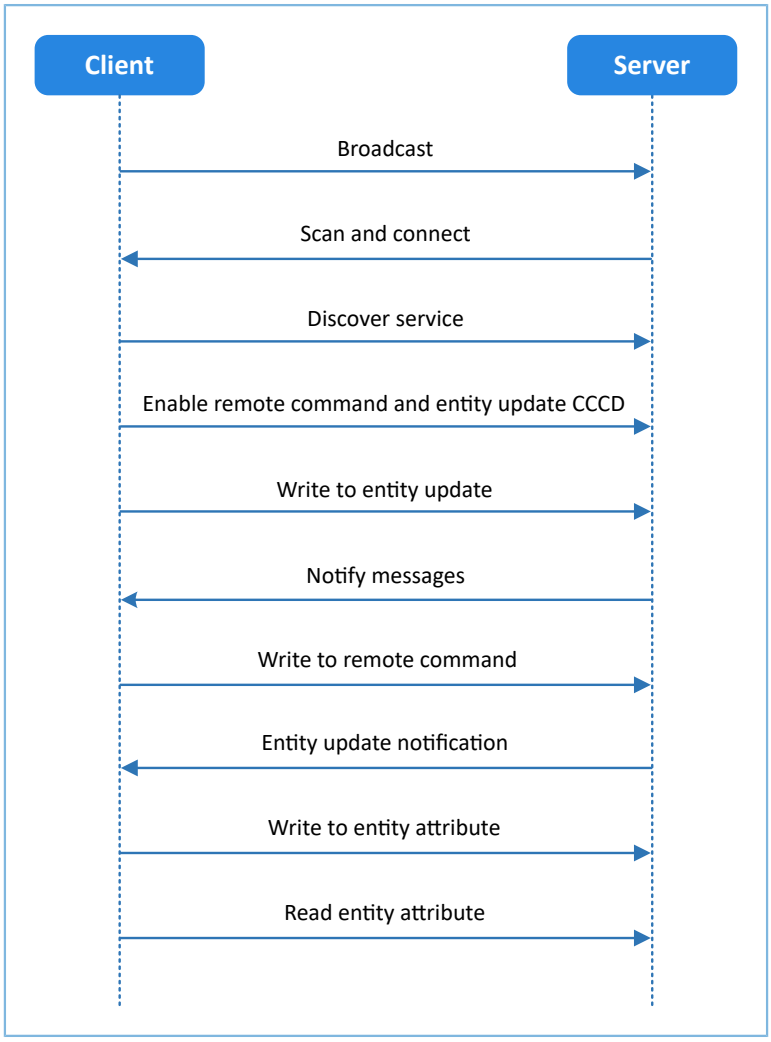

Figure 2-1 Server-Client interaction process

AMS characteristics include Remote Command, Entity Update, and Entity Attribute, as listed in Table 2-1.

Table 2-1 AMS characteristics

Characteristic	UUID	Type	Support	Security	Property
Remote Command	9B3C81D8-57B1-4A8A-B8DF0E56F7CA51C2	128 bits	Mandatory	None	Write and Notify
Entity Update	2F7CABCE-808D-411F-9A0C-BB92BA96C102	128 bits	Mandatory	None	Write and Notify

Characteristic	UUID	Type	Support	Security	Property
Entity Attribute	C6B2F38C-23AB-46D8-A6AB-A3A870BBD5D7	128 bits	Mandatory	None	Write and Read

The role of each AMS characteristic:

- Remote Command: Used to send remote commands and receive updates of available remote commands. The Write property means a Bluetooth device sends a remote command to an iOS device to implement remote control. The Notify property means when the available remote command of the iOS device changes, the iOS device notifies the Bluetooth device of the updated available remote command.
- Entity Update: Used to set a focused entity and receive updates from the focused entity. The Write property is used to set a focused entity at the Bluetooth device. The Notify property is used when the focused entity of an iOS device changes, the iOS device notifies a Bluetooth device of the updated value of the focused entity (the value may be incomplete due to maximum transmission unit/MTU limit).
- Entity Attribute: Used to set and read display entity. The Write property is used to set a display entity on a Bluetooth device, and the Read property is used to read the complete value of the display entity from an iOS device.

3 Initial Operation

This chapter introduces the preparations, connection approaches, and test methods for running an AMS Client example for the first time by using GR5515 Starter Kit Board (GR5515 SK Board) as an AMS Client and an iOS device as an AMS Server.

Note:

SDK_Folder is the root directory of GR551x SDK.

3.1 Preparation

Perform the following tasks before running an AMS Client example.

- **Hardware preparation**

Table 3-1 Hardware preparation

Name	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/
Development board	GR5515 Starter Kit Board (GR5515 SK Board)
Connection cable	Micro USB 2.0 serial cable
iOS device	Any iOS device supporting Bluetooth LE 4.0 and later versions, such as iPhone 4s and iPad 3

- **Software preparation**

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
GProgrammer (Windows)	A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`.
GRUart (Windows)	A GR551x serial port debugging tool. Available in `SDK_Folder\tools\GRUart`.

3.2 Hardware Connection

Connect the GR5515 SK Board to a PC with a Micro USB 2.0 cable.

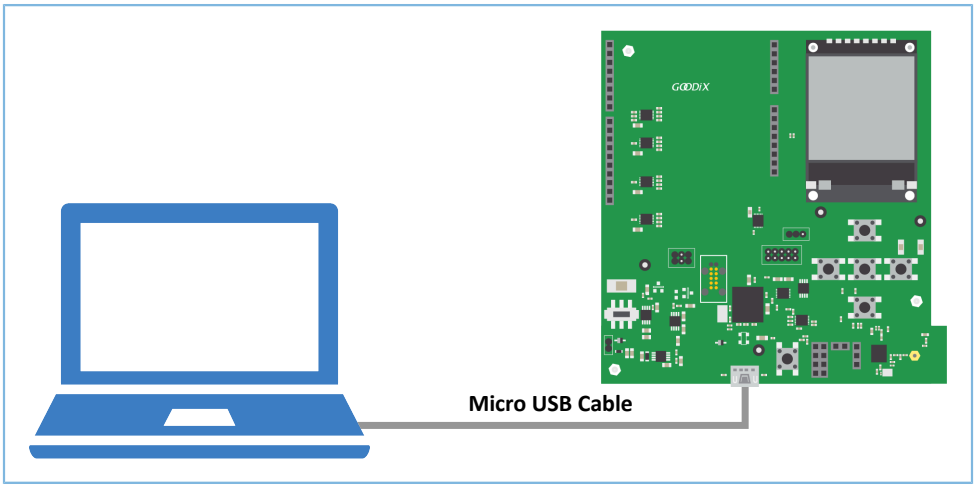

Figure 3-1 Hardware connection

3.3 Firmware Download

Download *ble_app_ams_c_fw.bin* firmware to the GR5515 SK Board by using GProgrammer. For details, see *GProgrammer User Manual*.

Note:

The *ble_app_ams_c_fw.bin* is in:

`SDK_Folder\projects\ble\ble_peripheral\ble_app_ams_c\build\`

3.4 Serial Port Settings

Start GRUart, and configure the serial ports according to the parameters in the table below.

Table 3-3 Configuring serial port parameters on GRUart

PortName	BaudRate	DataBits	Parity	StopBits	Flow Control
Select on demand	115200	8	None	1	Uncheck

When configuration is complete, click **Open Port**, as shown in the figure below.

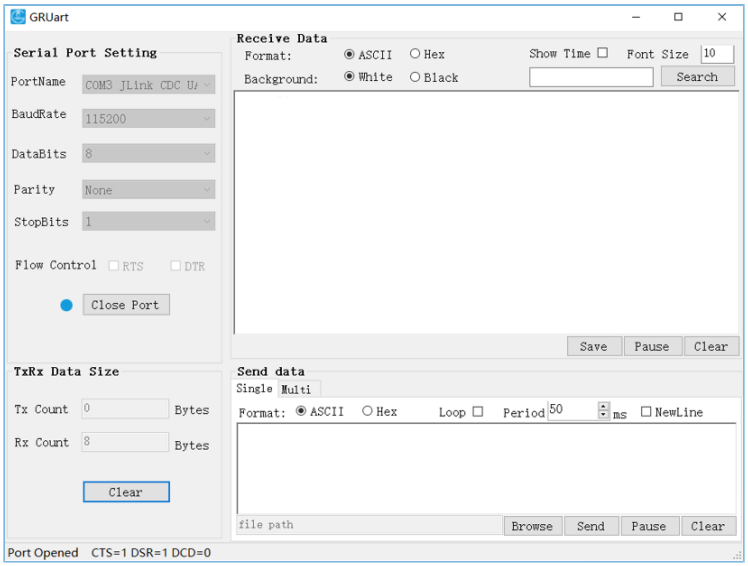

Figure 3-2 Serial port settings on GRUart

The GRUart displays no data in **Receive Data** and **Send Data** because no AMS notification is published.

3.5 Bluetooth Connection

Power the GR5515 SK Board on. Turn on Bluetooth on an iOS device to scan nearby Bluetooth devices. The device discovers a GR5515 SK Board with an advertising name of **Goodix_AMS_C**, as shown in Figure 3-3.

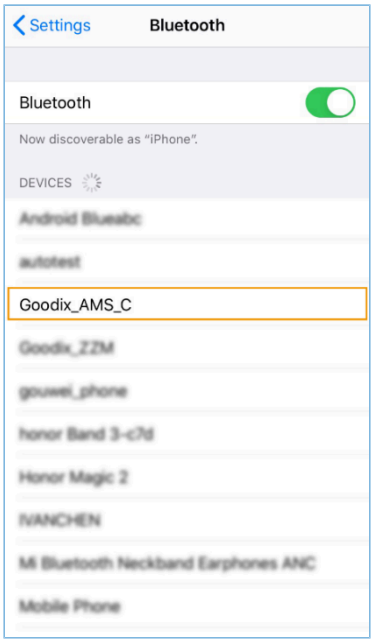

Figure 3-3 Discovering **Goodix_AMS_C**

Note:

This document is based on tests on an iPhone 6s running on iOS 12.4.1. The interface can be different depending on the device and operating system in use.

Tap **Goodix_AMS_C** to connect the device to the GR5515 SK Board. A pairing request dialog pops up, as shown in Figure 3-4. Enter **123456** in the dialog, and tap **Pair**. (For methods in setting a pairing password, see app_sec_rcv_enc_req_cb() function descriptions in "Section 4.2.1 Connection, Pairing, and Bonding")

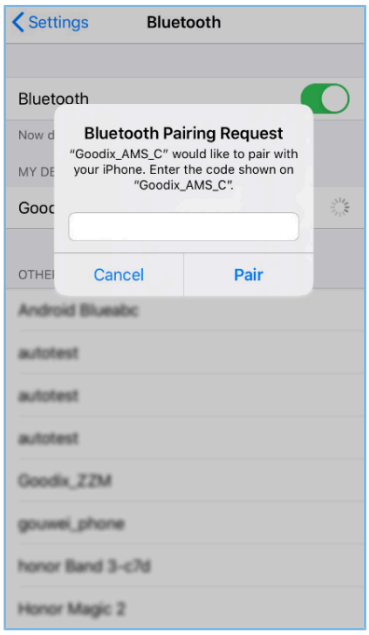

Figure 3-4 Entering pairing password

After pairing, **Goodix_AMS_C** displays as **Connected** under **MY DEVICES**.

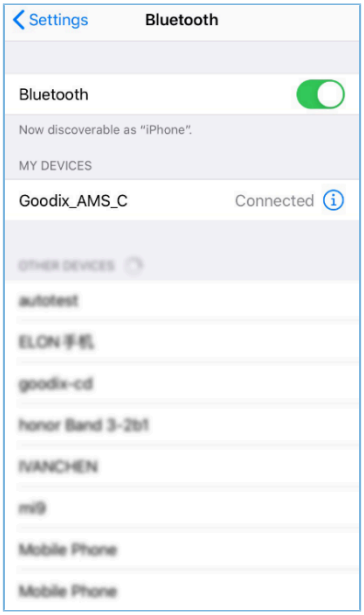

Figure 3-5 Successful pairing

After the iOS device is successfully connected to the GR5515 SK Board via Bluetooth, the connection information displays on GRUart serial port debugging tool, as shown in the figure below.

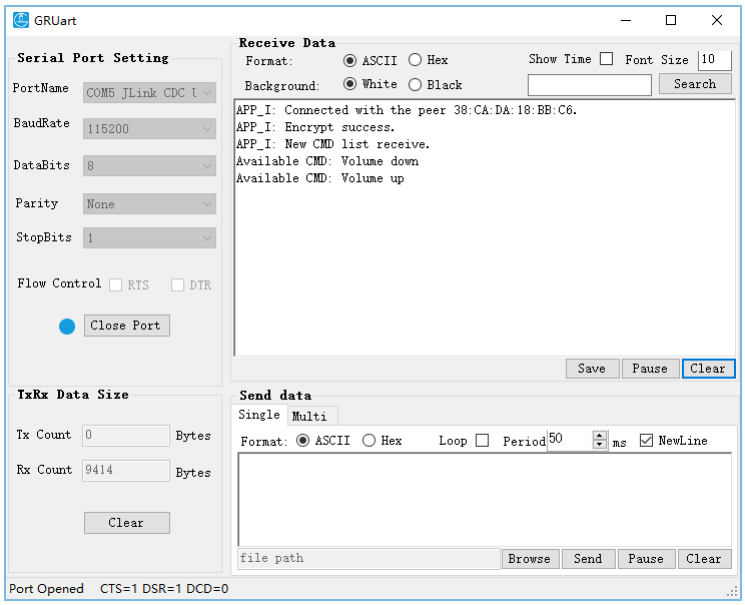

Figure 3-6 Serial port information on GRUart

3.6 Test and Verification

Perform AMS communications test on the iOS device and GR5515 SK Board when all the previous preparations are ready. This document aims to test AMS from two aspects:

- GR5515 SK Board accesses media notifications from the iOS device.
- GR5515 SK Board sends AMS control commands to the iOS device.

Users can verify the AMS based on serial port printing information on GRUart. For more information about AMS, see *Apple Media Service Reference*.

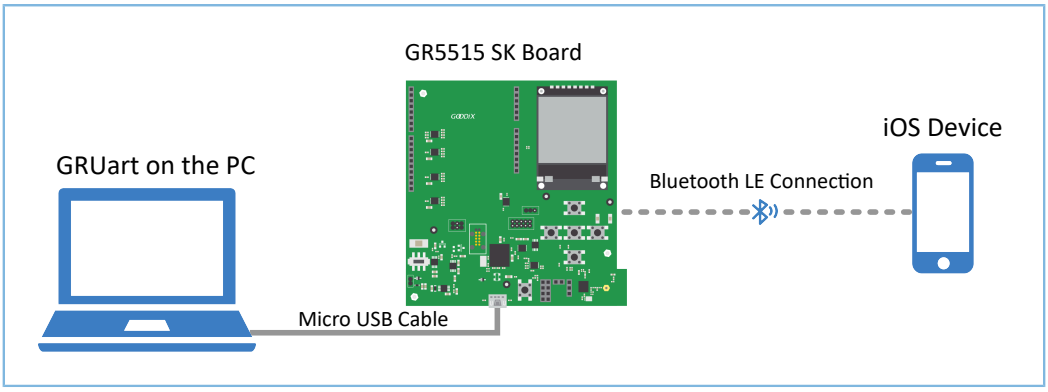

Figure 3-7 Hardware connection for AMS test scenarios

3.6.1 Access Media Notifications from an iOS Device

Follow the steps below to test how a GR5515 SK Board accesses media notifications from an iOS device:

1. Launch a music player App on the iOS device.
2. Play a track on the App. In the test, *The Sound Of Silence* (By Pat Metheny) is played with the App interface shown in Figure 3-8.

Figure 3-8 Playing music on the iOS device

3. View the printed information of serial port on GRUart on the PC.

The printed information (shown in Figure 3-9) comprises:

- List of remote commands supported by the music player App
- Update information of focused entities. For example, the GR5515 SK Board focuses on the media Track-Title and Track-Artist information of the iOS device, so any update on these entities may incur notifications. (For more details about focused entities, refer to *Apple Media Service Reference*.)

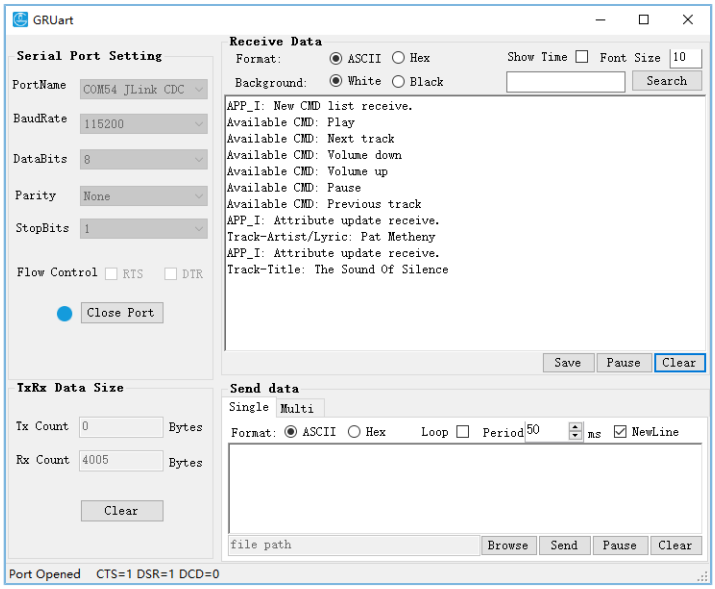

Figure 3-9 Printed information on GRUart

Description of printed information on GRUart is as follows:

Table 3-4 Description of information on GRUart

Name	Description
APP_I: New CMD list receive	Receive a new list of available remote commands.
Available CMD: Next track	Indicates the Next track command is available currently, and ready to send a Next track command.
APP_I: Attribute update receive	Receive the new value of the focused entity when the entity changes.
Track-Artist/Lyric: Pat Metheny	The Track-Artist entity changes with a new value of “Pat Metheny”.

Note:

According to *Apple Media Service Reference*, the ID of the Track-Artist entity corresponds to the artist of the track currently being played. However, most music player Apps display the lyrics through the ID. Therefore, GRUart displays the ID of the Track-Artist entity as Track-Artist/Lyric.

According to Figure 3-9, the AMS functions properly between the GR5515 SK Board and the iOS device.

3.6.2 Sending AMS Control Commands to an iOS Device

Follow the steps below to test how the GR5515 SK Board sends AMS control commands to the iOS device:

1. Launch a music player App on the iOS device.
2. Play a track on the App.
3. Press **RIGHT** on the GR5515 SK Board (to send a Next track command). In this test, *Deep in A Dream* (By Jim Hall) is used as the next track.

4. Check whether the music player App switches to the next track and the printed information on the GRUart.

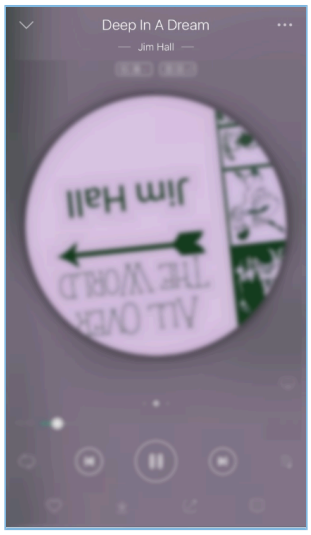

Figure 3-10 Playing music on the iOS device

The printed information (shown in Figure 3-11) comprises:

- Message indicating the remote commands have been successfully delivered
- Up-to-date track name and artist information (indicating the remote command has been executed)

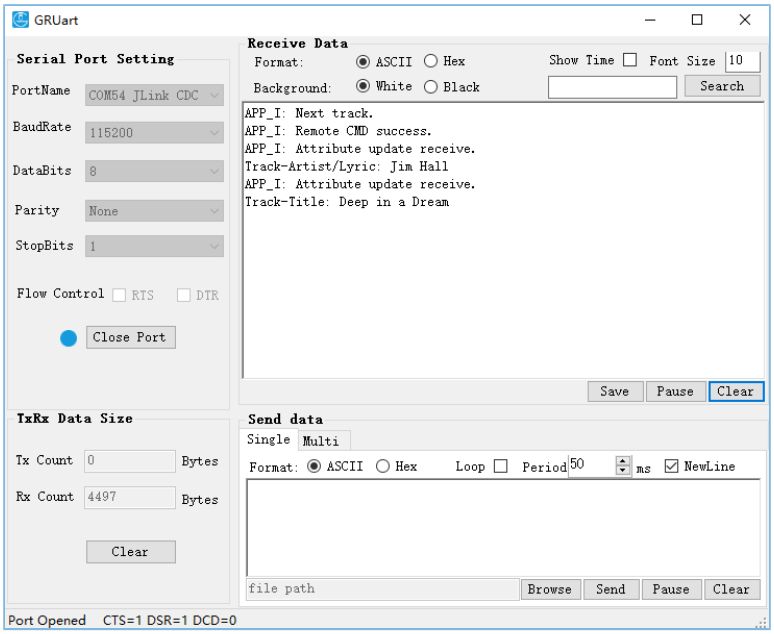

Figure 3-11 Printed information on GRUart

According to Figure 3-11, the AMS functions properly between the GR5515 SK Board and the iOS device.

Note:

For more information about buttons on a GR5515 SK Board, see "Chapter 7 Buttons and LEDs" in *GR5515 Starter Kit User Guide*.

Table 3-5 Button-command relations of GR5515 SK Board

Button	Command
OK (a short press)	Play
OK (two short presses)	Pause
OK (a long press)	Toggle play/pause
RIGHT	Next track
LEFT	Previous track
UP	Volume up
DOWN	Volume down

4 Application Details

This chapter elaborates on the interaction process and relevant code of the AMS Client example.

4.1 Project Directory

The source code and project file of the AMS Client example are in SDK_Folder\projects\ble\ble_peripheral\ble_app_ams_c, and the project file is in the Keil_5 folder.

Double-click the project file, *ble_app_ams_c.uvprojx*, to check the ble_app_ams_c project directory structure of the AMS Client example in Keil. Related files are described in Table 4-1.

Table 4-1 File description of ble_app_ams_c

Group	File	Description
gr_profiles	ams_c.c	This file helps perform AMS-related GATT operations.
user_callback	user_gap_callback.c	This file implements GAP Callback, such as connection, disconnection, and GAP parameter update.
	user_sm_callback.c	This file implements SM Callback, such as pairing and bonding.
user_platform	user_periph_setup.c	This file configures APP LOG, device address, and power management mode.
user_app	main.c	This file contains the main() function.
	user_app.c	This file sets advertising parameters of AMS Client and handle events.
	user_arms_decode.c	This file decodes input data of the peer device and outputs the decoded data.

4.2 Interaction Process and Major Code

Using AMS Client interaction process as an example, this section elaborates on the AMS Client pairing and bonding, AMS discovery, Client Characteristic Configuration Descriptor (CCCD) enablement, as well as notification handling, read, and write processes. Major code of each module is provided in the following sections to help users with better understanding.

The interaction process between an AMS Server and an AMS Client is illustrated in Figure 4-1.

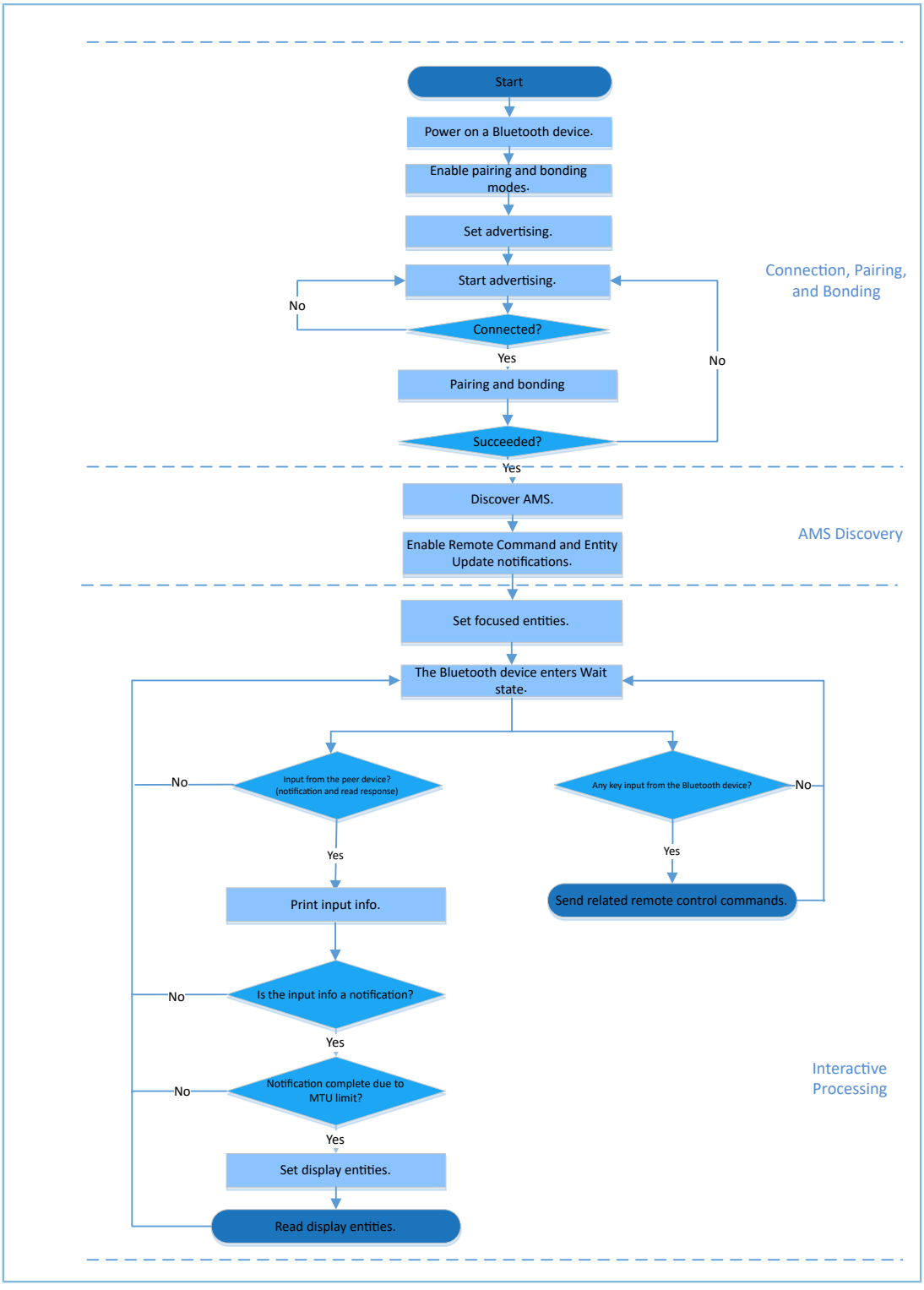

Figure 4-1 AMS Server-Client interaction process

4.2.1 Connection, Pairing, and Bonding

A Bluetooth device powered by a GR5515 SoC (GR5515 Bluetooth device) broadcasts as the Peripheral after being powered on. As the Central, an iOS device searches for nearby Bluetooth devices.

Turn on Bluetooth on the iOS device, and it connects to, pairs with, and bonds to the GR5515 Bluetooth device after scanning. Major code is as follows:

Path: `user_app/user_app.c` under AMS Client project directory

Name: gap_params_init();

The gap_params_init() function is used to enable pairing and bonding modes, as well as set security level (default: Security Mode Level 3; user-defined) and advertising parameters.

Note:

For more information about setting LE Security Modes Level, see “Section 10.2 LE security modes (Vol 3, Part C)” in *Bluetooth Core Spec v5.1*.

```
static void gap_params_init(void)
{
    sdk_err_t   error_code;

    ble_gap_pair_enable(true);

    error_code = ble_gap_privacy_params_set(150, true);
    APP_ERROR_CHECK(error_code);

    sec_param_t sec_param =
    {
        .level     = SEC_MODE1_LEVEL3,
        .io_cap    = IO_DISPLAY_ONLY,
        .oob       = false,
        .auth      = AUTH_BOND | AUTH_MITM | AUTH_SEC_CON,
        .key_size  = 16,
        .ikey_dist = KDIST_ENCKEY | KDIST_IDKEY,
        .rkey_dist = KDIST_ENCKEY | KDIST_IDKEY,
   };

    error_code = ble_sec_params_set(&sec_param);
    APP_ERROR_CHECK(error_code);

    s_gap_adv_param.adv_intv_max = APP_ADV_MAX_INTERVAL;
    s_gap_adv_param.adv_intv_min = APP_ADV_MIN_INTERVAL;
    s_gap_adv_param.adv_mode     = GAP_ADV_TYPE_ADV_IND;
    s_gap_adv_param.chnl_map     = GAP_ADV_CHANNEL_37_38_39;
    s_gap_adv_param.disc_mode    = GAP_DISC_MODE_NON_DISCOVERABLE;
    s_gap_adv_param.filter_pol   = GAP_ADV_ALLOW_SCAN_ANY_CON_ANY;

    error_code = ble_gap_device_name_set(BLE_GAP_WRITE_PERM_DISABLE,
                                         DEVICE_NAME, strlen(DEVICE_NAME));
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_param_set(0, BLE_GAP_OWN_ADDR_STATIC, &s_gap_adv_param);
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_DATA,
                                      s_adv_data_set, sizeof(s_adv_data_set));
    APP_ERROR_CHECK(error_code);

    error_code = ble_gap_adv_data_set(0, BLE_GAP_ADV_DATA_TYPE_SCAN_RSP,
                                      s_adv_rsp_data_set, sizeof(s_adv_rsp_data_set));
    APP_ERROR_CHECK(error_code);
```

```
    s_gap_adv_time_param.duration    = 0;
    s_gap_adv_time_param.max_adv_evt = 0;
}
```

Path: `user_callback/user_sm_callback.c` under AMS Client project directory

Name: app_sec_rcv_enc_req_cb();

Set the user-defined pairing password to **123456** in the callback function, and assign the value to cfm_enc.data.tk.key[4] array. The code is as displayed below:

```
static void app_sec_rcv_enc_req_cb(uint8_t conn_idx, sec_enc_req_t *p_enc_req)
{
    ...
    switch (p_enc_req->req_type)
    {
      ...
        // user need to input the password
        case TK_REQ:
            cfm_enc.req_type = TK_REQ;
            cfm_enc.accept = true;
            tk = 123456;
            memset(cfm_enc.data.tk.key, 0, 16);
            cfm_enc.data.tk.key[0] = (uint8_t)((tk & 0x000000FF) >> 0);
            cfm_enc.data.tk.key[1] = (uint8_t)((tk & 0x0000FF00) >> 8);
            cfm_enc.data.tk.key[2] = (uint8_t)((tk & 0x00FF0000) >> 16);
            cfm_enc.data.tk.key[3] = (uint8_t)((tk & 0xFF000000) >> 24);
            break;
        default:
            break;
    }
    ble_sec_enc_cfm(conn_idx, &cfm_enc);
}
```

Path: `user_callback/user_sm_callback.c` under AMS Client project directory

Name: app_sec_rcv_enc_ind_cb();

This function is executed after the GR5515 Bluetooth device is paired with the iOS device. Upon pairing, the system directly calls the ams_c_disc_srvc_start() function to discover services.

```
static void app_sec_rcv_enc_ind_cb(uint8_t conn_idx, sec_enc_ind_t enc_ind, uint8_t auth)
{
    sdk_err_t error_code;
    if (ENC_SUCCESS ! = enc_ind)
    {
        return;
    }
    APP_LOG_INFO("Encrypt success." );

    error_code = ams_c_disc_srvc_start(conn_idx);
    APP_ERROR_CHECK(error_code);
}
```

4.2.2 Discover AMS

The GR5515 Bluetooth device sends a request to discover AMS on the iOS device after pairing with and connecting to the iOS device.

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_disc_srvc_start();

This function searches for and discovers a service on the peer device based on a specified UUID. The scanning results are returned through the callback function, ams_c_srvc_browse_cb() in *ams_c.c*.

```
sdk_err_t ams_c_disc_srvc_start(uint8_t conn_idx)
{
    ble_uuid_t ble_ams_uuid =
    {
        .uuid_len = BLE_ATT_UUID_128_LEN,
        .uuid     = (uint8_t *)ams_service_uuid,
    };
    return ble_gattc_prf_services_browse(s_ams_c_env.prf_id, conn_idx, &ble_ams_uuid);
}
```

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_srvc_browse_cb();

This function is used to access AMS from the iOS device and enumerate corresponding handles of each service attribute by using UUID.

```
static void ams_c_srvc_browse_cb(uint8_t conn_idx, uint8_t status, const
                                 ble_gattc_browse_srvc_t *p_browse_srvc)
{
    ams_c_evt_t  ams_c_evt;
    uint16_t     handle_disc;

    ams_c_evt.conn_idx = conn_idx;
    ams_c_evt.evt_type = AMS_C_EVT_DISCOVERY_FAIL;

    if(BLE_GATT_ERR_BROWSE_NO_ANY_MORE == status)
    {
        return;
    }
    if (BLE_SUCCESS == status)
    {
        if (0 == memcmp(p_browse_srvc->uuid, ams_service_uuid, BLE_ATT_UUID_128_LEN))
        {
            s_ams_c_env.handles.ams_srvc_start_handle = p_browse_srvc->start_hdl;
            s_ams_c_env.handles.ams_srvc_end_handle   = p_browse_srvc->end_hdl;
            for (uint32_t i = 0; i < p_browse_srvc->end_hdl -
                 p_browse_srvc->start_hdl; i++)
            {
                if (BLE_GATTC_BROWSE_ATTR_VAL == p_browse_srvc->info[i].attr_type)
                {
                    handle_disc = p_browse_srvc->start_hdl + i + 1;
                   if (0 == memcmp(p_browse_srvc->info[i].attr.uuid, ams_cmd_uuid,
                        BLE_ATT_UUID_128_LEN))
                    {
                        s_ams_c_env.handles.ams_cmd_handle = handle_disc;
                        s_ams_c_env.handles.ams_cmd_cccd_handle = handle_disc + 2;
                    }
                   else if (0 == memcmp(p_browse_srvc->info[i].attr.uuid,
                         ams_attr_update_uuid, BLE_ATT_UUID_128_LEN))

                    {
                        s_ams_c_env.handles.ams_attr_update_handle = handle_disc;
                        s_ams_c_env.handles.ams_attr_update_cccd_handle = handle_disc + 2;
                    }
                    else if (0 == memcmp(p_browse_srvc->info[i].attr.uuid,
```

```
                            ams_attr_display_uuid, BLE_ATT_UUID_128_LEN))
                    {
                        s_ams_c_env.handles.ams_attr_display_handle = handle_disc;
                    }
                }
                else if (p_browse_srvc->info[i].attr_type == BLE_GATTC_BROWSE_NONE)
                {
                    break;
                }
            }
            ams_c_evt.evt_type = AMS_C_EVT_DISCOVERY_CPLT;
        }
    }
    ams_c_evt_handler_excute(&ams_c_evt);
}
```

After discovering the service, enable the notification function of each characteristic respectively. For details, see "Section 4.2.3 Notification Event and Interpretation".

4.2.3 Notification Event and Interpretation

Remote Command and Entity Update support notification function. If publishing notifications is required, set the CCCD of the characteristic to enable notification, and ensure the CCCD handle has been founded during service discovery. Major code is as follows:

Path 1: `gr_profiles/ams_c.c` under AMS Client project directory

Name 1: ams_c_cmd_notify_set();

Path 2: `gr_profiles/ams_c.c` under AMS Client project directory

Name 2: ams_c_attr_update_notify_set();

The ams_c_cmd_notify_set() function is used to enable notification of Remote Command. The ams_c_attr_update_notify_set() function is used to enable notification of Entity Update. Code of the ams_c_attr_update_notify_set() function is similar to that of the ams_c_cmd_notify_set() function. Detailed code of the ams_c_cmd_notify_set() function is displayed below

```
sdk_err_t ams_c_cmd_notify_set(uint8_t conn_idx, bool is_enable)
{
    gattc_write_attr_value_t write_attr_value;
    uint16_t ntf_value = is_enable ? PRF_CLI_START_NTF : PRF_CLI_STOP_NTFIND;

    if (BLE_ATT_INVALID_HDL == s_ams_c_env.handles.ams_cmd_cccd_handle)
    {
        return BLE_ATT_ERR_INVALID_HANDLE;
    }
    write_attr_value.handle  = s_ams_c_env.handles.ams_cmd_cccd_handle;
    write_attr_value.offset  = 0;
    write_attr_value.length  = 2;
    write_attr_value.p_value = (uint8_t *)&ntf_value;

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_att_ntf_ind_cb();

This function receives notifications of Remote Command and Entity Update by using corresponding data structures, as shown in the code below.

```
static void ams_c_att_ntf_ind_cb(uint8_t conn_idx, const ble_gattc_ntf_ind_t *p_ntf_ind)
{
    ams_c_evt_t ams_c_evt;
    ams_c_evt.conn_idx = conn_idx;
    ams_c_evt.evt_type = AMS_C_EVT_INVALID;
    if (p_ntf_ind->handle == s_ams_c_env.handles.ams_cmd_handle)
    {
        ams_c_evt.evt_type = AMS_C_EVT_CMD_UPDATE_RECEIVE;
        ams_c_evt.param.cmd_list.p_cmd = p_ntf_ind->p_value;
        ams_c_evt.param.cmd_list.length = p_ntf_ind->length;
    }
    else if (p_ntf_ind->handle == s_ams_c_env.handles.ams_attr_update_handle)
    {
        ams_c_evt.evt_type = AMS_C_EVT_ATTR_UPDATE_RECEIVE;
        ams_c_attr_info_decode(p_ntf_ind->p_value, p_ntf_ind->length,
                               &ams_c_evt.param.attr_info);
    }
    ams_c_evt_handler_excute(&ams_c_evt);
}
```

4.2.4 Read/Write Interaction

GR5515 Bluetooth devices write data to and read data from the three characteristics of AMS, which enables remote interaction between the GR5515 Bluetooth device and the iOS device. This section elaborates on the read and write interactions supported by AMS.

4.2.4.1 Remote Command

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_cmd_send();

This function is used to implement remote control by writing command IDs to Remote Command.

```
sdk_err_t ams_c_cmd_send(uint8_t conn_idx, uint8_t cmd_id)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_cmd_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = 1;
    write_attr_value.p_value = (uint8_t *)&cmd_id;

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

Path: `gr_profiles/ams_c.h` under AMS Client project directory

Name: ams_c_cmd_id_t;

This structure defines the enumeration values of all the remote command IDs.

```
typedef enum
{
    AMS_CMD_ID_PLAY,                        /**< Command index of play. */
    AMS_CMD_ID_PAUSE,                       /**< Command index of pause. */
```

```
    AMS_CMD_ID_TOGGLE_PLAY_PAUSE,          /**< Command index of toggle. */
    AMS_CMD_ID_NEXT_TRACK,                 /**< Command index of next track. */
    AMS_CMD_ID_PREVIOUS_TRACK,             /**< Command index of previous track. */
    ...
} ams_c_cm
```

4.2.4.2 Set a Focused Entity

Path: `user_app/user_app.c` under AMS Client project directory

Name: attr_focus_set();

This function sets focused entities by writing entity IDs to Entity Update (an Entity Update command comprises Entity ID and Attribute ID; for details, see *Apple Media Service Reference*). When entity setting completes, the Entity Update publishes notifications if the set entity changes.

In addition, this function defines structures as containers of entity IDs and calls ams_c_attr_focus_set() to write data to Entity Attribute. Users can set focused entities on demand.

```
static void attr_focus_set(uint8_t conn_idx)
{
    sdk_err_t   error_code;
    ams_c_ett_attr_id_t track_attr_id =
    {
        .ett_id     = AMS_TRACK_ID,
        .attr_id    = {AMS_TRACK_ARTIST_ID, AMS_TRACK_TITTLE_ID},
        .attr_count = 2
    };
    error_code = ams_c_attr_focus_set(conn_idx, &track_attr_id);
    APP_ERROR_CHECK(error_code);
}
```

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_attr_focus_set();

This function is used to write data to Entity Attribute.

```
sdk_err_t ams_c_attr_focus_set(uint8_t conn_idx, const ams_c_ett_attr_id_t *p_ett_attr_id)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_attr_update_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = p_ett_attr_id->attr_count + 1;
    write_attr_value.p_value = (uint8_t *)&(p_ett_attr_id->ett_id);

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

4.2.4.3 Set a Display Entity

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_attr_display_set();

Limited by MTU, the entity data notified by Entity Update may be truncated. Entity Attribute should be used to obtain complete entity data.

The ams_c_attr_display_set() function is used to set a display entity by writing entity IDs to Entity Attribute. After successful writing, complete entity data can be accessed by reading the Entity Attribute value.

```
sdk_err_t ams_c_attr_display_set(uint8_t conn_idx, const ams_c_attr_info_t *p_attr_info)
{
    ...
    gattc_write_attr_value_t write_attr_value;
    write_attr_value.handle = s_ams_c_env.handles.ams_attr_display_handle;
    write_attr_value.offset = 0;
    write_attr_value.length = 2;
    write_attr_value.p_value = (uint8_t *)&(p_attr_info->ett_id);

    return ble_gattc_prf_write(s_ams_c_env.prf_id, conn_idx, &write_attr_value);
}
```

4.2.4.4 Read a Display Entity

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_cplt_attr_read();

The function is used to set a display entity, read the entity, and access the complete value of the display entity.

```
sdk_err_t ams_c_cplt_attr_read(uint8_t conn_idx)
{
    if (BLE_ATT_INVALID_HDL == s_ams_c_env.handles.ams_attr_display_handle)
    {
        return BLE_ATT_ERR_INVALID_HANDLE;
    }
    return ble_gattc_prf_read(s_ams_c_env.prf_id, conn_idx,
                              s_ams_c_env.handles.ams_attr_display_handle, 0);
}
```

Path: `gr_profiles/ams_c.c` under AMS Client project directory

Name: ams_c_att_read_cb();

The function implements receiving data in a corresponding data structure.

```
static void ams_c_att_read_cb(uint8_t conn_idx, uint8_t status,
                              const ble_gattc_read_rsp_t *p_read_rsp)
{
    ...
    if (p_read_rsp->vals[0].handle == s_ams_c_env.handles.ams_attr_display_handle)
    {
        ams_c_evt.conn_idx = conn_idx;
        ams_c_evt.evt_type = AMS_C_EVT_CPLT_ATTR_READ_RSP;
        ams_c_evt.param.cplt_attr_data.p_data = p_read_rsp->vals[0].p_value;
        ams_c_evt.param.cplt_attr_data.length = p_read_rsp->vals[0].length;
    }

    ams_c_evt_handler_excute(&ams_c_evt);
}
```

5 FAQs

This chapter describes possible problems, reasons, and solutions during verification and application of the AMS Client example.

5.1 Why Is there No Output Information from GRUart?

- Description

  No printed information displays on GRUart, or GRUart encounters garbled printing.

- Analysis

  The *ble_app_ams_c_fw.bin* firmware is not programmed on the board correctly, or the **BaudRate** on the GRUart is incorrect, resulting in GRUart's failure to print information.

- Solution

  1. Confirm on GRUart, the **BaudRate** is 115200 with **DataBits** of 8, **StopBits** of 1, None **Parity**, and no **Flow Control**. Confirm the serial port cable has been correctly connected.
  2. If there is nothing wrong with the serial port connection, redo the firmware programming, and ensure no code modification has been done on the project, then directly download the firmware to the Bluetooth device using GProgrammer.

5.2 Why does an iOS Device Fail to Scan Any Bluetooth Advertising from Goodix_AMS_C?

- Description

  An iOS device with Bluetooth enabled fails to find advertising from Goodix_AMS_C.

- Analysis

  Exceptions occur in the Bluetooth antenna connection or firmware.

- Solution

  1. Check whether the device is truly powered by iOS operating system and the Bluetooth function has been turned on. If the iOS device still cannot find Goodix_AMS_C when the Bluetooth function is turned on, check the antennae of the GR5515 Bluetooth device.
  2. If both the Bluetooth function on the iOS device and the antennae of the GR5515 Bluetooth device operate properly, check hardware problems by downloading a template firmware, *ble_app_template_fw.bin*, to the GR5515 Bluetooth device. The *ble_app_template_fw.bin* is in SDK_`Folder\projects\ble\ble_peripheral\ble_app_template\build\`. After downloading, run the firmware. If the hardware functions properly, the iOS device can scan the advertising from Goodix_Tem; otherwise, exceptions occur in the Bluetooth antenna connection or firmware.

5.3 Why does an iOS Device Fail to Connect to Goodix_AMS_C Bluetooth Requests?

- Description

  An iOS device cannot connect to Goodix_AMS_C even though it displays under **MY DEVICES**.

- Analysis

  The iOS device has been connected and bonded to GR5515 Bluetooth devices, but the data on the Bluetooth device has been erased or overwritten, leading to bonding failure due to loss of bonding information.

- Solution

  1. From **MY DEVICES**, tap **Goodix_AMS_C** to **Forget This Device**.
  2. Redo the Bluetooth scanning, pairing, and bonding on the iOS device.