



# **RT58x BLE SDK**

## **Application Guide**

### **V1.3**

## **About this Document**

This document supports at least “**Rafael RT58x BLE SDK v1.1.0**”.

## **Table of Contents**

About this Document .....	1
1. Introduction .....	3
2. BLE Peripheral Demo Application .....	4
2.1 HRS_Peripheral .....	4
2.1.1 BLE Services .....	4
2.1.2 Handle HRS Data Transmitted Over BLE .....	5
2.1.3 Running “HRS_Peripheral” Demo .....	7
2.2 TRSP_Peripheral .....	8
2.2.1 BLE Services .....	9
2.2.2 Handle Data Received Over UART .....	10
2.2.3 Handle Data Transmitted Over BLE .....	11
2.2.4 Handle Data Received Over BLE .....	12
2.2.5 Handle “Read Request” from client .....	13
2.2.6 Running “TRSP_Peripheral” Demo .....	15
2.3 HOGP_Peripheral .....	19
2.3.1 BLE Services .....	19
2.3.2 Handle the IO Capability Setting .....	20
2.3.3 Handle the HID Data Transmitted .....	24
2.3.4 Running “HOGP_Peripheral” Demo .....	28
2.3.5 Running “HOGP_Peripheral_Privacy” Demo .....	30
2.4 Data_Rate_Peripheral .....	30
2.4.1 BLE Services .....	31

2.4.2	Handle Data Rate Command Received Over BLE .....	32
2.4.3	Running “Data_Rate_Peripheral” Demo with Rafeal BLE APP .....	39
2.4.4	Running “Data_Rate_Peripheral” Demo with “Data_Rate_Central” .....	42
3.	BLE Central Demo Application.....	43
3.1	TRSP_Central.....	43
3.1.1	BLE Services .....	44
3.1.2	Handle Data Received Over UART .....	46
3.1.3	Handle Data Transmitted Over BLE .....	46
3.1.4	Handle Data Received Over BLE .....	47
3.1.5	Running “TRSP_Central” Demo .....	48
3.2	Data_Rate_Central .....	51
3.2.1	BLE Services .....	51
3.2.2	Handle Data Rate Command Received Over BLE .....	52
3.2.3	Running “Data_Rate_Central” Demo .....	54
4.	BLE Central and Peripheral Demo Application.....	56
4.1	TRSP_1C1P.....	57
4.1.1	BLE Services .....	57
4.1.2	Command List.....	59
4.1.3	Running “TRSP_1C1P” Demo .....	60
5.	iBeacon Demo Application.....	63
5.1.1	Running “iBeacon” Demo .....	63
6.	AT Command Demo Application .....	64
6.1	AT Command List .....	65
6.1.1	Common Command.....	65
6.1.2	Advertising Command.....	76
6.1.3	Scan Command .....	82
6.1.4	Create Connection Command .....	86
6.1.5	Connection Command.....	89
	Revision History .....	102

## 1. Introduction

The RT58x is built around the 32-bit ARM® Cortex™-M3 CPU with clock running up to 64 MHz. And it is capable of running multi-protocol concurrently. It can support protocols including Bluetooth LE, Bluetooth mesh, ZigBee, 802.15.4, and 2.4GHz proprietary.

This document is mainly about the instructions related to the Bluetooth LE example demonstration. The content includes peripheral roles (HRS/TRSP/ HOGP/Data\_Rate\_P), central roles (TRSP/Data\_Rate\_C), and BLE multi-link (1C1P).

(intentionally blank)

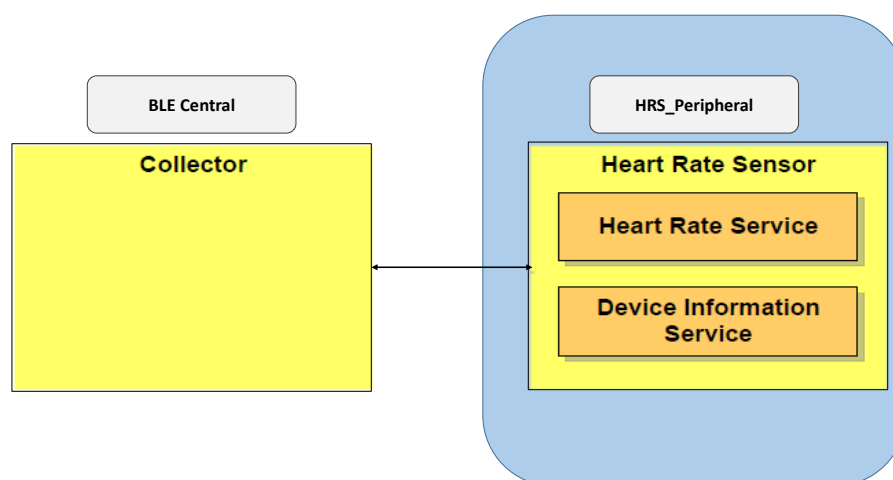
## 2. BLE Peripheral Demo Application

There are multiple BLE peripheral demo projects for reference, which are HRS\_Peripheral, TRSP\_UART\_Peripheral, HOGP\_Peripheral and Data\_Rate\_Peripheral.

***It is important to note here that most of the application demo cases are done in power saving mode. If you need to use UART input command, you need to send "Break" signal to 2ms to wake up and then execute the command.***

### 2.1 HRS\_Peripheral

The Heart Rate Service (HRS) application demonstrates how heart rate sensor data transmitted via standard BLE services. In this application demo, the device is connected to the phone and simply turns on the Heart Rate Service notification to get the heart rate value transmitted by the app device.



#### 2.1.1 BLE Services

According to the BLE heart rate profile specification "[HRP\\_V10.pdf](#)", it consists of at least three services, GAP, DIS (Device Information Service) and HRS (Heart Rate Service).

##### ● GAP

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification

“[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C\*](#)” for more details.

- **Device Information Service (DIS)**

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification “[\*DIS\\_SPEC\\_V11r00.pdf\*](#)” for more details. The implementation please refer to “ble\_service\_dis.c/ ble\_service\_dis.h”.

- **Heart Rate Service (HRS)**

Heart Rate Service (HRS) is a SIG defined service and please refer to SIG Bluetooth specification “[\*HRS\\_SPEC\\_V10.pdf\*](#)” for more details.

### 2.1.2 Handle HRS Data Transmitted Over BLE

Create a FreeRTOS software timer with a period of 1s to send heart rate measurement requests to the application. The application will send the heart rate measurement data to the client by using “ble\_cmd\_gatt\_notification” function.

```
static void hrs_timer_handler( TimerHandle_t timer)
{
    /* Optionally do something if the pxTimer parameter is NULL. */
    configASSERT( timer );

    // HRS
    if (ble_app_link_info[APP_HRS_P_HOST_ID].state == STATE_CONNECTED)
    {
        // send HRS data
        app_request_set(APP_HRS_P_HOST_ID, APP_REQUEST_HRS_NTF, false);
    }
}
```

(intentionally blank)

```
// HRS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ble_err_t status;
    uint8_t host_id;
    ble_profile_info_t *p_profile_info;

    host_id = p_param->host_id;
    p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    switch (p_param->app_req)
    {
        case APP_REQUEST_ADV_START:
            ...
            break;

        case APP_REQUEST_HRS_NTF:
        {
            ble_gatt_data_param_t gatt_param;

            // send heart rate measurement value to client
            if ((p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[0] & BIT1) == 0) //initial is 0x14 & 0x02 = 0, toggle "device detected"
            // "device not detected" information
            {
                p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[0] |= BIT1; //set Sensor Contact Status bit
            }
            else
            {
                p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[0] &= ~BIT1; //clear Sensor Contact Status bit
            }

            // set parameters
            gatt_param.host_id = host_id;
            gatt_param.handle_num = p_profile_info->svcs_info_hrs.server_info.handles.hdl_heart_rate_measurement;
            gatt_param.length = sizeof(p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement) / sizeof(p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[0]);
            gatt_param.p_data = p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement;

            // send notification
            status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
            if (status == BLE_ERR_OK)
            {
                p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[1]++; //+1, Heart Rate Data. Here just a simulation, increase 1 about
                every second
                p_profile_info->svcs_info_hrs.server_info.data.heart_rate_measurement[2]++; //+1, Heart Rate RR-Interval
            }
        }
        break;

        default:
            break;
    }
}
```

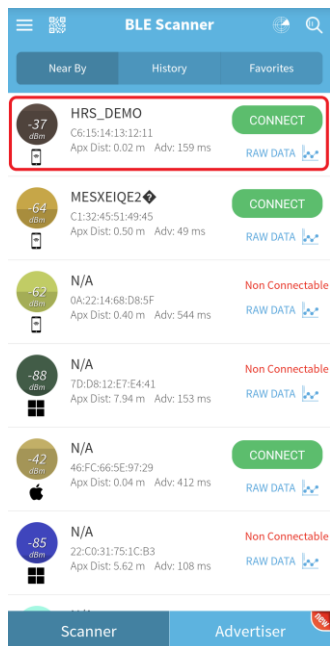
(intentionally blank)

### 2.1.3 Running “HRS\_Peripheral” Demo

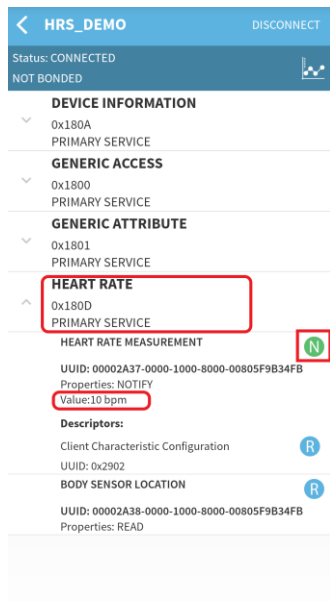
This demo code can be tested with the smart phone.

[Testing]

- Open “BLE Scanner” app and click “Scan” button to start scanning.
- Connect with the desired BLE device.

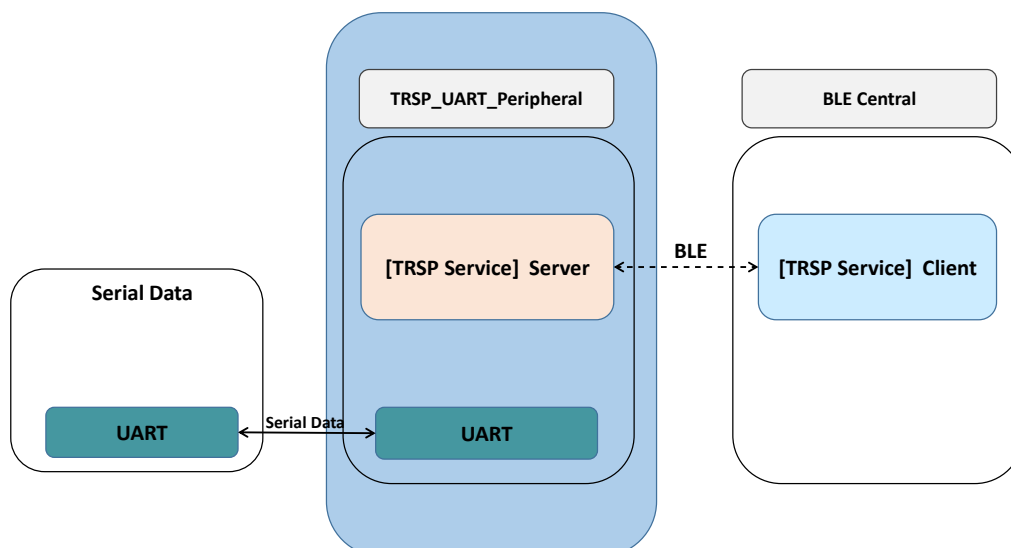


- Click “Heart Rate” service. Enable NOTIFY (it may be shown as N icon, click it to enable/disable) and you will see "Value" increase 1 every one second.



## 2.2 TRSP\_Peripheral

Transparent (TRSP) UART demonstrates the serial data transferred via BLE TRSP service with peripheral role. User could transmit data received from serial port to BLE Peripheral Device via BLE. This sample application could be a reference for users to develop their own applications, i.e., streaming, AT command application.





## 2.2.1 BLE Services

TRSP\_Peripheral application includes three services which are GAP, DIS (Device Information Service) and TRSP service.

### ● GAP

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification “[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C\*](#)” for more details.

### ● Device Information Service (DIS)

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification “[\*DIS SPEC V11r00.pdf\*](#)” for more details. The implementation please refer to “ble\_service\_dis.c/ ble\_service\_dis.h”.

### ● Transparent Service(TRSPS)

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to emulate the serial port over BLE. The procedure of creating a customer service please see “[\*RT58x BLE SDK Service Profile Guide.pdf\*](#)” for more details. The implementation please refer to “ble\_service\_trsp.c/ ble\_service\_trsp.h”.

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.
Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.
Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.

## 2.2.2 Handle Data Received Over UART

Implement MCU UART peripheral and register UART handler function to handle data received from UART. And ask the application to send the data.

[Handle received UART data]

```
bool uart_data_handler(char ch)
{
    bool status = false;
    static uint8_t rx_buffer[250];
    static uint8_t index = 0;

    rx_buffer[index++] = ch;

    if (index >= (g_trsp_mtu - 3)) // 3 bytes header
    {
        // send data via TRSP service
        trsp_data_send_from_isr(rx_buffer, index);

        // reset index
        index = 0;
    }
    else if ((ch == '\n') || (ch == '\r'))
    {
        // send data via TRSP service
        trsp_data_send_from_isr(rx_buffer, (index - 1)); // (-1) removed '\n' or '\r'

        // reset index
        index = 0;

        // set status to true indicates the system can enable sleep mode
        status = true;
    }
    return status;
}
```

[Copy data to global array]

```
// true: indicates that there is data pending, still send old data and skip new data from UART
static bool g_data_send_pending = false;

// transmitted data length
static uint8_t g_rx_buffer_length = 0;

// transmitted data buffer
static uint8_t g_rx_buffer[250];

static void trsp_data_send_from_isr(uint8_t *p_data, uint8_t length)
{
    // if there is data pending, still send old data and skip new data
    if (g_data_send_pending == false)
    {
        g_rx_buffer_length = length;
        memcpy(g_rx_buffer, p_data, g_rx_buffer_length);

        // send queue to task bla app
        app_request_set(APP_TRSP_P_HOST_ID, APP_REQUEST_TRSPS_DATA_SEND, true);
    }
}
```

## 2.2.3 Handle Data Transmitted Over BLE

Wait for requests to transmit data over BLE by using “ble\_cmd\_gatt\_notification” function and also check the sending status to do the flow control.

```
// TRSPS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ble_err_t status;
    uint8_t host_id;
    ble_profile_info_t *p_profile_info;

    host_id = p_param->host_id;
    p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    switch (p_param->app_req)
    {
        case APP_REQUEST_ADV_START:
            ...
            break;

        case APP_REQUEST_TRSPS_DATA_SEND:

            // send data to client
            if ((p_profile_info->svcs_info.trspserver_info.data.udatni01_cccd & BLEGATT_CCCD_NOTIFICATION) != 0)
            {
                status = ble_svcs_trspserver_send( host_id,
                                                    BLEGATT_CCCD_NOTIFICATION,
                                                    p_profile_info->svcs_info.trspserver_info.han-
dles.hdl_udatni01,
                                                    g_rx_buffer,
                                                    g_rx_buffer_length);
            }
            else if ((p_profile_info->svcs_info.trspserver_info.data.udatni01_cccd & BLEGATT_CCCD_INDICATION) !=
0)
            {
                status = ble_svcs_trspserver_send( host_id,
                                                    BLEGATT_CCCD_INDICATION,
                                                    p_profile_info->svcs_info.trspserver_info.han-
dles.hdl_udatni01,
                                                    g_rx_buffer,
                                                    g_rx_buffer_length);
            }

            if (status == BLE_ERR_OK)
            {
                g_data_send_pending = false;
            }
            else
            {
                g_data_send_pending = true;

                // re-send
                app_request_set(host_id, APP_REQUEST_TRSPS_DATA_SEND, false);
            }
            break;

        default:
            break;
    }
}
```

## 2.2.4 Handle Data Received Over BLE

Implement the callback function “ble\_svcs\_trsps\_evt\_handler” which initialized in “server\_profile\_init” function to handle the data received over BLE. In this sample code, print the received data out over UART.

```
static ble_err_t server_profile_init(uint8_t host_id)
{
    ble_err_t status = BLE_ERR_OK;
    ble_profile_info_t *p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    // set link's state
    ble_app_link_info[host_id].state = STATE_STANDBY;

    do
    {
        ...

        // TRSPS Related
        // -----
        status = ble_svcs_trsps_init(host_id, BLE_GATT_ROLE_SERVER, &(p_profile_info->svcs_info_trsps), ble_svcs_trsps_evt_handler);
        if (status != BLE_ERR_OK)
        {
            break;
        }
    } while (0);

    return status;
}
```

(intentionally blank)

```
static void ble_svcs_trsps_evt_handler(ble_evt_att_param_t *p_param)
{
    if (p_param->gatt_role == BLE_GATT_ROLE_SERVER)
    {
        /* ----- Handle event from client ----- */
        switch (p_param->event)
        {
            case BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT:
            case BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT:
                p_param->data[p_param->length] = '\0';
                info_color(LOG_CYAN, "%s\n", p_param->data);
                break;

            case BLESERVICE_TRSPS_UDATRW01_READ_EVENT:
            {
                ble_err_t status;
                const uint8_t readData[] = "UDATRW01 data";
                ble_gatt_data_param_t gatt_data_param;

                gatt_data_param.host_id = p_param->host_id;
                gatt_data_param.handle_num = p_param->handle_num;
                gatt_data_param.length = SIZE_STRING(readData);
                gatt_data_param.p_data = (uint8_t *)readData;

                status = ble_svcs_data_send(TYPE_BLE_GATT_READ_RSP, &gatt_data_param);
                if (status != BLE_ERR_OK)
                {
                    info_color(LOG_RED, "ble_gatt_read_rsp status: %d\n", status);
                }
            }
            break;

            default:
                break;
        }
    }
}
```

## 2.2.5 Handle “Read Request” from client

Implement the callback function “ble\_svcs\_trsps\_evt\_handler” first, then the event “BLESERVICE\_TRSP\_UDATRW01\_READ\_EVENT” will be received when receiving the read request from the client. Issue “ble\_cmd\_gatt\_read\_rsp()” function to send read response with data to the client.

(intentionally blank)

```
static void ble_svcs_trsps_evt_handler(ble_evt_att_param_t *p_param)
{
    if (p_param->gatt_role == BLE_GATT_ROLE_SERVER)
    {
        /* ----- Handle event from client ----- */
        switch (p_param->event)
        {
            case BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT:
            case BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT:
                p_param->data[p_param->length] = '\0';
                info_color(LOG_CYAN, "%s\n", p_param->data);
                break;

            case BLESERVICE_TRSPS_UDATRW01_READ_EVENT:
            {
                ble_err_t status;
                const uint8_t readData[] = "UDATRW01 data";
                ble_gatt_data_param_t gatt_data_param;

                gatt_data_param.host_id = p_param->host_id;
                gatt_data_param.handle_num = p_param->handle_num;
                gatt_data_param.length = SIZE_STRING(readData);
                gatt_data_param.p_data = (uint8_t *)readData;

                status = ble_svcs_data_send(TYPE_BLE_GATT_READ_RSP, &gatt_data_param);
                if (status != BLE_ERR_OK)
                {
                    info_color(LOG_RED, "ble_gatt_read_rsp status: %d\n", status);
                }
            }
            break;

            default:
                break;
        }
    }
}
```

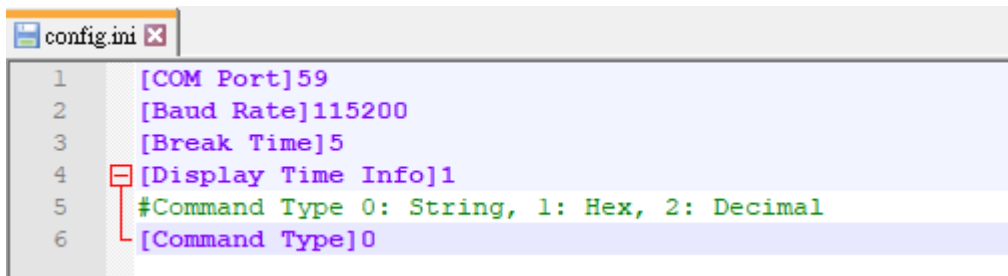
(intentionally blank)

## 2.2.6 Running “TRSP\_Peripheral” Demo

This demo code can be tested with the smart phone or with the demo “TRSP\_Peripheral”. Please download “BLE Scanner” app for the test. Here shows the step of testing with [smart phone](#). The steps of testing with demo “TRSP\_Central” please refer to [section 3.1.5](#).

### [Preparation]

- Setup “UART BREAK” Tool. The settings are as follows.



- Setup UART Environment: power-on the device and the initial log will show on the screen.

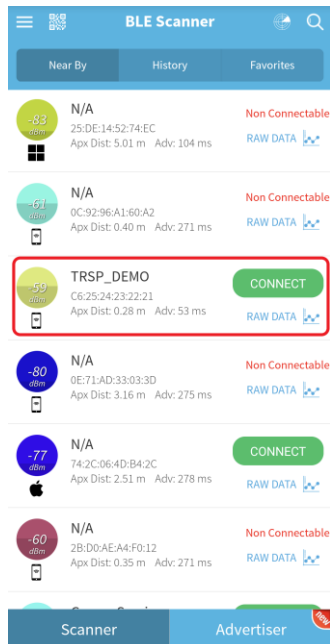
```
***** Please send command *****
[BOOT] Bootloader Running...
[BOOT] No Secure Boot.
[BOOT] Bootloader check is complete. Boot into application.
-----
BLE TRSP (P) demo: start...
-----
BLE stack initial...
Write default data length,status: 0
Advertising...
```

- Open “BLE Scanner” app and ensure the Bluetooth is enabled.

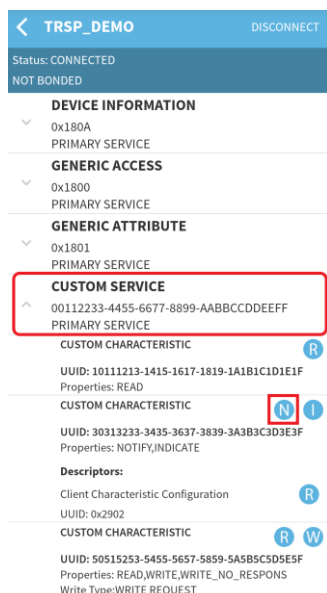
(intentionally blank)

## [Testing]

- Open “BLE Scanner” app and click “Scan” button to start scanning.
- Connect with the desired BLE device.



- Click “CUSTOM SERVICE” and click “N” to enable notification to receive the notification from the BLE device.





- Data transmitting test → UART data sending over BLE.  
Enter data in UART to transmit to smart phone over BLE.

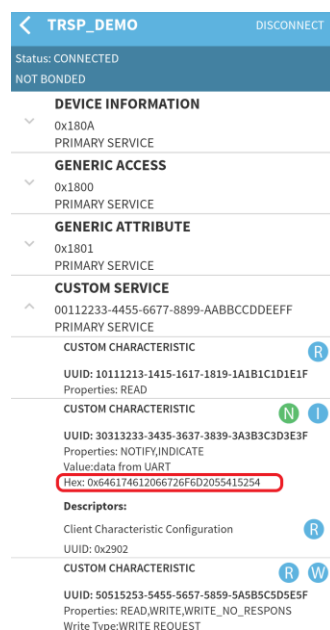
```

BLE TRSP (P) demo: start...

BLE stack initial...
Write default data length, status: 0
Advertising...
Connected, ID=0, Connected to 6d:20:ed:06:d3:f0
Connection updated
ID: 0, Interval: 6, Latency: 0, Supervision Timeout: 500
Data length changed, ID: 0
MaxTxOctets: 251 MaxTxTime:2120
MaxRxOctets: 27 MaxRxTime:328
Connection updated
ID: 0, Interval: 39, Latency: 0, Supervision Timeout: 500
Data From UART

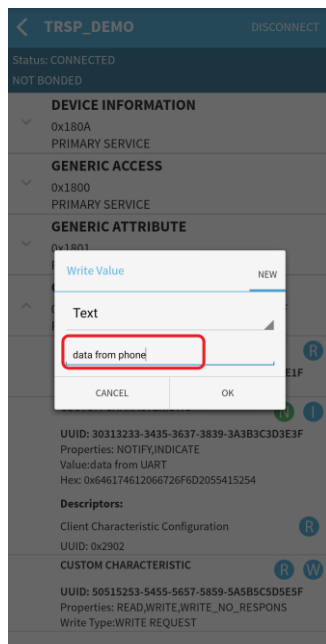
```

In app, here shows the data transmitted from BLE device in red region.



(intentionally blank)

- Data transmitting test → Printing data which received from BLE.  
Press “W” button and enter data in the message box then press “OK” to transmit the data over BLE.



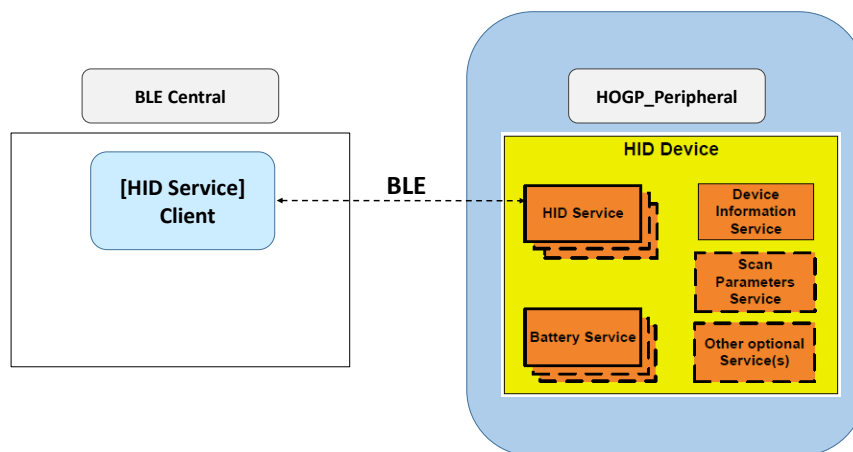
In UART, here shows the data received from BLE.

```
-----
BLE TRSP (P) demo: start...
-----
BLE stack initial...
Write default data length, status: 0
Advertising...
Connected, ID=0, Connected to 6d:20:ed:06:d3:f0
Connection updated
ID: 0, Interval: 6, Latency: 0, Supervision Timeout: 500
Data length changed, ID: 0
MaxTxOctets: 251 MaxTxTime:2120
MaxRxOctets: 27 MaxRxTime:328
Connection updated
ID: 0, Interval: 39, Latency: 0, Supervision Timeout: 500
Data From UART
Data from phone
█
```

(intentionally blank)

## 2.3 HOGP\_Peripheral

The HOGP Application Demo is an example that implements the HID over GATT profile for mouse, keyboard and multimedia functions. In this demo, keyboard, mouse and multimedia data transfer will be shown in action from a BLE HOGP-enabled device.



### 2.3.1 BLE Services

According to the BLE HOGP profile specification “[HOGP\\_SPEC\\_V10.pdf](#)”, it consists of at least four services, GAP, DIS (Device Information Service), BAS (Battery Service) and Human Interface Device Service (HIDS).

#### ● GAP

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification “[BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C](#)” for more details.

#### ● Device Information Service (DIS)

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification “[DIS\\_SPEC\\_V11r00.pdf](#)” for more details. The implementation please refer to “ble\_service\_dis.c/ ble\_service\_dis.h”.

#### ● Battery Service (BAS)

Battery Service (BAS) is a SIG defined service and please refer to SIG Bluetooth specification “[BAS\\_SPEC\\_V10.pdf](#)” for more details.

## ● Human Interface Service (HIDS)

Human Interface Device Service (HIDS) is a SIG defined service and please refer to SIG Bluetooth specification “*HIDS\_SPEC\_V10.pdf*” for more details.

### 2.3.2 Handle the IO Capability Setting

“HOGP\_Peripheral” demo project demonstrates how to set two different IO Capability in Rafael RT58x BLE SDK. The definition of “IO\_CAPABILITY\_SETTING” provides “**KEYBOARD\_ONLY**” and “**DISPLAY\_ONLY**” for selection in “ble\_app.h” file.

```
#define IO_CAPABILITY_SETTING      KEYBOARD_ONLY    //selectd IO Capability: KEYBOARD_ONLY / DISPLAY_ONLY
```

```
static ble_err_t ble_init(void)
{
    ble_err_t status = BLE_ERR_OK;

    do
    {
        status = ble_cmd_phy_controller_init();
        if (status != BLE_ERR_OK)
        {
            break;
        }

        status = ble_cmd_device_addr_set((ble_gap_addr_t *)&DEVICE_ADDR);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        status = ble_cmd_suggest_data_len_set(BLE_GATT_DATA_LENGTH_MAX);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        // BLE profile init --> only 1 link (host id = 0)
        status = server_profile_init(APP_HID_P_HOST_ID);
        if (status != BLE_ERR_OK)
        {
            break;
        }

        //set BLE IO capabilities
        status = io_capability_init(IO_CAPABILITY_SETTING);
        if (status != BLE_ERR_OK)
        {
            break;
        }
    } while (0);

    return status;
}
```

(intentionally blank)

- IO Capability - **KEYBOARD\_ONLY**

When pairing, the client (i.e., smart phone) shows a pairing window with a pairing key. User have to enter the pairing key through UART to complete the pairing. The pairing key is set in “user.h”.

Info user to enter the passkey when get “BLE\_SM\_EVT\_PASKEY\_CONFIRM” event.

```
static void ble_evt_handler(ble_evt_param_t *p_param)
{
    switch (p_param->event)
    {
        ...
        case BLE_SM_EVT_PASKEY_CONFIRM:
        {
            //enter a scanned Passkey or use a randomly generated passkey.
            #if (IO_CAPABILITY_SETTING == DISPLAY_ONLY)

                ble_evt_sm_passkey_confirm_param_t *p_cfm_param = (ble_evt_sm_passkey_confirm_param_t *)&p_param-
                >event_param.ble_evt_sm.param.evt_passkey_confirm_param;
                ble_sm_passkey_param_t passkey_param;

                passkey_param.host_id = p_cfm_param->host_id;
                passkey_param.passkey = (uint32_t)DEMO_HID_DISPLAY_PASSKEY;

                // set passkey
                ble_cmd_passkey_set(&passkey_param);

            #elif (IO_CAPABILITY_SETTING == KEYBOARD_ONLY)
                ble_passkey_confirmed_state = 1;
                info_color(LOG_CYAN, "Please enter passkey...\n");
            #endif
        }
        break;

        ...

        default:
            break;
    }
}
```

(intentionally blank)

Set the received passkey to BLE stack by using “ble\_cmd\_passkey\_set()” function.

```
// HIDS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ble_err_t status;
    uint8_t host_id;
    ble_profile_info_t *p_profile_info;

    host_id = p_param->host_id;
    p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    switch (p_param->app_req)
    {
        ...

        case APP_REQUEST_HIDS_PASKEY_ENTRY:
        {
            #if (IO_CAPABILITY_SETTING == KEYBOARD_ONLY)
                ble_sm_passkey_param_t param;

                param.host_id = host_id;
                param.passkey = (uint32_t)passkey;

                info_color(LOG_CYAN, "BLE PAIRING_KEY = %06d\n", passkey);           // show the passkey
                ble_cmd_passkey_set(&param);
            #endif
        }
        break;

        ...

        default:
            break;
    }
}
```

(intentionally blank)

- IO Capability - **DISPLAY\_ONLY**

When pairing, the client (i.e., smart phone) shows a pairing window. User has to enter the pairing key in the client to complete the pairing. The pairing key is set in “ble\_app.c” file.

```
#define DEMO_HID_DISPLAY_PASSKEY          654321 //HID pairing key
```

Set the defined passkey by using “ble\_cmd\_passkey\_set()” function.

```
static void ble_evt_handler(ble_evt_param_t *p_param)
{
    switch (p_param->event)
    {
        ...
        case BLE_SM_EVT_PASSKEY_CONFIRM:
        {
            //enter a scanned Passkey or use a randomly generated passkey.
            #if (IO_CAPABILITY_SETTING == DISPLAY_ONLY)

                ble_evt_sm_passkey_confirm_param_t *p_cfm_param = (ble_evt_sm_passkey_confirm_param_t *)&p_param-
>event_param.ble_evt_sm.param.evt_passkey_confirm_param;
                ble_sm_passkey_param_t passkey_param;

                passkey_param.host_id = p_cfm_param->host_id;
                passkey_param.passkey = (uint32_t)DEMO_HID_DISPLAY_PASSKEY;

                // set passkey
                ble_cmd_passkey_set(&passkey_param);

            #elif (IO_CAPABILITY_SETTING == KEYBOARD_ONLY)
                ble_passkey_confirmed_state = 1;
                info_color(LOG_CYAN, "Please enter passkey...\n");
            #endif
        }
        break;

        ...

        default:
            break;
    }
}
```

(intentionally blank)

### 2.3.3 Handle the HID Data Transmitted

Create a FreeRTOS software timer with a period of 50ms to control keyboard, mouse and multimedia data to the client by using “ble\_cmd\_gatt\_notification” function.

**[Mouse]** The mouse will have a diamond-shaped movement track

(intentionally blank)



```
// HIDS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ...
    case APP_REQUEST_HIDS_NTF:
    {
        ble_gatt_data_param_t gatt_param;

        // send heart rate measurement value to client
        if ((hid_report_count & 0x3F) != 0x3F)    //(counter value!=0x3F or 0x7F or 0xBF or 0xFF)
        {
            if (hid_report_count < 0x80)
            {
                if (p_profile_info->svcs_info_hids.server_info.data.mouse_input_report_cccd != 0)
                {
                    if (hid_report_count <= 0x1F)    //counter 0~0x1F, mouse move right-down
                    {
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_L] = 0x05;    //
right
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_H] = 0x00;
down
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_L] = 0x05;    //
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_H] = 0x00;
                    }
                    else if (hid_report_count <= 0x3F)    //counter 0x20~0x3F, mouse move left-down
                    {
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_L] = 0xFA;    //
left
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_H] = 0xFF;
down
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_L] = 0x05;    //
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_H] = 0x00;
                    }
                    else if (hid_report_count <= 0x5F)    //counter 0x40~0x5F, mouse move left-up
                    {
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_L] = 0xFA;    //
left
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_H] = 0xFF;
up
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_L] = 0xFA;    //
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_H] = 0xFF;
                    }
                    else if (hid_report_count <= 0x7F)    //counter 0x60~0x7F, mouse move right-up
                    {
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_L] = 0x05;    //
right
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_L_R_H] = 0x00;
up
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_L] = 0xFA;    //
                        p_profile_info->svcs_info_hids.server_info.data.mouse_input_report[HDL_HIDS_REPORT_TAB_DIR_U_D_H] = 0xFF;
                    }
                }

                // set parameters
                gatt_param.host_id = host_id;
                gatt_param.handle_num = p_profile_info->svcs_info_hids.server_info.handles.hdl_mouse_input_report;
                gatt_param.length = sizeof(p_profile_info->svcs_info_hids.server_info.data.mouse_input_report);
                gatt_param.p_data = p_profile_info->svcs_info_hids.server_info.data.mouse_input_report;

                // send notification
                status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
                if (status == BLE_ERR_OK)
                {
                    hid_report_count++;    //counter++
                }
            }
            else
            {
                hid_report_count++;    //counter++
            }
        }
        else
        {
            hid_report_count++;    //counter++
        }
    }
    ...
}
}
```

**[Keyboard]** The device will get the a~z and 1~9 keyboard input actions.

```
// HIDS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ...
    case APP_REQUEST_HIDS_NTF:
    {
        ble_gatt_data_param_t gatt_param;

        if ((hid_report_count == 0x3F) || (hid_report_count == 0xBF)) //control keyboard when counter=0x3F, 0xBF
        {
            if (p_profile_info->svcs_info_hids.server_info.data.keyboard_input_report_cccd != 0)
            {
                if ((hid_keyboard_report_state & STATE_HID_REPORT_KB_DATA_UPD) == 0) //check keyboard report status
                {
                    if ((hid_keyboard_report_count <= 0x04) || (hid_keyboard_report_count >= 0x27))
                    {
                        hid_keyboard_report_count = 0x04; //0x04 mean 'a'; 0x27 mean '9'; see USB HID spec.
                    }
                    p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report[HDL_HIDS_REPORT_TAB_KEY_DATA0] = hid_keyboard_report_count; // repeat keyCode: 'a' 'b' ~ 'z' ~ '1' '2' ~ '9'

                    // set parameters
                    gatt_param.host_id = host_id;
                    gatt_param.handle_num = p_profile_info->svcs_info_hids.server_info.handles.hdl_keyboard_input_report;
                    gatt_param.length = sizeof(p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report);
                    gatt_param.p_data = p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report;

                    // send notification
                    status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
                    if (status == BLE_ERR_OK)
                    {
                        hid_keyboard_report_state |= STATE_HID_REPORT_KB_DATA_UPD;
                        hid_keyboard_report_count++; //keyboard keycode
                    }
                }
            }
            else //release key
            {
                p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report[HDL_HIDS_REPORT_TAB_KEY_DATA0] = 0x00; // release key

                // set parameters
                gatt_param.host_id = host_id;
                gatt_param.handle_num = p_profile_info->svcs_info_hids.server_info.handles.hdl_keyboard_input_report;
                gatt_param.length = sizeof(p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report);
                gatt_param.p_data = p_profile_info->svcs_info_hids.server_info.data.keyboard_intput_report;

                // send notification
                status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
                if (status == BLE_ERR_OK)
                {
                    hid_keyboard_report_state &= ~STATE_HID_REPORT_KB_DATA_UPD;
                    hid_report_count++;
                }
            }
        }
        else
        {
            hid_report_count++;
        }
    }
    ...
}
```

## [Multimedia]Multimedia volume control.

```
// HIDS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ...
    case APP_REQUEST_HIDS_NTF:
    {
        ble_gatt_data_param_t gatt_param;

        if ((hid_report_count == 0x7F) || (hid_report_count == 0xFF)) //control volume when counter=0x7F, 0xFF
        {
            if (p_profile_info->svcs_info_hids.server_info.data.consumer_input_report_cccd != 0)
            {
                if ((hid_consumer_report_state & STATE_HID_REPORT_CS_DATA_UPD) == 0) // check consumer report status
                {
                    if ((hid_consumer_report_count & 0x01) == 0x01)
                    {
                        p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[0] = hids_consumer_re-
port_keycode_demo[0][0]; // vol+
                        p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[1] = hids_consumer_re-
port_keycode_demo[0][1];
                    }
                    else
                    {
                        p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[0] = hids_consumer_re-
port_keycode_demo[1][0]; // vol-
                        p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[1] = hids_consumer_re-
port_keycode_demo[1][1];
                    }

                    // set parameters
                    gatt_param.host_id = host_id;
                    gatt_param.handle_num = p_profile_info->svcs_info_hids.server_info.handles.hdl_consumer_input_report;
                    gatt_param.length = sizeof(p_profile_info->svcs_info_hids.server_info.data.consumer_input_report);
                    gatt_param.p_data = p_profile_info->svcs_info_hids.server_info.data.consumer_input_report;

                    // send notification
                    status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
                    if (status == BLE_ERR_OK)
                    {
                        hid_consumer_report_state |= STATE_HID_REPORT_CS_DATA_UPD;
                        hid_consumer_report_count++; // just counter for send another consumer data
                    }
                }
            }
            else //release key
            {
                p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[0] = 0x00; // release key
                p_profile_info->svcs_info_hids.server_info.data.consumer_input_report[1] = 0x00;

                // set parameters
                gatt_param.host_id = host_id;
                gatt_param.handle_num = p_profile_info->svcs_info_hids.server_info.handles.hdl_consumer_input_report;
                gatt_param.length = sizeof(p_profile_info->svcs_info_hids.server_info.data.consumer_input_report);
                gatt_param.p_data = p_profile_info->svcs_info_hids.server_info.data.consumer_input_report;

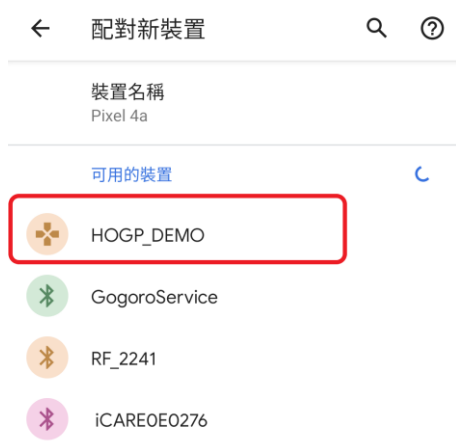
                // send notification
                status = ble_svcs_data_send(TYPE_BLE_GATT_NOTIFICATION, &gatt_param);
                if (status == BLE_ERR_OK)
                {
                    hid_consumer_report_state &= ~STATE_HID_REPORT_CS_DATA_UPD;
                    hid_report_count++;
                }
            }
        }
        else
        {
            hid_report_count++;
        }
    }
}
break;
...
}
```

### 2.3.4 Running “HOGP\_Peripheral” Demo

User can test this demo code with smart phone and notebook or BLE HOGP-enabled device. When connecting to a HOGP device, the connected device will show corresponding mouse, keyboard and multimedia control actions.

#### [Testing]

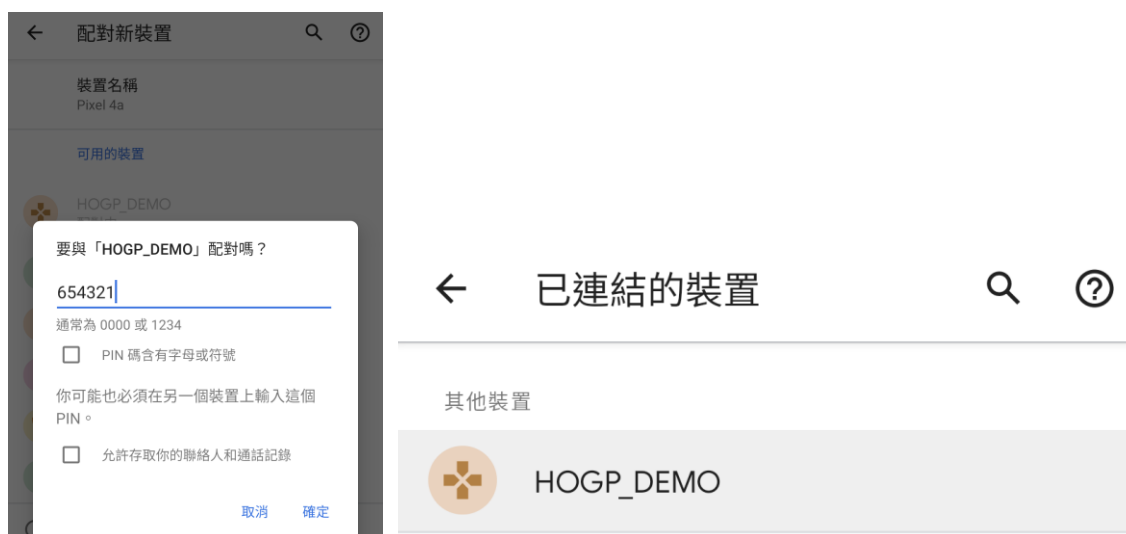
- Enable the central device's (i.e., smart phone) Bluetooth. The central device's (i.e., smart phone) should scan the BLE device, like the picture below:



- Select the target device and the central device's (i.e., smart phone) shows a pairing window. Please note that the pairing process is different with different IO Capability setting. For more information, please refer to [section 2.3.2](#).

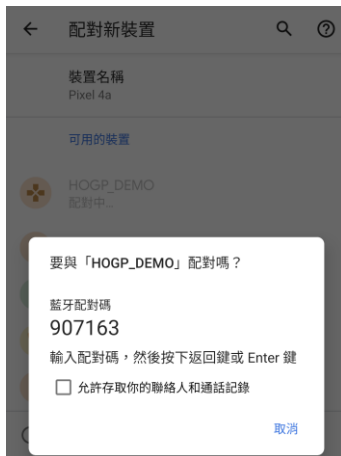
#### [IO Capability: **DISPLAY\_ONLY**]

Enter pairing key "654321" in the central device (i.e., smart phone) to complete the pairing.



# [IO Capability:KEYBOARD\_ONLY]

The pairing key shows on the pairing window in the central device (i.e., smart phone). Please enter the pairing key in UART to complete the pairing.



```
-----
BLE HOGP demo: start...
-----
Advertising...
[BLE_HOST_TASK] BLE_GAP_EVT_CONN_COMPLETE
Connected, ID=0, Connected to 57:e9:cc:e0:2c:06
Please enter passkey...
907163
BLE_PAIRING_KEY = 907163
BLE authentication status = 0
-----
```

- After pairing, a mouse pointer will appear. HOGP\_mouse controls pointer to draw a diamond slowly. At the time mouse pointer moves to the top, HOGP\_consumer controls volume.



- If user open a key-in window application (e.g.: google search) on the central device (i.e., smart phone), HOGP\_keyboard repeat keys 'a' 'b' ~ 'z' '1' '2' ~ '9'.



- the central device (i.e., smart phone), turn off Bluetooth to disconnect the link. Since this demo enables **BONDING** option, security key is stored in MCU flash.
- On the central device (i.e., smart phone), turn on Bluetooth. Since EVK and mobile phone are bonded, BLE HOGP device auto connect to mobile phone and start the mouse, keyboard, multimedia volume control actions.

### 2.3.5 Running “HOGP\_Peripheral\_Privacy” Demo

This demonstration is derivation from HOGP\_Peripheral, the purpose of this demo is to let users understand how to implement privacy functions through specific text comparison tool. So the application operation method is the same as HOGP\_Peripheral, the only difference is this demo provides enabling and disabling privacy feature through the key1 of EVK and the privacy feature is disabled by default. Note that the privacy feature can only be enable or disable when advertising is disabled.

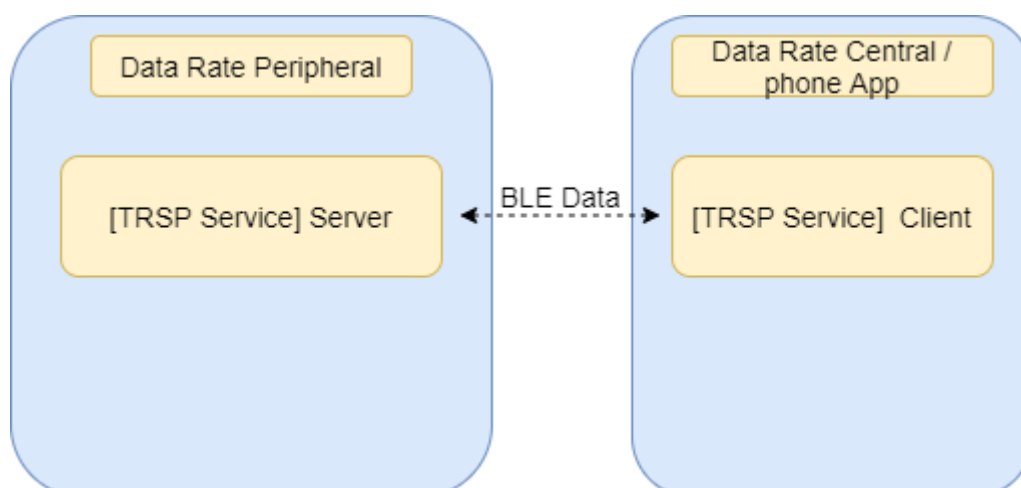
```
BLE authentication status = 0
Connection updated
ID: 0, Interval: 6, Latency: 0, Supervision Timeout: 500
Connection updated
ID: 0, Interval: 36, Latency: 0, Supervision Timeout: 500
privacy enable success
```

When privacy feature is enabled, the device will only accept the connections create by the same central device, if device finds that the central device is different, it will immediately terminate the connection.

```
privacy enable success
Disconnect, ID:0, Reason:0x13
Advertising...
Connected, ID=0, Connected to 6c:4d:e7:b5:46:74
Disconnect, ID:0, Reason:0x16
Advertising...
Connected, ID=0, Connected to 6c:4d:e7:b5:46:74
Disconnect, ID:0, Reason:0x16
Advertising...
```

### 2.4 Data\_Rate\_Peripheral

The Data Rate Peripheral demonstrates BLE data transmission with the Peripheral role via the BLE TRSP service. The user can demonstrate the effectiveness of data transmission through the handset or the data rate central device. With this demonstration, users can plan high throughput applications, but it should be noted that the amount of transmission throughput will depend on the phone, and application development planning still needs to be done at the highest transmission rate that most handsets can support.



### 2.4.1 BLE Services

Data\_Rate\_Peripheral application includes three services which are GAP, DIS (Device Information Service) and TRSP service.

- **GAP**

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification “[BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C](#)” for more details.

- **Device Information Service (DIS)**

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification “[DIS SPEC V11r00.pdf](#)” for more details. The implementation please refer to “ble\_service\_dis.c/ ble\_service\_dis.h”.

- **Transparent Service(TRSPS)**

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to emulate the serial port over BLE. The procedure of creating a customer service please see “[RT58x BLE SDK Service Profile Guide.pdf](#)” for more details. The implementation please refer to “ble\_service\_trsp.c/ ble\_service\_trsp.h”.

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.

Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.
Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.

(intentionally blank)

### 2.4.2 Handle Data Rate Command Received Over BLE

Implement the callback function “ble\_svcs\_trsps\_evt\_handler()” which initialized in “server\_profile\_init” function to handle the data received over BLE. In this sample code, define multiple data rate identified strings to indicate different commands, and handle the received data to identify the command by “trsps\_write\_cmd\_handler” function.



```
/* Data rate cmd identifier */

// Receive from central device, set data rate parameters for the test
#define SET_PARAM_STR          "set_param"
// Receive from central device, send current data rate parameters for
// the central device
#define GET_PARAM_STR          "get_param"
// Receive from central device, test C->P data rate
#define PRX_TEST_STR           "pRxtest"
// Receive from central device, test P->C data rate
#define PTX_TEST_STR           "pTxtest"
// Receive from central device, cancel the test case
#define CANCEL_TEST_STR        "canceltest"

static void ble_svcs_trsps_evt_handler(ble_evt_att_param_t *p_param)
{
    if (p_param->gatt_role == BLE_GATT_ROLE_SERVER)
    {
        /* ----- Handle event from client ----- */
        switch (p_param->event)
        {
            case BLESERVICE_TRSPS_UDATRW01_WRITE_EVENT:
            case BLESERVICE_TRSPS_UDATRW01_WRITE_WITHOUT_RSP_EVENT:
                // process test command
                trsps_write_cmd_handler(p_param->host_id, p_param->length, p_param->data);
                break;

            case BLESERVICE_TRSPS_UDATRW01_READ_EVENT:
                // send read rsp with current test parameters
                trsps_read_handler(p_param->host_id, p_param->handle_num);
                break;

            default:
                break;
        }
    }
}
```

#### Defined Identifiers:

##### - “set\_param”

Set preferred BLE parameters which are BLE PHY, transmitted packet data length, minimum connection interval and maximum connection interval from the client.

Format is shown as below:

*“set\_param:phy,packetLength, connection interval min., connection interval max.”*

```
static void trsps_write_cmd_handler(uint8_t host_id, uint8_t length, uint8_t *data)
{
    uint32_t dataLen;

    ...

    else if (CHECK_STR(data, SET_PARAM_STR))
    {
        int state;

        // get Data Rate Parameters
        data[length] = 0;
        state = sscanf((char *)data, "set_param=%hhu,%hhu,%hu,%hu",
                        &(g_temp_test_param.phy),
                        &(g_temp_test_param.packet_data_len),
                        &(g_temp_test_param.conn_interval_min),
                        &(g_temp_test_param.conn_interval_max));

        if (state == -1)
        {
            info_color(LOG_RED, "the params of set_param cmd is wrong!\n");
        }
        else
        {
            // set set parameteres request
            app_request_set(host_id, APP_REQUEST_TEST_PARAM_SET, false);
        }
    }
    else if (CHECK_STR(data, GET_PARAM_STR))
    {
        // set get parameteres request
        app_request_set(host_id, APP_REQUEST_TEST_PARAM_GET, false);
    }
}

...
```

(intentionally blank)

## - “get\_param”

Send device current BLE parameters which are BLE PHY, MTU size, transmitted packet data length, connection interval, latency and connection supervision timeout in hex to the client by “ble\_cmd\_gatt\_notification” function.

```
static void trsps write cmd handler(uint8_t host_id, uint8_t length, uint8_t *data)
{
    uint32_t dataLen;
    ...
    else if (CHECK_STR(data, GET_PARAM_STR))
    {
        // set get parameteres request
        app_request_set(host_id, APP_REQUEST_TEST_PARAM_GET, false);
    }
    ...
}

// TRSPS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ble_err_t status;
    uint8_t host_id;
    ble_profile_info_t *p_profile_info;

    host_id = p_param->host_id;
    p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    switch (p_param->app_req)
    {
        ...
        case APP_REQUEST_TEST_PARAM_GET:
        {
            uint8_t notify_data[100];
            uint32_t notify_len;

            notify_len = sprintf((char *)notify_data, "%x,%x,%x,%x,%x,%x",
                                g_curr_test_param.phy,
                                g_curr_test_param.mtu_size,
                                g_curr_test_param.packet_data_len,
                                g_curr_test_param.conn_interval,
                                g_curr_test_param.conn_latency,
                                g_curr_test_param.conn_supervision_timeout);

            status = ble_svcs_trsps_server_send(host_id, BLEGATT_CCCD_NOTIFICATION, p_profile_info->svcs_info_trsps.server_info.handles.hdl_uatni01, notify_data, notify_len);
            if (status != BLE_ERR_OK)
            {
                info_color(LOG_RED, "ble_svcs_trsps_server_send status = 0x%02x\n", status);
            }
            break;
            ...
        }
    }
}
```

(intentionally blank)

- **“canceltest”**

Stop the current test immediately.

```
static void trsps_write_cmd_handler(uint8_t host_id, uint8_t length, uint8_t *data)
{
    uint32_t dataLen;

    if (CHECK_STR(data, CANCEL_TEST_STR))
    {
        // cancel the test
        ble_app_link_info[host_id].state = STATE_TEST_STANDBY;

        // stop timer
        Timer_Stop(APP_HW_TIMER_ID);

        info_color(LOG_DEFAULT, "Cancel Test.\n");
    }
    ...
}
```

(intentionally blank)

## - “pTxtest”

Following the identifier “pTxtest” is the total length of device transmission test. Device transmission test will be start after parsing the total length.

```
static void trsps_write_cmd_handler(uint8_t host_id, uint8_t length, uint8_t *data)
{
    uint32_t dataLen;

    ...

    else if (CHECK_STR(data, PTX_TEST_STR))
    {
        // start device TX test
        ble_app_link_info[host_id].state = STATE_TEST_TXING;

        // init parameter
        g_curr_tx_lenght = 0;

        // total test length follows the test string
        data[length] = 0;
        sscanf((char *) (data + strlen(PTX_TEST_STR)), "%d", &dataLen);
        g_test_length = dataLen;

        // set TX test request
        app_request_set(host_id, APP_REQUEST_TX_TEST, false);

        info_color(LOG_DEFAULT, "Test length = %d\n", g_test_length);
        info_color(LOG_DEFAULT, "Start TX...\n");
    }

    ...
}
```

```
// TRSPS Peripheral
static void app_peripheral_handler(app_req_param_t *p_param)
{
    ...
    case APP_REQUEST_TX_TEST:
    {
        uint32_t packet_len = g_curr_test_param.packet_data_len;

        if ((g_curr_tx_lenght + g_curr_test_param.packet_data_len) > g_test_length)
        {
            packet_len = g_test_length - g_curr_tx_lenght;
        }

        status = ble_svcs_trsps_server_send(host_id,
                                           BLEGATT_CCCD_NOTIFICATION,
                                           (ble_profile_info_t
*)ble_app_link_info[host_id].profile_info->svcs_info_trsps.server_info.handles.hdl_udatni01,
                                           g_test_buffer,
                                           packet_len);

        if (status == BLE_ERR_OK)
        {
            g_curr_tx_lenght += packet_len;

            if (g_curr_tx_lenght >= g_test_length)
            {
                // end of the TX test.
                ble_app_link_info[host_id].state = STATE_TEST_STANDBY;
                info_color(LOG_DEFAULT, "Stop TX\n");
                return;
            }
        }

        // issue APP_REQUEST_TX_TEST again
        app_request_set(host_id, APP_REQUEST_TX_TEST, false);
    }
    break;
    ...
}
```

## - “pRxtest”

The client device will start transmitting data after sending identifier “pRxtest” to the device. The total length is the string follows “pRxtest”.

```
static void trsps_write_cmd_handler(uint8_t host_id, uint8_t length, uint8_t *data)
{
    uint32_t dataLen;

    ...

    if (CHECK_STR(data, PRX_TEST_STR))
    {
        // start device RX test
        ble_app_link_info[host_id].state = STATE_TEST_RXING;

        // init parameters
        g_time_ms = 0;
        g_curr_rx_length = 0;

        // start timer
        Timer_Start(APP_HW_TIMER_ID, APP_TIMER_MS_TO_TICK(1));

        // total test length follows the test string
        data[length] = 0;
        sscanf((char *) (data + strlen(PRX_TEST_STR)), "%d", &dataLen);
        g_test_length = dataLen;

        info_color(LOG_DEFAULT, "Test length = %d\n", g_test_length);
        info_color(LOG_DEFAULT, "Start RX...\n");
    }

    ...
}
```

## - “connLatencytest0”

The client device “connLatencytest0” to the device. And the device will send the connection parameter update with latency value is 0.

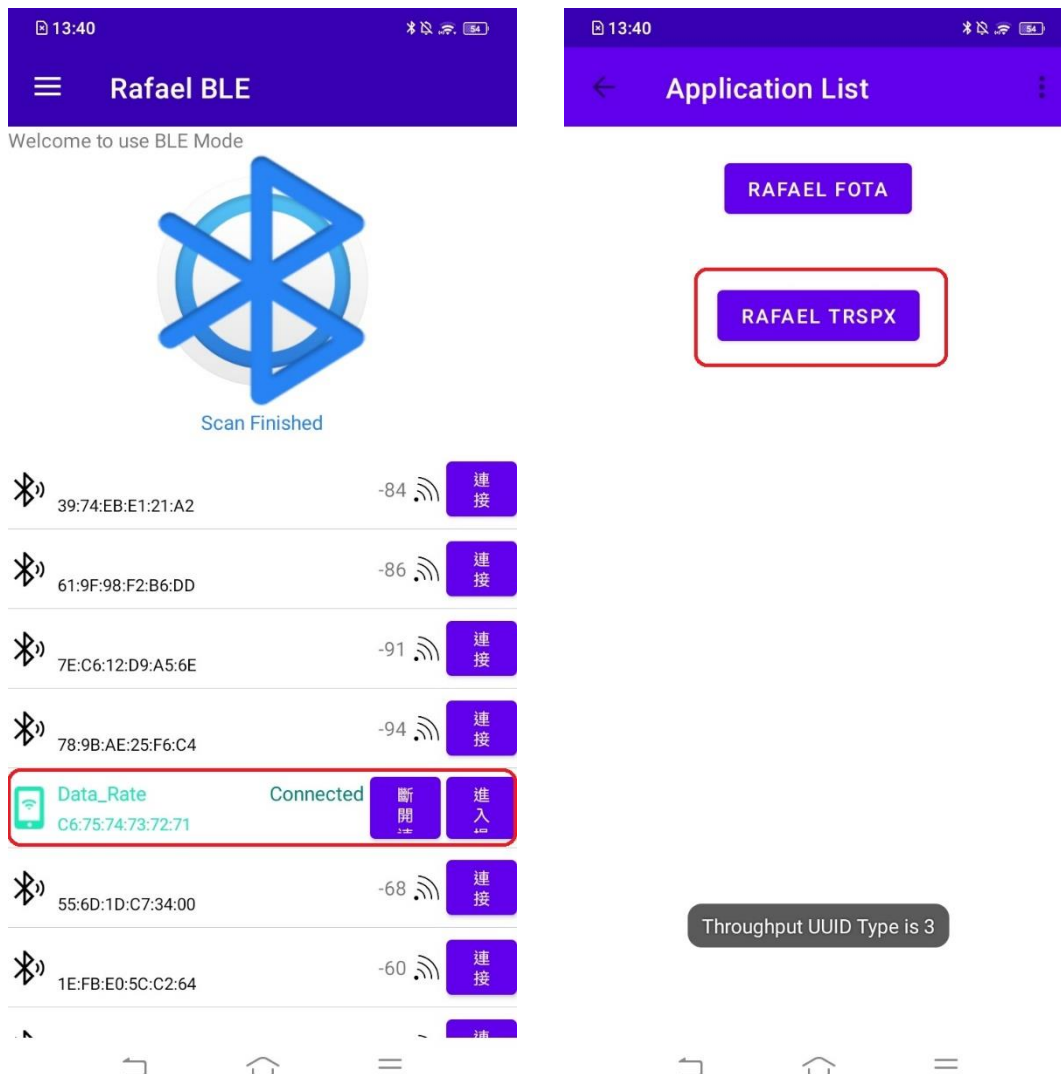
(intentionally blank)

### 2.4.3 Running “Data\_Rate\_Peripheral” Demo with Rafeal BLE APP

User can test this demo code with smart phone. Please download “Rafael\_BLE App” for the test

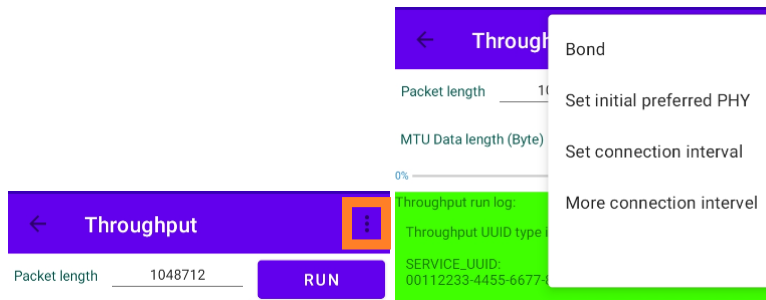
#### [Preparation]

- Open “Rafael\_BLE App” app and enter “Rafael BLE” page.
- click “START SCANNING” button to start scanning.
- Click “Connect” button to connect with the desired BLE device.
- Click “enter” to test page.

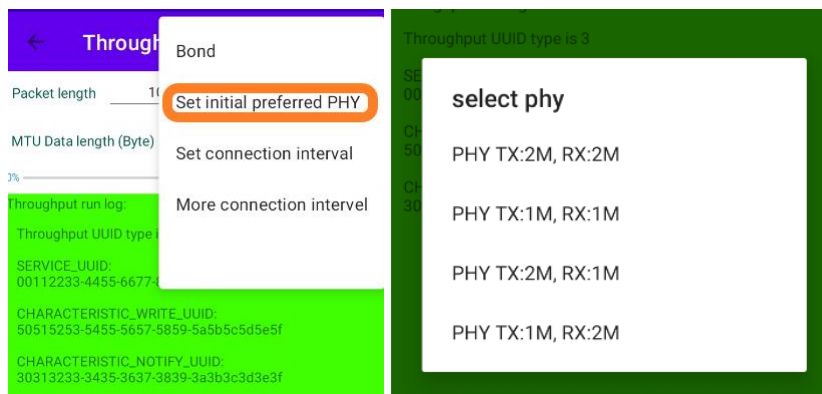


## [General Setting]

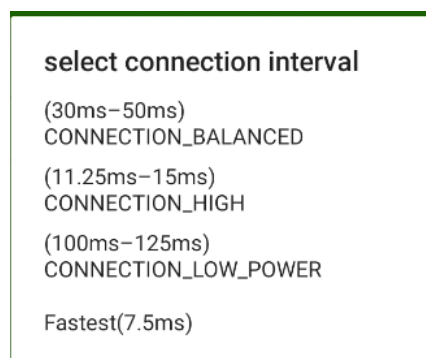
- Click “.” button in the upper right corner.



- Set **PHY**:  
Click “Set initial preferred PHY”, and click the specified button to set BLE TX/RX PHY.  
(The setting of TX PHY must equal to RX PHY)



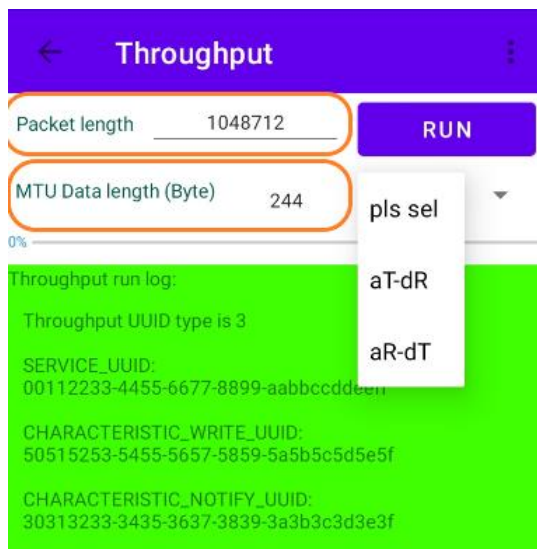
- Set **Connection Interval**:  
Click “Set connection interval”, and click the specified button to set connection interval range. There are three connection interval range to choose from,  
“CONNECTION\_BALANCED” (30ms-50ms), “CONNECTION\_HIGH” (11.25ms-15ms)  
and “CONNECTION\_LOW\_POWER” (100ms-125ms)



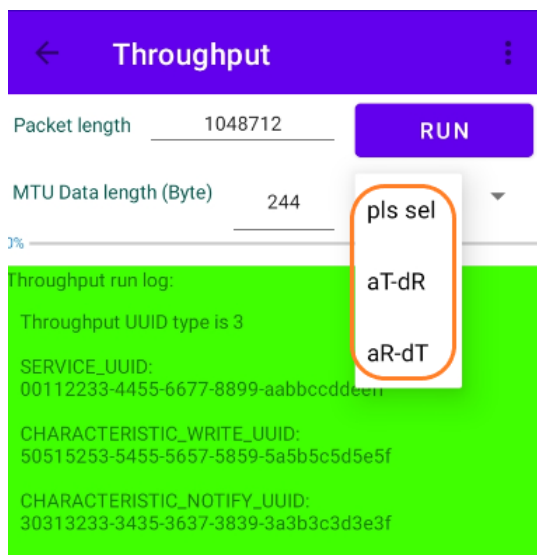


## [Demo Setting]

- Packet length:  
The length of total data which will be transmitted over this test. The value should be large than 0. Default is 1048712 Bytes.
- Packet length (Byte):  
The length of data which will be transmitted. The range of this value is 20-244, and the default is 244. Notice that the MTU size will be set to (MTU data length+3) Bytes (the length reserved of header).



- “pls sel” (please select):  
The direction of data transmitting. The option of “aT-dR” means that data transmits from App to Device. The option of “aR-dT” means that data transmits from Device to App.



#### 2.4.4 Running “Data\_Rate\_Peripheral” Demo with “Data\_Rate\_Central”

User can test this demo code with Data\_Rate\_Central. Please refer to *section 3.2*

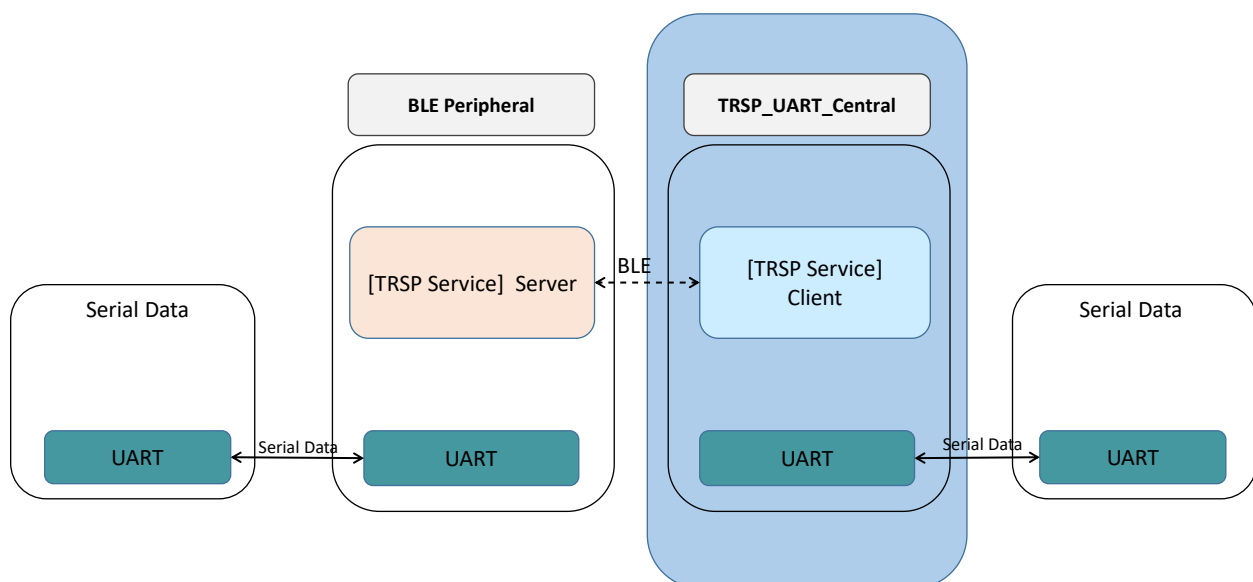
(intentionally blank)

### 3. BLE Central Demo Application

There are two BLE central demo projects for reference, which are TRSP\_UART\_Central and Data\_Rate\_Central.

#### 3.1 TRSP\_Central

Transparent (TRSP) UART demonstrates the serial data transferred via BLE TRSP service with central role. User could transmit data received from serial port to BLE Peripheral Device via BLE. This sample application could be a reference for users to develop their own applications, i.e., streaming, AT command application.



(intentionally blank)

### 3.1.1 BLE Services

TRSP\_Central application includes four services which are GAP, GATT, DIS (Device Information Service) and TRSP service.

#### ● GAP

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification “[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C\*](#)” for more details.

#### ● GATT

Generic Attribute Profile Service (GATT) is a SIG defined service and please refer to SIG Bluetooth specification “[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part G\*](#)” for more details.

#### ● Device Information Service (DIS)

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification “[\*DIS SPEC V11r00.pdf\*](#)” for more details. The implementation please refer to “ble\_service\_dis.c/ ble\_service\_dis.h”.

#### ● Transparent Service (TRSPS)

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to emulate the serial port over BLE. The procedure of creating a customer service please see “[\*RT58x BLE SDK Service Profile Guide.pdf\*](#)” for more details. The implementation please refer to “ble\_service\_trsps.c/ ble\_service\_trsps.h”.

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.
Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.

Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.
-------------	------------------------------------	--

(intentionally blank)

### 3.1.2 Handle Data Received Over UART

The same as [section 2.2.2](#).

### 3.1.3 Handle Data Transmitted Over BLE

Wait for request to transmit data over BLE by using “ble\_svcs\_trsps\_client\_send()” function and also check the sending status to do the flow control.

```
// TRSPS Central
static void app_central_handler(app_req_param_t *p_param)
{
    ...
    switch (p_param->app_req)
    {
        ...
        case APP_REQUEST_TRSPC_DATA_SEND:
            if ((BLE_TRSP_WRITE_TYPE != BLEGATT_WRITE) && (BLE_TRSP_WRITE_TYPE != BLEGATT_WRITE_WITHOUT_RSP))
            {
                printf("Error write type. \n");
            }
            else
            {
                status = ble_svcs_trsps_client_send(host_id,
                                                    BLE_TRSP_WRITE_TYPE,
                                                    p_profile_info->svcs_info_trsps.client_info.handles.hdl_udatrw01,
                                                    (uint8_t *)g_rx_buffer,
                                                    (g_rx_buffer_length));

                if (status == BLE_ERR_OK)
                {
                    g_data_send_pending = false;
                }
                else
                {
                    g_data_send_pending = true;

                    // re-send
                    app_request_set(host_id, APP_REQUEST_TRSPC_DATA_SEND, false);
                }
            }
            break;
    }
}
```

(intentionally blank)

### 3.1.4 Handle Data Received Over BLE

Implement the callback function “ble\_svcs\_trsps\_evt\_handler()” which initialized in “client\_profile\_init” function to handle the data received over BLE. In this sample code, print the received data out over UART.

```
static ble_err_t client_profile_init(uint8_t host_id)
{
    ble_err_t status = BLE_ERR_OK;
    ble_profile_info_t *p_profile_info = (ble_profile_info_t *)ble_app_link_info[host_id].profile_info;

    // set link's state
    ble_app_link_info[host_id].state = STATE_STANDBY;

    do
    {
        // GAP Related
        // -----
        ...
        // GATT Related
        // -----
        ...
        // DIS Related
        // -----
        ...
        // TRSPS Related
        // -----
        status = ble_svcs_trsps_init(host_id, BLE_GATT_ROLE_CLIENT, &(p_profile_info->svcs_info_trsps), ble_svcs_trsps_evt_handler);
        if (status != BLE_ERR_OK)
        {
            break;
        }
    } while (0);

    return status;
}
```

```
static void ble_svcs_trsps_evt_handler(ble_evt_att_param_t *p_param)
{
    if (p_param->gatt_role == BLE_GATT_ROLE_CLIENT)
    {
        /* ----- Handle event from server ----- */
        switch (p_param->event)
        {
            case BLESERVICE TRSPS UDATNI01 NOTIFY EVENT:
            case BLESERVICE TRSPS UDATNI01 INDICATE EVENT:
            {
                p_param->data[p_param->length] = '\0';
                printf("%s\n", p_param->data);
            }
            break;

            default:
                break;
        }
    }
}
```

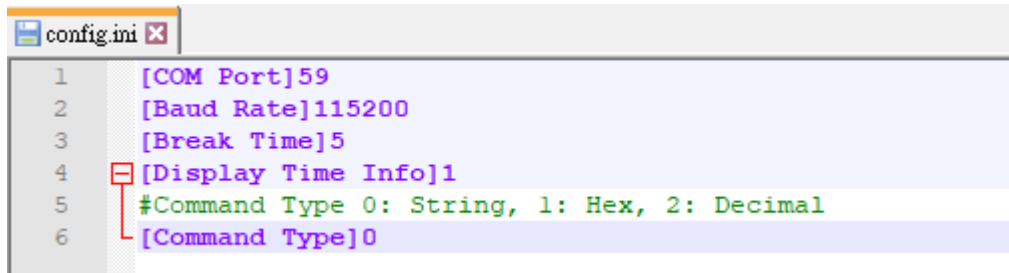
(intentionally blank)

### 3.1.5 Running “TRSP\_Central” Demo

This demo code can be tested with the demo “TRSP\_Peripheral”.

[Preparation]

- DUT:
  - Setup “UART BREAK” Tool. The settings are as follows.



```

1 [COM Port]59
2 [Baud Rate]115200
3 [Break Time]5
4 [Display Time Info]1
5 #Command Type 0: String, 1: Hex, 2: Decimal
6 [Command Type]0
  
```

- Setup UART Environment: baud rate=115200, power-on the device and the initial log will show on the screen.



```

-----
BLE TRSP (C) demo: start...
-----
BLE stack initial...
Scanning...
  
```

- Test with device in “TRSP\_Peripheral” demo.

(intentionally blank)



## [Testing]

### 1. Scan & Connection:

TRSP\_Central demo will enable scan and create BLE connection if finding the target device. The central device will do MTU size exchange, CCCD configuration and get DIS manufacturer name and Firmware revision after connecting with the target device. The logs are shown as below:

```
-----
BLE TRSP (C) demo: start...
-----
BLE stack initial...
Scanning...
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -68]
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -73]
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -69]
Found [Name:AT_DEMO] [Address: c0:2e:b3:96:3f:ad] [RSSI= -55]
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -70]
Found [Name:AT_DEMO] [Address: c0:2e:b3:96:3f:ad] [RSSI= -59]
Found [Name:AT_DEMO] [Address: c0:89:df:2e:78:51] [RSSI= -73]
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -72]
Found [Name:AT_DEMO] [Address: c0:89:df:2e:78:51] [RSSI= -72]
Found [Name:AT_DEMO] [Address: c0:89:df:2e:78:51] [RSSI= -66]
Found [Name:Data_Rate] [Address: c6:75:74:73:72:71] [RSSI= -73]
Found [Name:TRSP_DEMO] [Address: c6:25:24:23:22:21] [RSSI= -29]
Idle.
Connected, ID=0, Connected to c6:25:24:23:22:21
Data length changed, ID: 0
MaxTxOctets: 251 MaxTxTime:2120
MaxRxOctets: 251 MaxRxTime:2120
DB Parsing completed, ID:0 status:0x00
Data length changed, ID: 0
MaxTxOctets: 27 MaxTxTime:328
MaxRxOctets: 251 MaxRxTime:2120
MTU Exchanged, ID:0, size: 23
Manufacturer name: BLE
Ready to TX/RX data to/from the connected server.
FW rev.: 1.0
```

### 2. Transmit Data over BLE.

Typing strings with an Enter (\n) to transmit data to the connected peer device. The “transmitted data” will show on the UART of “TRSP\_Peripheral”.

[TRSP\_Central UART] type string and press enter

```
[BLE_HOST_TASK] BLE_GAP_EVT_CONN_COMPLETE
Connected, ID=0, Connected to c6:25:24:23:22:21
DB Parsing completed, ID:0 status:0x00
MTU Exchanged, ID:0, size: 23
Manufacturer name: BLE
Ready to TX/RX data to/from the connected server.
FW rev.: 1.0
transmit data test
```

[TRSP\_Peripheral UART] show receive data from central device

```
Advertising...  
[BLE_HOST_TASK] BLE_GAP_EVT_CONN_COMPLETE  
Connected, ID=0, Connected to c6:45:44:43:42:41  
MTU Exchanged, ID:0, size: 23  
transmit data test
```

### 3. Receive Data over BLE.

The procedures are the same as “Transmit Data over BLE” but typing string with an Enter(\n) on “TRSP\_Peripheral” UART instead.

[TRSP\_Peripheral UART] type string and press enter

```
Advertising...  
[BLE_HOST_TASK] BLE_GAP_EVT_CONN_COMPLETE  
Connected, ID=0, Connected to c6:45:44:43:42:41  
MTU Exchanged, ID:0, size: 23  
transmit data test  
receive data test
```

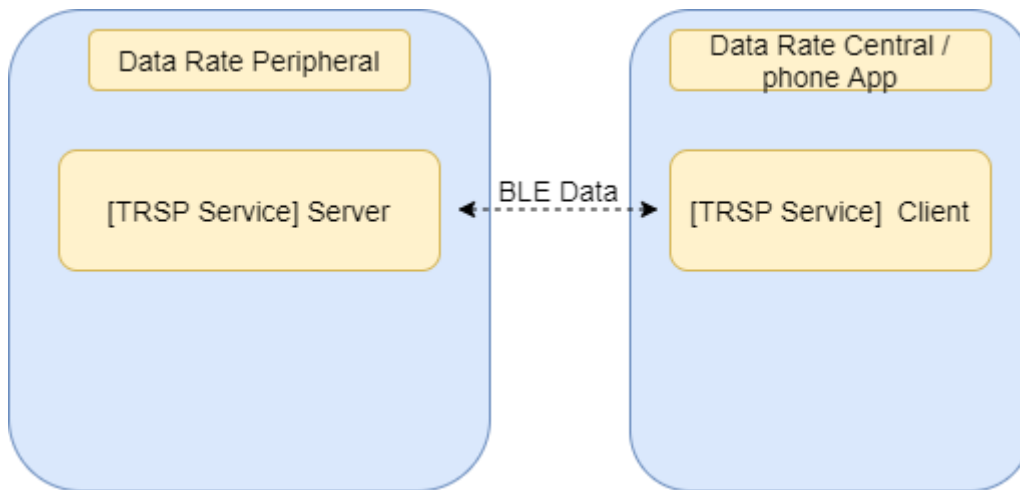
[TRSP\_Central UART] show receive data from central device

```
Connected, ID=0, Connected to c6:25:24:23:22:21  
DB Parsing completed, ID:0 status:0x00  
MTU Exchanged, ID:0, size: 23  
Manufacturer name: BLE  
Ready to TX/RX data to/from the connected server.  
FW rev.: 1.0  
transmit data test  
receive data test
```

(intentionally blank)

## 3.2 Data\_Rate\_Central

The data rate central demonstrates the BLE data transferred with central role through the BLE TRSP service. User could demonstrate data transfer effectiveness via data rate peripheral.



### 3.2.1 BLE Services

Data\_Rate\_Central application includes four services which are GAP, GATT, DIS (Device Information Service) and TRSP service.

- **GAP**

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification "[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C\*](#)" for more details.

- **GATT**

Generic Attribute Profile Service (GATT) is a SIG defined service and please refer to SIG Bluetooth specification "[\*BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part G\*](#)" for more details.

- **Device Information Service (DIS)**

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification "[\*DIS\\_SPEC\\_V11r00.pdf\*](#)" for more details. The implementation please refer to "ble\_service\_dis.c/ ble\_service\_dis.h".

### ● Transparent Service (TRSPS)

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to emulate the serial port over BLE. The procedure of creating a customer service please see “[RT58x BLE SDK Service Profile Guide.pdf](#)” for more details. The implementation please refer to “ble\_service\_trsps.c/ ble\_service\_trsps.h”.

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.
Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.
Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.

### 3.2.2 Handle Data Rate Command Received Over BLE

Implement the callback function “ble\_svcs\_trsps\_evt\_handler()” which initialized in “client\_profile\_init” function to handle the data received over BLE. In this sample code, multiple data rate recognition strings are defined to represent different commands, which can be used to test the maximum transfer speed under different conditions.

```
// Data rate cmd identification
// set data rate parameters from central device
#define SET_PARAM_STR          "set_param"
// transmit data from central device, test C->P data rate
#define PRX_TEST_STR           "pRxtest"
// Receive from peripheral device, test P->C data rate
#define PTX_TEST_STR           "pTxtest"
```

#### Defined UART command Identifiers:

```

help      :Help
param     :Get all parameters.
lAddr     :Set local device address. (=112233445566 means 66:55:44:33:22:11)
tAddr     :Set target device address. (=112233445566 means 66:55:44:33:22:11)
createCon :Set create connection.
cancelCon :Set cancel the On-going connection.
disconnect:Terminate the connection.
conEnc    :Enable encrypted link. Re-connect to reset to unencrypted link.
conInt    :Set connection interval in unit(1.25ms). (=6 means 7.5ms)
phy       :Set PHY. =1:BLE_PHY_1M; =2:BLE_PHY_2M. =3:BLE_CODED_PHY_S2. =4:BLE_CODED_PHY_S8.
dataLen   :Set data length per packet, the range = 20~244
testTxLen :Set total TX test length.
testMode  :Set test mode. =1:TX; =2:RX.
testStart :Set to start data rate test.
testStop  :Set to stop data rate test.
channelmapSet:Set channel map. (=FFFFFFFFF channel all open.)
channelmapRead:Read channel map.

```

- **“help”**  
List of all supported UART commands for users to query the corresponding command string.
- **“param”**  
Get the default values of the current data rate test, including mac address, destination mac address, connection interval, PHY setting, packet size, amount of data tested and test mode.
- **“lAddr”**  
List the device mac address.
- **“tAddr”**  
List the target mac address.
- **“createCon”**  
The establish connection command causes the data rate central role to attempt to establish a connection with a data rate peripheral role.
- **“cancelCon”**  
Cancel the create connection.
- **“disconnect”**  
Send a disconnect command to release the current connection status.
- **“conEnc”**  
Enable BLE encryption. When encryption is complete, the data will be protected with BLE encryption during the test.
- **“conInt”**  
Set the connection interval.
- **“phy”**

Set the BLE PHY.

- **“dataLen”**

Set the maximum amount of packet data for the test.

- **“testTxLen”**

Set the total amount of data to be tested.

- **“testMode”**

Set the test mode to transmit or receiver.

- **“testStart”**

Start the data rate test.

- **“testStop”**

Stop the data rate test.

- **“channelmapSet”**

Set the channel classification of BLE.

- **“channelmapRead”**

Read the BLE channel map.

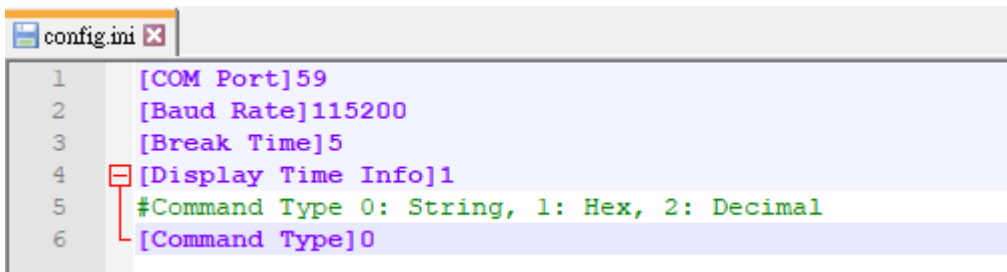
(intentionally blank)

### 3.2.3 Running “Data\_Rate\_Central” Demo

This demo code needs to be tested with the demo “Data\_Rate\_Peripheral”.

### [Preparation]

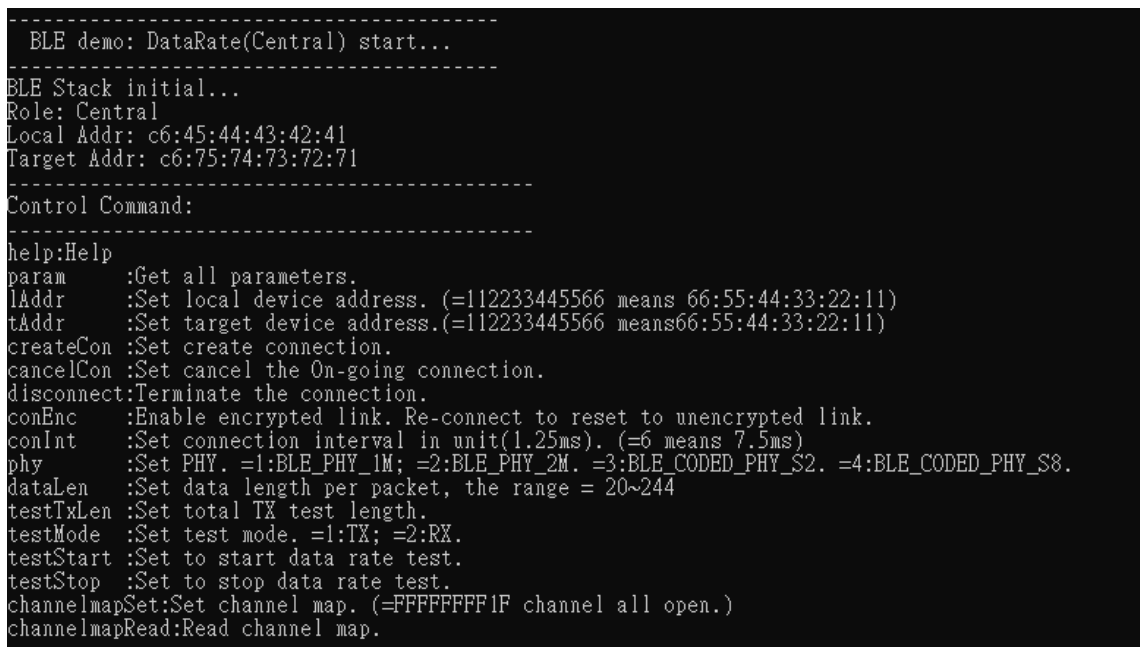
- DUT: (Data Rate Central)
- Setup “UART BREAK” Tool. The settings are as follows.



```

1 [COM Port]59
2 [Baud Rate]115200
3 [Break Time]5
4 [Display Time Info]1
5 #Command Type 0: String, 1: Hex, 2: Decimal
6 [Command Type]0
  
```

- Setup UART Environment: baud rate=115200, power-on the device and the initial log will show on the screen.



```

-----
BLE demo: DataRate(Central) start...
-----
BLE Stack initial...
Role: Central
Local Addr: c6:45:44:43:42:41
Target Addr: c6:75:74:73:72:71
-----
Control Command:
-----
help:Help
param :Get all parameters.
lAddr :Set local device address. (=112233445566 means 66:55:44:33:22:11)
tAddr :Set target device address.(=112233445566 means66:55:44:33:22:11)
createCon :Set create connection.
cancelCon :Set cancel the On-going connection.
disconnect:Terminate the connection.
conEnc :Enable encrypted link. Re-connect to reset to unencrypted link.
conInt :Set connection interval in unit(1.25ms). (=6 means 7.5ms)
phy :Set PHY. =1:BLE_PHY_1M; =2:BLE_PHY_2M. =3:BLE_CODED_PHY_S2. =4:BLE_CODED_PHY_S8.
dataLen :Set data length per packet, the range = 20~244
testTxLen :Set total TX test length.
testMode :Set test mode. =1:TX; =2:RX.
testStart :Set to start data rate test.
testStop :Set to stop data rate test.
channelmapSet:Set channel map. (=FFFFFFFFIF channel all open.)
channelmapRead:Read channel map.
  
```

- Test with device in “Data\_Rate\_Peripheral” demo.

(intentionally blank)

### [Testing]

#### 1. Create Connection:

After confirming the mac address of the target, enter the connection establishment to complete the BLE connection. The central device will perform MTU size exchange, CCCD configuration and get DIS manufacturer name and firmware version after connecting with the target device. The logs are displayed as follows.

```
Recv cfm status 0
Ready to TX/RX data to/from the connected server.
FW rev.: 1.0
```

## 2. Set test conditions.

Enter the UART command with test condition values to set the test parameters.

## 3. Start test.

Enter the “testStart” command to start the test. When the test is finished, the transmission rate of the test can be obtained from the Rx side.

```
Test length = 1048712
Start RX...
Stop RX
Total Rx Received Time: 90857 ms
Total Rx Received 1048712 Bytes
Rx Through: 92339.567 bps
```

(intentionally blank)

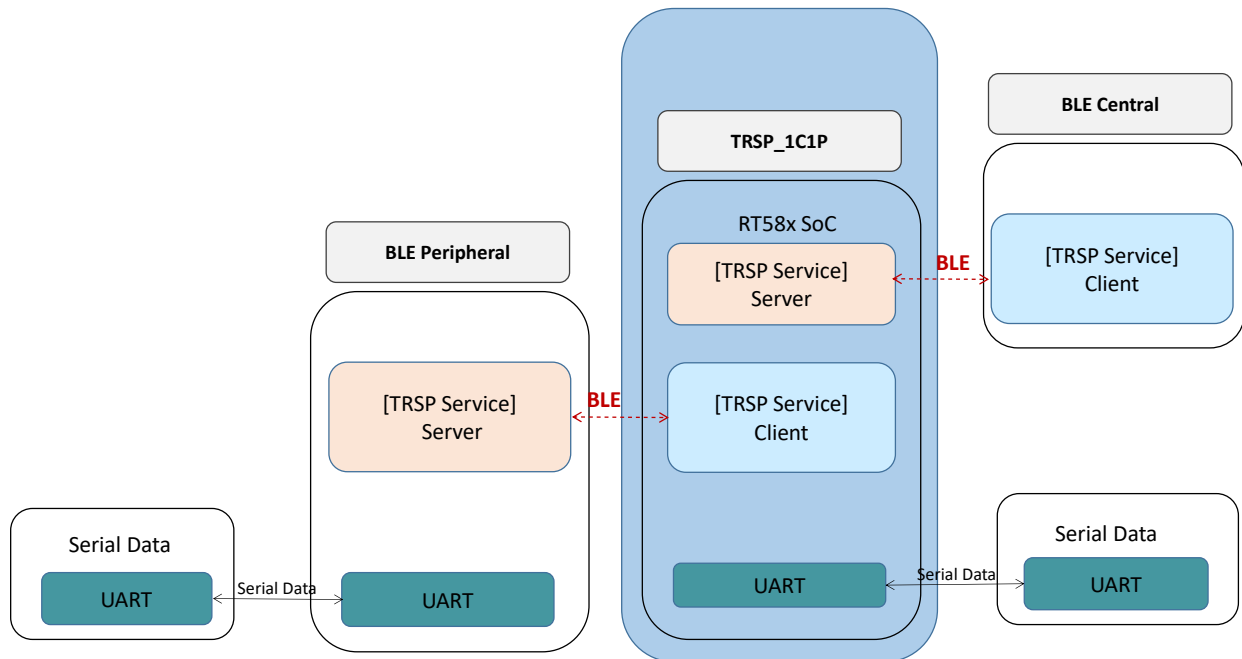
## 4. BLE Central and Peripheral Demo Application

The BLE central and peripheral demo project is TRSP\_1C\_1P.



## 4.1 TRSP\_1C1P

Transparent (TRSP) UART demonstrates the serial data transferred via BLE TRSP service. This project demonstrates TRSP with both central role and peripheral role. It could connect to the TRSP server and be connected to the TRSP client.



### 4.1.1 BLE Services

TRSP\_Central application includes four services which are GAP, GATT, DIS (Device Information Service) and TRSP service.

- **GAP**

GAP Service is a SIG defined service and please refer to SIG Bluetooth specification "[BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part C](#)" for more details.

- **GATT**

Generic Attribute Profile Service (GATT) is a SIG defined service and please refer to SIG Bluetooth specification "[BLUETOOTH SPECIFICATION Version 5.0 | Vol 3, Part G](#)" for more details.

- **Device Information Service (DIS)**

Device Information Service (DIS) is a SIG defined service and please refer to SIG Bluetooth specification "[\*DIS SPEC V11r00.pdf\*](#)" for more details. The implementation please refer to "ble\_service\_dis.c/ ble\_service\_dis.h".

- **Transparent Service (TRSPS)**

Transparent Service (TRSP Service) is a proprietary customer defined service which uses to emulate the serial port over BLE. The procedure of creating a customer service please see "[\*RT58x BLE SDK Service Profile Guide.pdf\*](#)" for more details. The implementation please refer to "ble\_service\_trsps.c/ ble\_service\_trsps.h".

Service	UUID	Description
TRSP Service	0x00112233445566778899AABBCCDDEEFF	Serial data transferred via BLE.
Characteristic	UUID	Description
Read	0x101112131415161718191A1B1C1D1E1F	The client can get the static data from this characteristic.
Notify	0x303132333435363738393A3B3C3D3E3F	The peer device can send data to client by writing this characteristic.
Read/ Write	0x505152535455565758595A5B5C5D5E5F	The peer device can send data to the client or receive data from the client via this characteristic.

(intentionally blank)

## 4.1.2 Command List

```
-----
BLE TRSP 1C1P demo: start...
-----
*****
1C1P Host id List
Role : Central, Host id = 0
Role : Peripheral, Host id = 1
*****
BLE stack initial...
Help          :Help.
Send          :Transmit data. ex: enter Send=0,5,12345 => Host_Id=0, Len=5, Data=12345.
EnAdv         :Enable advertising.
DisAdv        :Disable advertising.
EnScan        :Enable scan.
DisScan       :Disable scan.
Disconnect    :Disable connection (command with host id.), ex: enter Disconnect=0.
```

- **“Help”**  
List of all supported UART commands for users to query the corresponding command string.
- **“Send”**  
Send the BLE data by host id with data length.
- **“EnAdv”**  
Enable Advertising.
- **“DisAdv”**  
Disable Advertising.
- **“EnScan”**  
Enable Scan.
- **“DisScan”**  
Disable Scan.
- **“Disconnect”**  
Send a disconnect command by host id to release the current connection status.

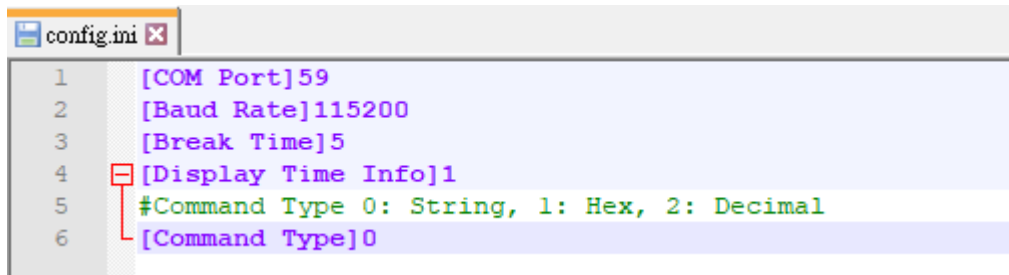
(intentionally blank)

### 4.1.3 Running “TRSP\_1C1P” Demo

This demo code can be tested with the demo “TRSP\_Peripheral” and smart phone or “TRSP\_Peripheral”.

[Preparation]

- DUT:
- Setup “UART BREAK” Tool. The settings are as follows.



- Setup UART Environment: baud rate=115200, power-on the device and the initial log will show on the screen.

```

-----
BLE TRSP 1C1P demo: start...
-----
*****
1C1P Host id List
Role : Central, Host id = 0
Role : Peripheral, Host id = 1
*****
BLE stack initial...
Help          :Help.
Send          :Transmit data. ex: enter Send=0,5,12345 => Host_Id=0, Len=5, Data=12345.
EnAdv         :Enable advertising.
DisAdv        :Disable advertising.
EnScan        :Enable scan.
DisScan       :Disable scan.
Disconnect    :Disable connection (command with host id.), ex: enter Disconnect=0.
  
```

- Test with device in “TRSP\_Peripheral” demo.

## [Testing]

### 1. Scan & Connection:

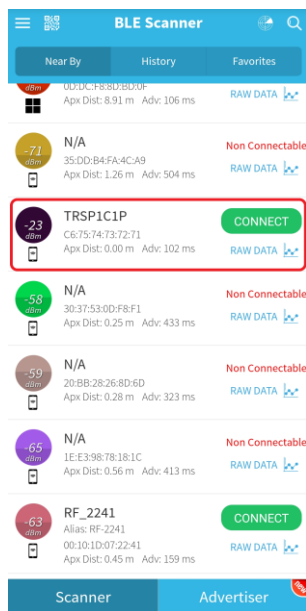
TRSP\_1C1P demo can follow UART command to enable scan, advertisement. It automatically creates BLE connection when finding the target device. The TRSP client will do MTU size exchange, CCCD configuration and get DIS manufacturer name and Firmware revision after connecting with the target device. The logs are shown as below:

### 2. TRSP Client Transmit or Receive Data over BLE.

Please refer to [section 3.1.5](#).

### 3. TRSP server Test.

- Use mobile phone as a TRSP client.
- Open “BLE Scanner” app and click “Scan” button to start scanning.
- Connect with the desired BLE device.



- Click “TRSP” service. Enable NOTIFY (it may be shown as N icon, click it to enable/disable) and you will send uart data and see the data from APP.

< TRSP1C1P

DISCONNECT

Status: CONNECTED  
NOT BONDED

DEVICE INFORMATION

0x180A  
PRIMARY SERVICE

GENERIC ACCESS

0x1800  
PRIMARY SERVICE

GENERIC ATTRIBUTE

0x1801  
PRIMARY SERVICE

CUSTOM SERVICE

00112233-4455-6677-8899-AABBCCDDEEFF  
PRIMARY SERVICE

CUSTOM CHARACTERISTIC

UUID: 10111213-1415-1617-1819-1A1B1C1D1E1F  
Properties: READ

CUSTOM CHARACTERISTIC

UUID: 30313233-3435-3637-3839-3A3B3C3D3E3F  
Properties: NOTIFY,INDICATE  
Value: 0x303132

Descriptors:

Client Characteristic Configuration  
UUID: 0x2902

CUSTOM CHARACTERISTIC

UUID: 50515253-5455-5657-5859-5A5B5C5D5E5F  
Properties: READ,WRITE,WRITE\_NO\_RESPONSE  
Write Type: WRITE REQUEST

(intentionally blank)

## 5. iBeacon Demo Application

The iBeacon Demo is an example that implements a transmitter beacon. And it follows the iBeacon format developed by apple.

The beacon broadcast information is implemented using an advertising package that contains the following items:

- A 128-bit UUID to identify the beacon's provider.
- An arbitrary Major value for coarse differentiation between beacons.
- An arbitrary Minor value for fine differentiation between beacons.
- The RSSI value of the beacon measured at 1-meter distance, which can be used for estimating the distance from the beacon.

Modify the sample code in the ADV data to beacon format to complete a simple beacon application.

```
//Adv data format. Set according to user select profile. See Bluetooth Spec. Ver5.0 [Vol 3], Part C, Section 11
uint8_t adv_data[] =
{
    0x02, GAP_AD_TYPE_FLAGS, 0x06,           // LE General Discoverable Mode
    0x1A, GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA, // Manufacturer data length, Type: Manufacturer data
    0x4C, 0x00,                               // 0x004C: Apple, Inc.
    0x02,                                     // iBeacon advertisement indicator
    0x15,                                     // Data length
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, // Proximity UUID part 1
    0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, // Proximity UUID part 2
    0x00, 0x11,                               // Major ID
    0x00, 0x22,                               // Minor ID
    0xD8,                                     // Measured TX Power (-40dBm RSSI: 256 - 40 = 0xD8)
};
```

This application can be used as a starting point for writing an iBeacon application. iBeacon application procedures and specifications are available at “<https://developer.apple.com/ibeacon/>”. Once the specification is obtained, this application can be modified to fit the iBeacon requirements according to the specification.

(intentionally blank)

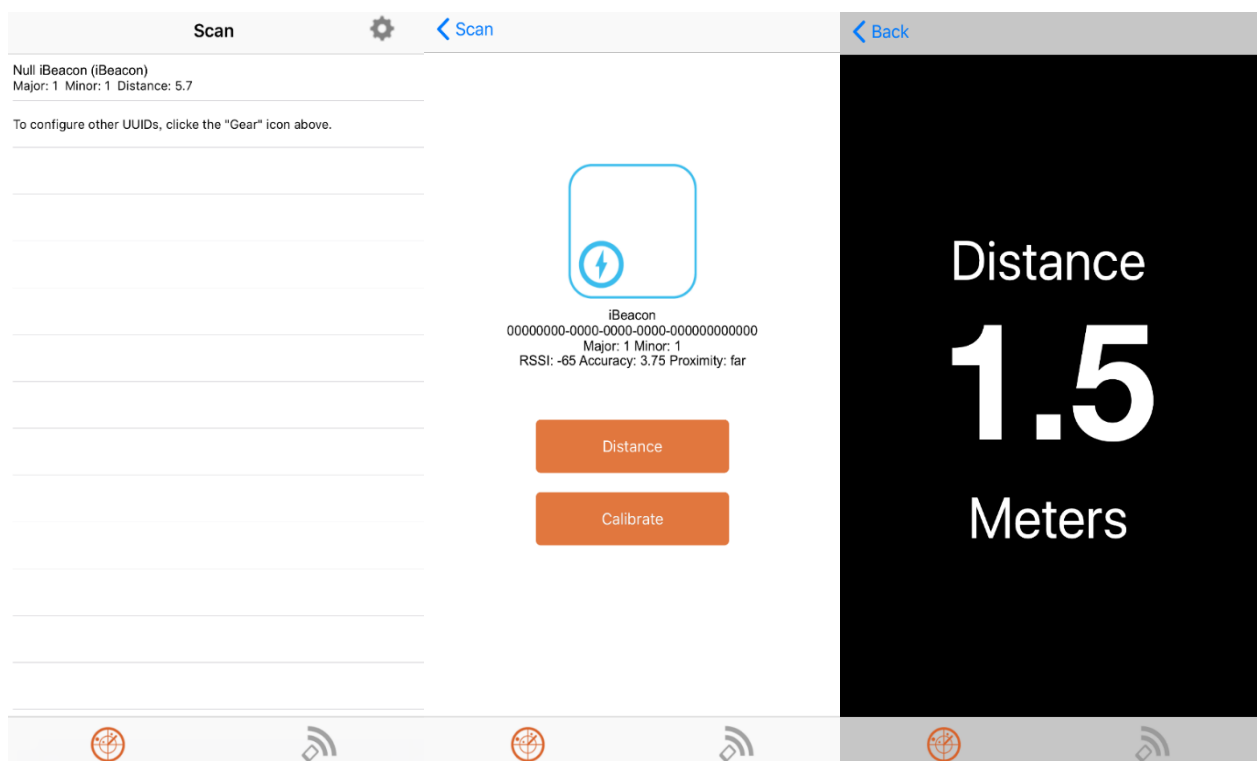
### 5.1.1 Running “iBeacon” Demo

## [Testing]

- Modified the Company ID to 0x004C (Apple.Inc) and Major ID/Minor ID as shown in the figure below.

```
//Adv data format. Set according to user select profile. See Bluetooth Spec. Ver5.0 [Vol 3], Part C, Section 11
uint8_t adv_data[] =
{
    0x02, GAP_AD_TYPE_FLAGS, 0x06,           // LE General Discoverable Mode
    0x1A, GAP_AD_TYPE_MANUFACTURER_SPECIFIC_DATA, // Manufacturer data length, Type: Manufacturer data
    0x4C, 0x00,                               // 0x004C: Apple, Inc.
    0x02,                                     // iBeacon advertisement indicator
    0x15,                                     // Data length
    0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, // Proximity UUID part 1
    0x09, 0x10, 0x11, 0x12, 0x13, 0x14, 0x15, 0x16, // Proximity UUID part 2
    0x00, 0x11,                               // Major ID
    0x00, 0x22,                               // Minor ID
    0xD8,                                     // Measured TX Power (-40dBm RSSI: 256 - 40 = 0xD8)
};
```

- Using “Locate Beacon” app to search beacon devices. It can get beacon data and distance reference values.



## 6. AT Command Demo Application



The AT command demo can be used to simply test the various features provided by the BLE library, including advertising, scanning, connectivity, etc. It also helps to understand how to use the provided API to speed up the development process for developers.

## 6.1 AT Command List

The Demo supports the following AT command set. Please note that some AT commands are only available in specified BLE state. If the AT command runs in invalid BLE state, it will return an error directly.

The following figure shows the initial message, then you can send the AT command message.

```
-----
BLE AT command demo: start...
-----
*****
BLE Host id List
Role : Central, Host id = 0
Role : Peripheral, Host id = 1
*****
BLE stack initial...

```

***It is important to note here that most of the application demo cases are done in power saving mode. If you need to use UART AT command, you need to send "Break" signal to 2ms to wake up and then execute the command.***

(intentionally blank)

### 6.1.1 Common Command

- +DADDR

- +DADDR?

Get the default target address for +CRCON

```
AT+DADDR?
at_queue->Push(+DADDR?)
1 at_queue->pop(+DADDR?)
do cmd : +DADDR
addr = 00:00:00:00:00:00, addr type = 0
OK
```

- +DADDR=<addr>

- +DADDR=<num>,<addr>

Set the default target address for +CRCON

<num>: the type of default target address

<addr>: the default target address

Format: XX:XX:XX:XX:XX:XX

ex. 01:02:03:04:05:FF, addr[0] = 0xFF, addr[5] = 0x01

```
AT+DADDR=11:22:33:44:55:66
at_queue->Push(+DADDR=11:22:33:44:55:66)
1 at_queue->pop(+DADDR=11:22:33:44:55:66)
do cmd : +DADDR
OK

AT+DADDR=0,11:22:33:44:55:66
at_queue->Push(+DADDR=0,11:22:33:44:55:66)
1 at_queue->pop(+DADDR=0,11:22:33:44:55:66)
do cmd : +DADDR
OK
```

(intentionally blank)

- +DEVADDR

- +DEVADDR?

Get the address of device.

```
AT+DEVADDR?  
at_queue->Push(+DEVADDR?)  
1 at_queue->pop(+DEVADDR?)  
do cmd : +DEVADDR  
addr = d1:d2:c3:c4:e5:c6  
OK
```

■ +DEVADDR=<addr>

Set the address of device.

<addr>: the address of device

format: XX:XX:XX:XX:XX:XX

ex.01:02:03:04:05:FF, addr[0] = 0xFF, addr[5] = 0x01

```
AT+DEVADDR=11:33:55:77:99:c0  
at_queue->Push(+DEVADDR=11:33:55:77:99:c0)  
1 at_queue->pop(+DEVADDR=11:33:55:77:99:c0)  
do cmd : +DEVADDR  
OK
```

(intentionally blank)

● +DEVADDRTYPE

■ +DEVADDRTYPE?

Get the address type of device.

```
AT+DEVADDRTYPE?
at_queue->Push(+DEVADDRTYPE?)
1 at_queue->pop(+DEVADDRTYPE?)
do cmd : +DEVADDRTYPE
addr type = 1
OK
```

- +DEVADDRTYPE=<num>

Set the address type of device.

<num> : the address type of device

0: public address

1: random address

notice

If BLE Address Type is set to random address, addr[5] >= 0xC0 (the two most significant bits of the address shall be equal to 1).

```
AT+DEVADDRTYPE=1
at_queue->Push(+DEVADDRTYPE=1)
1 at_queue->pop(+DEVADDRTYPE=1)
do cmd : +DEVADDRTYPE
OK
```

- +IBOND

- +IBOND

Initial BLE bonding information.

```
AT+IBOND
at_queue->Push(+IBOND)
1 at_queue->pop(+IBOND)
do cmd : +IBOND
OK
```

(intentionally blank)

- +RESET

- +RESET

Reset Device.

```
AT+RFRST
at_queue->Push(+RFRST)
1 at_queue->pop(+RFRST)
do cmd : +RFRST

-----
  BLE AT command demo: start...
-----
*****
BLE Host id List
Role : Central, Host id = 0
Role : Peripheral, Host id = 1
*****
BLE stack initial...
█
```

(intentionally blank)

- +ALLPARAM
- +ALLPARAM?

Read the all param.

```

AT+ALLPARAM?
at_queue->Push(+ALLPARAM?)
1 at_queue->pop(+ALLPARAM?)
do cmd : +ALLPARAM
device_addr = 11:33:55:77:99:c0 (type=1)
adv_param:
    adv_type = 0
    adv_interval_min = 160
    adv_interval_max = 160
    adv_peer_addr_param = 00:00:00:00:00:00 (type=0)
    adv_channel_map = 7
    adv_filter_policy = 0
scan_param:
    scan_type = 1
    scan_interval = 160
    scan_window = 160
    scan_filter_policy = 0
adv_data = 02:01:05:03:ff:12:34
scan_rsp = 0b:09:41:54:5f:43:6f:6d:6d:61:6e:64
connection param:
0 :
    min_conn_interval = 60
    max_conn_interval = 60
    periph_latency = 0
    supv_timeout = 1000
1 :
    min_conn_interval = 60
    max_conn_interval = 60
    periph_latency = 0
    supv_timeout = 1000
create connection param:
    scan_interval = 160
    scan_window = 160
    init_filter_policy = 0
    create connection param - conn_param:
        min_conn_interval = 60
        max_conn_interval = 60
        periph_latency = 0
        supv_timeout = 1000
preferred_tx_data_length = 27
preferred_mtu_size:0 :
    preferred_mtu_size: = 247
1 :
    preferred_mtu_size: = 247
OK

```

● +HELP

■ +HELP

List all at command.

```
AT+HELP
at_queue->Push(+HELP)
1 at_queue->pop(+HELP)
do cmd : +HELP
The Cmd List :
AT+RFRST: rf reset
AT+HELP: help
AT+ROLE: role
AT+ALLPARAM: read the all param
AT+SLEEP: sleep
AT+WAKEUP: wake up
AT+READFALS: read filter accept list size
AT+ADDFA: add filter accept list
AT+CLEARFAL: clear filter accept list
AT+REMOVEFAL: remove filter accept list
AT+ENADV: enable adv
AT+DISADV: disable adv
AT+ADVINT: adv interval
AT+ADVCHMAP: adv channel map
AT+ADVTYPE: adv type
AT+ADVDATA: adv data
AT+SCANRSP: scan response
AT+DISCON: disconnect
AT+CONINT: connection interval
AT+CONLAT: connection latency
AT+CONSUTMO: connection supervision timeout
AT+CONPARAM: connection param
AT+ENSCAN: enable scan
AT+DISSCAN: disable scan
AT+SCANTYPE: scan type
AT+SCANINT: scan interval
AT+SCANWIN: scan window
AT+DEVADDR: device address
AT+DEVADDRTYPE: device address type
AT+DADDR: default target address
AT+PLDATALEN: payload data length
AT+PREPLDATALEN: preferred payload data length
AT+PFMTUS: preferred mtu size
AT+RMTUS: read mtu size
AT+PHY: phy
AT+READPHY: read phy
AT+READRSSI: read rssi
AT+SBOND: set bonding flag
AT+SEC: security request
AT+RSCCCD: restore CCCD
AT+IBOND: init bonding info
AT+EXMTUS: exchange mtu size
AT+READ: read
AT+WRITE: write
AT+WRITENRSP: write without response
AT+READCCCD: read CCCD
AT+WRITECCCD: write CCCD
```

● +ROLE

■ +ROLE?

Read the Role message.

```
AT+ROLE?
at_queue->Push(+ROLE?)
1 at_queue->pop(+ROLE?)
do cmd : +ROLE
Host Id:0, Central
Host Id:1, Peripheral
OK
```

- +SLEEP

- +SLEEP

Set the device to go into sleep mode when it is idle.

```
AT+SLEEP
at_queue->Push(+SLEEP)
1 at_queue->pop(+SLEEP)
do cmd : +SLEEP
OK
```

- +WAKEUP

- +WAKEUP

Set the device to go into Idle mode when it is not busy.

```
AT+WAKEUP
at_queue->Push(+WAKEUP)
1 at_queue->pop(+WAKEUP)
do cmd : +WAKEUP
OK
```

- +DEVAP

- +DEVAP=<num>

Set BLE GAP appearance

<num> : appearance

```
AT+DEVAP=200
at_queue->Push(+DEVAP=200)
1 at_queue->pop(+DEVAP=200)
do cmd : +DEVAP
OK
```

- +DEVNAME

- +DEVNAME=<string>



set GAP device name

<string> : the GAP device name data

```
AT+DEVNAME=AT Command
at_queue->Push(+DEVNAME=ATCommand)
1 at_queue->pop(+DEVNAME=ATCommand)
do cmd : +DEVNAME
OK
```

- +PCONPAR

- +PCONPAR=<num1>, <num2>, <num3>, <num4>

Set GAP perfer connection parameter

<num1> : connIntervalMin

<num2> : connIntervalMax

<num3> : connLatency

<num4> : connSupervisionTimeout

```
AT+PCONPAR=6,6,0,1000
at_queue->Push(+PCONPAR=6,6,0,1000)
1 at_queue->pop(+PCONPAR=6,6,0,1000)
do cmd : +PCONPAR
OK
```

- +READFALS

- +READFALS

Read filter accept list size.

```
AT+READFALS
at_queue->Push(+READFALS)
1 at_queue->pop(+READFALS)
do cmd : +READFALS
OK

Read filter accept list size = 8
```

(intentionally blank)

- +CLEARFAL

- +CLEARFAL

Clear filter accept list.

```
AT+CLEARFAL
at_queue->Push(+FAT+CLEARFAL)
1 at_queue->pop(+FAT+CLEARFAL)
command [+FAT+CLEARFAL] doesn't exist
ERROR CMD_NOT_SUPPORTED
```

- +ADDFAL

- +ADDFAL=<num>,<addr>

Add device to filter accept list.

<num> : the type of default address

<addr> : the default address

format : XX:XX:XX:XX:XX:XX

ex. 01:02:03:04:05:FF, addr[0] = 0xFF, addr[5] = 0x01

notice

If BLE Address Type is set to random address, addr[5] >= 0xC0 (the two most significant bits of the address shall be equal to 1).

```
AT+ADDFAL=0,22:33:44:55:66:77
at_queue->Push(+ADDFAL=0,22:33:44:55:66:77)
1 at_queue->pop(+ADDFAL=0,22:33:44:55:66:77)
do cmd : +ADDFAL
OK
```

(intentionally blank)

- +REMOVEFAL

- +REMOVEFAL=<num>,<addr>

Remove device from filter accept list.

<num> : the type of default address

<addr> : the default address

format : XX:XX:XX:XX:XX:XX

ex. 01:02:03:04:05:FF, addr[0] = 0xFF, addr[5] = 0x01

notice

If BLE Address Type is set to random address, addr[5] >= 0xC0 (the two most significant bits of the address shall be equal to 1).

```
AT+REMOVEFAL=0,22:33:44:55:66:77
at_queue->Push(+REMOVEFAL=0,22:33:44:55:66:77)
1 at_queue->pop(+REMOVEFAL=0,22:33:44:55:66:77)
do cmd : +REMOVEFAL
OK
```

- +PFMTUS

- +PFMTUS

Set preferred MTU size of specific host ID

<num1>: host ID

range : 0-1

<num2>: preferred rx mtu size

range : 23-247

```
AT+PFMTUS=1,247
at_queue->Push(+PFMTUS=1,247)
1 at_queue->pop(+PFMTUS=1,247)
do cmd : +PFMTUS
OK
```

(intentionally blank)

- +SBOND

- +SBOND=<num>

set BLE Bonding Flag

+SBOND = <num>

<num>: flag

```
AT+SBOND=1
at_queue->Push(+SBOND=1)
1 at_queue->pop(+SBOND=1)
do cmd : +SBOND
OK
```

(intentionally blank)

### 6.1.2 Advertising Command

- +ADVTTYPE

- +ADVTTYPE?

Get advertising type.

```
AT+ADVTTYPE?
at_queue->Push(+ADVTTYPE?)
1 at_queue->pop(+ADVTTYPE?)
do cmd : +ADVTTYPE
0
OK
```

- +ADVTTYPE=<num>

set advertising type

<num> : the advertising type

0: Connectable and scannable undirected advertising

1: Connectable directed advertising

2: Scannable undirected advertising

3: Non-Connectable undirected advertising

```
AT+ADVTTYPE=2
at_queue->Push(+ADVTTYPE=2)
1 at_queue->pop(+ADVTTYPE=2)
do cmd : +ADVTTYPE
OK
```

- +ADVTTYPE=1,<num>,<addr>

when set advertising type to 1(direct address mode), it has to provide an address as a target

<num> : the address type of device

0: public address

1: random address

<addr> : the address of device.

format : XX:XX:XX:XX:XX:XX

ex.01:02:03:04:05:FF, addr[0] = 0xFF, addr[5] = 0x01

```
AT+ADVTTYPE=1,0,22:33:44:55:66:77
at_queue->Push(+ADVTTYPE=1,0,22:33:44:55:66:77)
1 at_queue->pop(+ADVTTYPE=1,0,22:33:44:55:66:77)
do cmd : +ADVTTYPE
OK
```

- +ADVINT

- +ADVINT?

Get the advertising interval.

```
AT+ADVINT?  
at_queue->Push(+ADVINT?)  
1 at_queue->pop(+ADVINT?)  
do cmd : +ADVINT  
min:160 max:160  
OK
```

- +ADVINT=<num1>,<num2>

Set the advertising interval

<num1> : the minimum advertising interval

range : 32-16384

interval = <num> \* 0.625ms

<num2> : the maximum advertising interval

range : 32-16384

interval = <num> \* 0.625ms

notice:

<num1> must smaller <num2>

```
AT+ADVINT=100,100  
at_queue->Push(+ADVINT=100,100)  
1 at_queue->pop(+ADVINT=100,100)  
do cmd : +ADVINT  
OK
```

(intentionally blank)

- +ADVCHMAP

- +ADVCHMAP?

Get the advertising channel.

```
AT+ADVCHMAP?  
at_queue->Push(+ADVCHMAP?)  
1 at_queue->pop(+ADVCHMAP?)  
do cmd : +ADVCHMAP  
7  
OK
```

■ +ADVCHMAP=<num>

Set the advertising channel.

<num> : the advertising channel

range : 0-7

bit1: channel 37

bit2: channel 38

bit3: channel 39

```
AT+ADVCHMAP=3  
at_queue->Push(+ADVCHMAP=3)  
1 at_queue->pop(+ADVCHMAP=3)  
do cmd : +ADVCHMAP  
OK
```

(intentionally blank)

● +ADVDATA

■ +ADVDATA?

Get the advertising data.

```
AT+ADVDATA?
at_queue->Push(+ADVDATA?)
1 at_queue->pop(+ADVDATA?)
do cmd : +ADVDATA
02:01:05:03:ff:12:34
OK
```

■ +ADVDATA=<string>

Set the advertising data.

<string> : the advertising data

format : XX:XX:XX:XX:XX:XX

notice

XX : hex

It should follow adv\_data format

The first XX means the length of value,

first XX : 00-1F

ex. 07:02:01:05:03:FF:12:34

It has 7 length of value

It has two AD structures

the first AD structure has 2 length

type\_flags(01) which value is 05

the second AD structure has 3 length

manufacturer\_specific(FF) which value is 3412

```
AT+ADVDATA=02:01:05:03:ff:12:34
at_queue->Push(+ADVDATA=02:01:05:03:ff:12:34)
1 at_queue->pop(+ADVDATA=02:01:05:03:ff:12:34)
do cmd : +ADVDATA
OK
```

(intentionally blank)

● +SCANRSP

■ +SCANRSP?



Get the scan response data.

```
AT+SCANRSP?
at_queue->Push(+SCANRSP?)
1 at_queue->pop(+SCANRSP?)
do cmd : +SCANRSP
0b:09:41:54:5f:43:6f:6d:6d:61:6e:64
OK
```

■ +SCANRSP=<string>

set the scan response data

<string> : the scan response data

format : XX:XX:XX:XX:XX:XX

notice

XX : hex

It should follow adv\_data format

The first XX means the length of value, first XX : 00-1F

ex. 09:08:09:61:62:63:64:65:66:67

It has 9 length of value

It has one AD structures

the first AD structure has 8 length

local\_name\_complete(09) which value is 61:62:63:64:65:66:67 = "abcdefg"

```
AT+SCANRSP=0b:09:41:54:5f:43:6f:6d:6d:61:6e:64
at_queue->Push(+SCANRSP=0b:09:41:54:5f:43:6f:6d:6d:61:6e:64)
1 at_queue->pop(+SCANRSP=0b:09:41:54:5f:43:6f:6d:6d:61:6e:64)
do cmd : +SCANRSP
OK
```

(intentionally blank)

● +ENADV

■ +ENADV

Enable advertising of host id = 1 (Peripheral Role)

```
AT+ENADV
at_queue->Push(+ENADV)
1 at_queue->pop(+ENADV)
do cmd : +ENADV
OK

Advertising...
```

- +DISADV

- +DISADV

Disable advertising.

```
AT+DISADV
at_queue->Push(+DISADV)
1 at_queue->pop(+DISADV)
do cmd : +DISADV
OK

End Advertising...
```

(intentionally blank)

### 6.1.3 Scan Command

- +SCANTYPE

- +SCANTYPE?

Get scan type.

```
AT+SCANTYPE?  
at_queue->Push(+SCANTYPE?)  
1 at_queue->pop(+SCANTYPE?)  
do cmd : +SCANTYPE  
scan type = 1  
OK
```

- +SCANTYPE=<num>

set scan type

<num> : the scan type

0: passive

1: active

```
AT+SCANTYPE=0  
at_queue->Push(+SCANTYPE=0)  
1 at_queue->pop(+SCANTYPE=0)  
do cmd : +SCANTYPE  
OK
```

(intentionally blank)

- +SCANWIN

- +SCANWIN?

Get the scan window.

```
AT+SCANWIN?  
at_queue->Push(+SCANWIN?)  
1 at_queue->pop(+SCANWIN?)  
do cmd : +SCANWIN  
scan window = 160  
OK
```

- +SCANWIN=<num>  
set scan window  
<num> : the scan window  
range : 4-16384  
window = <num> \* 0.625ms

```
AT+SCANWIN=100  
at_queue->Push(+SCANWIN=100)  
1 at_queue->pop(+SCANWIN=100)  
do cmd : +SCANWIN  
OK
```

(intentionally blank)

- +SCANINT
- +SCANINT?

Get the scan interval

```
AT+SCANINT?  
at_queue->Push(+SCANINT?)  
1 at_queue->pop(+SCANINT?)  
do cmd : +SCANINT  
scan interval = 160  
OK
```

- +SCANINT=<num>  
set the scan interval  
<num> : the scan interval  
range : 4-16384  
interval = <num> \* 0.625ms

```
AT+SCANINT=100  
at_queue->Push(+SCANINT=100)  
1 at_queue->pop(+SCANINT=100)  
do cmd : +SCANINT  
OK
```

- +ENSCAN
  - +ENSCAN  
Enable Scan.

```
AT+ENSCAN  
at_queue->Push(+ENSCAN)  
1 at_queue->pop(+ENSCAN)  
do cmd : +ENSCAN  
OK
```

(intentionally blank)

- +DISSCAN
  - +DISSCAN

Disable scan.

```
at_queue->Push(+DISSCAN)
1 at_queue->pop(+DISSCAN)
do cmd : +DISSCAN
OK

End Scanning...
```

#### 6.1.4 Create Connection Command

- +CRCON

- +CRCON=<num>,<addr>

create connection of specific host ID = 0

<num> : address type

<addr> : the address of target device

format : XX:XX:XX:XX:XX:XX

ex. AT+CRCON=1,11:12:13:BB:BB:C4, random address 11:12:13:BB:BB:C4

```
AT+CRCON=1,11:22:33:44:55:C0
at_queue->Push(+CRCON=1,11:22:33:44:55:C0)
1 at_queue->pop(+CRCON=1,11:22:33:44:55:C0)
do cmd : +CRCON
OK
```

- +CCCON

- +CCCON

Cancel create connection.

```
AT+CCCON
at_queue->Push(+CCCON)
1 at_queue->pop(+CCCON)
do cmd : +CCCON
OK

Creating a connection is cancelled.
```

- +CCONINT

- +CCONINT?

Get the create connection interval

- +CCONINT=<num1>,<num2>

set the create connection interval

<num1> : the minimum create connection interval

range : 6-3200

interval = <num> \* 1.25ms

<num2> : the maximum create connection interval

range : 6-3200

interval = <num> \* 1.25ms

notice

<num1> must smaller <num2>

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CCONINT=6,10
at_queue->Push(+CCONINT=6,10)
1 at_queue->pop(+CCONINT=6,10)
do cmd : +CCONINT
OK
```

(intentionally blank)

- +CCONLAT

- +CCONLAT?

Get the create connection latency

```
AT+CCONLAT?  
at_queue->Push(+CCONLAT?)  
1 at_queue->pop(+CCONLAT?)  
do cmd : +CCONLAT  
0  
OK
```

■ +CCONLAT=<num>

set the create connection latency

<num> : the create connection latency

range : 0-499

interval = <num> \* 1.25ms

notice

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CCONLAT=0  
at_queue->Push(+CCONLAT=0)  
1 at_queue->pop(+CCONLAT=0)  
do cmd : +CCONLAT  
OK
```

(intentionally blank)

● +CCONSUPTMO

■ +CCONSUPTMO?



Get the create connection supervision timeout.

```
AT+CCONSUPTMO?  
at_queue->Push(+CCONSUPTMO?)  
1 at_queue->pop(+CCONSUPTMO?)  
do cmd : +CCONSUPTMO  
1000  
OK
```

■ +CCONSUPTMO=<num>

set the create connection supervision timeout

<num> : the create connection supervision timeout

range : 10-3200

supervision timeout = <num> \* 10ms

notice

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CCONSUPTMO?  
at_queue->Push(+CCONSUPTMO?)  
1 at_queue->pop(+CCONSUPTMO?)  
do cmd : +CCONSUPTMO  
1000  
OK
```

(intentionally blank)

## 6.1.5 Connection Command

- +CONINT

- +CONINT?

Get the connection interval

```
AT+CONINT?
at_queue->Push(+CONINT?)
1 at_queue->pop(+CONINT?)
do cmd : +CONINT
id=0 min:60 max:60
id=1 min:60 max:60
OK
```

- +CONINT=<num1>,<num2>,<num3>

set the connection interval of specific host ID

<num1> : host ID

range : 0-1

<num2> : the minimum connection interval

range : 6-3200

interval = <num2> \* 1.25ms

<num3> : the maximum connection interval

range : 6-3200

interval = <num3> \* 1.25ms

notice

<num2> must smaller <num3>

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CONINT=1,6,15
at_queue->Push(+CONINT=1,6,15)
1 at_queue->pop(+CONINT=1,6,15)
do cmd : +CONINT
OK

Connection updated
ID: 1, Interval: 10, Latency: 0, Supervision Timeout: 1000
```

(intentionally blank)

- +CONLAT

- +CONLAT?

Get the connection latency.

```
AT+CONLAT?
at_queue->Push(+CONLAT?)
1 at_queue->pop(+CONLAT?)
do cmd : +CONLAT
id=0 Latency=0
id=1 Latency=0
OK
```

- +CONLAT=<num1>,<num2>  
set the connection latency  
<num1> : host ID  
range : 0-1  
<num2> : the connection latency  
range : 0-499  
interval = <num2> \* 1.25ms

notice

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CONLAT=1,1
at_queue->Push(+CONLAT=1,1)
1 at_queue->pop(+CONLAT=1,1)
do cmd : +CONLAT
OK

Connection updated
ID: 1, Interval: 10, Latency: 1, Supervision Timeout: 1000
```

(intentionally blank)

- +CONSUARTMO
- +CONSUARTMO?

Get the connection supervision timeout.

```
AT+CONSPTMO?
at_queue->Push(+CONSPTMO?)
1 at_queue->pop(+CONSPTMO?)
do cmd : +CONSPTMO
1000
OK
```

■ +CONSPTMO=<num1>,<num2>

set the connection supervision timeout of specific host ID

<num1> : host ID

range : 0-1

<num2> : the connection supervision timeout

range : 10-3200

supervision timeout = <num2> \* 10ms

notice

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CONSPTMO=1,1200
at_queue->Push(+CONSPTMO=1,1200)
1 at_queue->pop(+CONSPTMO=1,1200)
do cmd : +CONSPTMO
OK

Connection updated
ID: 1, Interval: 10, Latency: 1, Supervision Timeout: 1200
```

(intentionally blank)

● +CONPARAM

■ +CONPARAM?

Get the connection parameters.

```
AT+CONPARAM?
at_queue->Push(+CONPARAM?)
1 at_queue->pop(+CONPARAM?)
do cmd : +CONPARAM
id=0 min=60 max=60 latency=0 timeout=1000
id=1 min=6 max=15 latency=1 timeout=1200
OK
```

- +CONPARAM=<num1>,<num2>,<num3>,<num4>,<num5>

set the connection interval of specific host ID

<num1> : host ID

range : 0-1

<num2> : the minimum connection interval

range : 6-3200

interval = <num2> \* 1.25ms

<num3> : the maximum connection interval

range : 6-3200

interval = <num3> \* 1.25ms

<num4> : the connection latency

range : 0-499

interval = <num4> \* 1.25ms

<num5> : the connection supervision timeout

range : 10-3200

supervision timeout = <num5> \* 10ms

notice

<num2> must smaller <num3>

must match this formula :  $\text{timeout} * 4 > \text{interval\_max} * (1 + \text{latency})$

```
AT+CONPARAM=1,10,30,0,1300
at_queue->Push(+CONPARAM=1,10,30,0,1300)
1 at_queue->pop(+CONPARAM=1,10,30,0,1300)
do cmd : +CONPARAM
OK

Connection updated
ID: 1, Interval: 20, Latency: 0, Supervision Timeout: 1300
```

- +DISCON

- +DISCON=<num1>

disable connection of specific host ID

<num> : host ID

range : 0-1

```
AT+DISCON=1
at_queue->Push(+DISCON=1)
1 at_queue->pop(+DISCON=1)
do cmd : +DISCON
OK

Disconnect, ID:1, Reason:0x16
```

- +PLDATALEN

- +PLDATALEN=<num1>,<num2>

Set the tx max payload octets of specific host ID

<num1> : host ID

range : 0-1

<num2> : the tx max payload octets

range : 27-251

```
AT+PLDATALEN=1,251
at_queue->Push(+PLDATALEN=1,251)
1 at_queue->pop(+PLDATALEN=1,251)
do cmd : +PLDATALEN
OK

Data length changed, ID: 1
MaxTxOctets: 251 MaxTxTime:2120
MaxRxOctets: 27 MaxRxTime:328
```

(intentionally blank)

- +PREPLDATALEN

- +PREPLDATALEN?

Get the preferred tx payload octets

```
AT+PREPLDATALEN?
at_queue->Push(+PREPLDATALEN?)
1 at_queue->pop(+PREPLDATALEN?)
do cmd : +PREPLDATALEN
27
OK
```

- +PREPLDATALEN=<num1>

set the preferred tx max payload octets

<num1> : the tx max payload octets

range : 27-251

```
AT+PREPLDATALEN=200
at_queue->Push(+PREPLDATALEN=200)
1 at_queue->pop(+PREPLDATALEN=200)
do cmd : +PREPLDATALEN
OK
```

- +RMTUS

- +RMTUS=<num>

read MTU size of specific host ID

<num> : host ID

range : 0-1

```
AT+RMTUS=1
at_queue->Push(+RMTUS=1)
1 at_queue->pop(+RMTUS=1)
do cmd : +RMTUS
mtu size = 23
OK
```

(intentionally blank)

- +PHY

- +PHY?

## Read PHY rate

- +PHY=<num1>,<num2>,<num3>,<num4>

set PHY rate of specific host ID

<num1> : host ID

range : 0-1

<num2> : TX PHY

1:BLE\_PHY\_1M

2:BLE\_PHY\_2M

4:BLE\_PHY\_CODED

<num3> : RX PHY

1:BLE\_PHY\_1M

2:BLE\_PHY\_2M

4:BLE\_PHY\_CODED

<num4> : phy option when TX/RX choose BLE\_PHY\_CODED(4)

notice

TX PHY is recommended to be equivalent to RX PHY

```
AT+PHY=1,2,2,0
at_queue->Push(+PHY=1,2,2,0)
1 at_queue->pop(+PHY=1,2,2,0)
do cmd : +PHY
OK
PHY updated/read, ID: 1, TX PHY: 2, RX PHY: 2
```

- +READPHY

- +READPHY=<num>

read PHY rate of specific host ID

<num> : host ID

range : 0-1

```
AT+READPHY=1
at_queue->Push(+READPHY=1)
1 at_queue->pop(+READPHY=1)
do cmd : +READPHY
OK
PHY updated/read, ID: 1, TX PHY: 2, RX PHY: 2
```

- +READRSSI

- +READRSSI=<num>



read RSSI value of specific host ID

<num> : host ID

range : 0-1

```
AT+READRSSI=1
at_queue->Push(+READRSSI=1)
1 at_queue->pop(+READRSSI=1)
do cmd : +READRSSI
OK

ID=1, RSSI=-20
```

- +SEC

- +SEC=<num>

request security to specific host ID

<num>: host ID

range : 0-1

```
AT+SEC=1
at_queue->Push(+SEC=1)
1 at_queue->pop(+SEC=1)
do cmd : +SEC
OK

AUTH Report, ID:1 , STATUS:0
```

- +EXMTUS

- +EXMTUS=<num1>,<num2>

exchange MTU size of specific host ID with server

<num1> : host ID

range : 0-1

<num2> : Client rx mtu size

range : 23-247

```
AT+EXMTUS=0,247
at_queue->Push(+EXMTUS=0,247)
1 at_queue->pop(+EXMTUS=0,247)
do cmd : +EXMTUS
OK

MTU Exchanged, ID:0, size: 247
```

- +RSCCCD

- +RSCCCD=<num>

restore last bond CCCD

<num> : host ID

range : 0-1

notice: this cmd is only valid in following situation

A connection bond with bond\_flag=1(bonding)

Doing some CCCD operate

Discon and reconnect the connection

Do the +RSCCCD command which can retrieve CCCD state before disconnection

```
AT+RSCCCD=1
at_queue->Push(+RSCCCD=1)
1 at_queue->pop(+RSCCCD=1)
do cmd : +RSCCCD
OK
```

- +READ

- +READ=<num1>,<num2>

read the specified characteristic value for specific host ID

<num1>: host ID

range : 0-1

<num2>: service handle value

Range:0-65535

```
AT+READ=0,17
at_queue->Push(+READ=0,17)
1 at_queue->pop(+READ=0,17)
do cmd : +READ
OK

Manufacturer name: BLE
```

(intentionally blank)

- +WRITE

- +WRITE=<num1>,<num2>,<num3>,<raw>

send write value for specific host ID

<num1> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

<num3> : the num of write length

range : 0-244

<raw>: Raw data

notice: the num of write value length should lower than (mtu size - 3)

```
AT+WRITE=0,42,5,abcde
at_queue->Push(+WRITE=0,42,5,abcde)
1 at_queue->pop(+WRITE=0,42,5,abcde)
do cmd : +WRITE
OK
```

- +WRITENRSP

- +WRITENRSP=<num1>,<num2>,<num3>,<raw>

send write value for specific host ID, which will not get response

<num1> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

<num3> : the num of write length

range : 0-244

<raw>: Raw data

notice: the num3 of write value length should lower than (mtu size – 3)

```
AT+WRITENRSP=0,42,10,abcdefghij
at_queue->Push(+WRITENRSP=0,42,10,abcdefghij)
1 at_queue->pop(+WRITENRSP=0,42,10,abcdefghij)
do cmd : +WRITENRSP
OK
```

- +WRITECCCD

- +WRITECCCD=<num1>,<num2>,<num3>

Write CCCD value for specific host id

<num1> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

<num3> : the num of write length

range : 0-244

notice :

0:disable notify & disable indicate

1:enable notify & disable indicate

2:disable notify & enable indicate

3:enable notify & enable indicate

```
AT+WRITECCCD=0,40,3
at_queue->Push(+WRITECCCD=0,40,3)
1 at_queue->pop(+WRITECCCD=0,40,3)
do cmd : +WRITECCCD
OK
```

- +READCCCD

- +READCCCD=<num1>,<num2>

read CCCD value for specific host ID

<num> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

notice :

0:disable notify & disable indicate

1:enable notify & disable indicate

2:disable notify & enable indicate

3:enable notify & enable indicate

```
AT+READCCCD=0,40
at_queue->Push(+READCCCD=0,40)
1 at_queue->pop(+READCCCD=0,40)
do cmd : +READCCCD
OK
```

- +NFY

- +NFY=<num1>,<num2>,<num3>,<raw>

send notification for specific host ID

<num1> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

<num3> : the num of write length

range : 0-244

<raw>: Raw data

notice: the num3 of write value length should lower than (mtu size - 3)

```
AT+NFY=1,39,5,abcde
at_queue->Push(+NFY=1,39,5,abcde)
1 at_queue->pop(+NFY=1,39,5,abcde)
do cmd : +NFY
OK
```

- +IND

- +IND=<num1>,<num2>,<num3>,<raw>

send indication for specific host ID

<num1> : host ID

range : 0-1

<num2>: service handle value

Range:0-65535

<num3> : the num of write length

range : 0-244

<raw>: Raw data

notice: the num3 of write value length should lower than (mtu size - 3)

```
AT+IND=1,39,10,ABCDEFGHIIJ
at_queue->Push(+IND=1,39,10,ABCDEFGHIIJ)
1 at_queue->pop(+IND=1,39,10,ABCDEFGHIIJ)
do cmd : +IND
OK
```

(intentionally blank)

## Revision History

Revision	Description	Owner	Date
1.0	Initial version.	Yuwei	2022/02/15
1.1	Remove the repeat introduction which already addressed in BLE SDK user guide.	Yuwei	2022/03/28
1.2	Add Beacon & AT command Demo	Yuwei	2022/06/06
1.3	Add HOGP_Peripheral_Privacy Demo	Nat	2023/11/28

© 2021 by Rafael Microelectronics, Inc.

All Rights Reserved.

Information in this document is provided in connection with **Rafael Microelectronics, Inc.** ("**Rafael Micro**") products. These materials are provided by **Rafael Micro** as a service to its customers and may be used for informational purposes only. **Rafael Micro** assumes no responsibility for errors or omissions in these materials. **Rafael Micro** may make changes to this document at any time, without notice. **Rafael Micro** advises all customers to ensure that they have the latest version of this document and to verify, before placing orders, that information being relied on is current and complete. **Rafael Micro** makes no commitment to update the information and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to its specifications and product descriptions.

THESE MATERIALS ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, RELATING TO SALE AND/OR USE OF **RAFAEL MICRO** PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, CONSEQUENTIAL OR INCIDENTAL DAMAGES, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. **RAFAEL MICRO** FURTHER DOES NOT WARRANT THE ACCURACY OR COMPLETENESS OF THE INFORMATION, TEXT, GRAPHICS OR OTHER ITEMS CONTAINED WITHIN THESE MATERIALS. **RAFAEL MICRO** SHALL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES, INCLUDING WITHOUT LIMITATION, LOST REVENUES OR LOST PROFITS, WHICH MAY RESULT FROM THE USE OF THESE MATERIALS.

**Rafael Micro** products are not intended for use in medical, lifesaving or life sustaining applications. **Rafael Micro** customers using or selling **Rafael Micro** products for use in such applications do so at their own risk and agree to fully indemnify **Rafael Micro** for any damages resulting from such improper use or sale. **Rafael Micro**, logos and **RT58x** are **Trademarks** of **Rafael Microelectronics, Inc.** Product names or services listed in this publication are for identification purposes only, and may be trademarks of third parties. Third-party brands and names are the property of their respective owners.