GR551x Power Consumption Profile Example Application

Version: 1.9

Release Date: 2020-12-15

Shenzhen Goodix Technology Co., Ltd.

Trademarks and Permissions

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

Disclaimer

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

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

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

Shenzhen Goodix Technology Co., Ltd.

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

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

Website: www.goodix.com

Preface

Purpose

This document introduces how to use and verify a power consumption measurement example in a GR551x SDK, to help users quickly get started with secondary development.

Audience

This document is intended for:

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

Release Notes

This document is the seventh release of *GR551x Power Consumption Profile Example Application*, corresponding to GR551x SoC series.

Revision History

Version	Date	Description
1.0	2019-12-08	Initial release
1.3	2020-03-16	Updated the release time in the footers.
1.5	2020-05-30	Updated the logos in the headers.
1.6	2020-06-30	Updated the document version based on SDK changes.
1.7	2020-09-25	Added a note for removing the jumper caps on J5 before power consumption measurement and added the hardware layout of GR5515 SK Board in “Section 3.2 Hardware Connection”.
1.8	2020-11-09	Updated Figure 3-3 in “Section 3.4.1 Setting of Measurement Scenarios”.
1.9	2020-12-15	Updated GRToolbox UI figure based on software update.

Contents

1 Introduction

GR551x Power Consumption Profile (PCP) example is based on the GR551x SDK and the GR5515 Starter Kit Board (GR5515 SK Board). Users can set parameters in real time through mobile phones, to configure GR551x power consumption measurement scenarios.

This document introduces how to use and verify a custom Goodix PCP example in a GR551x SDK. Before you get started, it is recommended to refer to the following documents.

Table 1-1 Reference documents

Name	Description
GR551x Sample Service Application and Customization	Introduces how to apply and customize Goodix Sample Service in developing Bluetooth LE applications based on GR551x SDK.
GR551x Developer Guide	Introduces the software/hardware and quick start guide of GR551x SoCs.
Bluetooth Core Spec v5.1	Offers official Bluetooth standards and core specification (v5.1) from Bluetooth SIG. Available at https://www.bluetooth.com/specifications/bluetooth-core-specification/.
Bluetooth GATT Spec	Provides details about Bluetooth profiles and services. Available at www.bluetooth.com/specifications/gatt.
J-Link/J-Trace User Guide	Provides J-Link operational instructions. Available at www.segger.com/downloads/jlink/UM08001_JLink.pdf.
Keil User Guide	Offers detailed Keil operational instructions. Available at www.keil.com/support/man/docs/uv4/.

2 Profile Overview

The Power Consumption Service (PCS) is defined in PCP. It is customized by Goodix, the 128-bit vendor-specific UUID of which is A6ED0501-D344-460A-8075-B9E8EC90D71B, to transmit data and commands and receive responses.

PCS includes two characteristics:

- TX: Transmit data.
- Setting: Send commands to customize power consumption measurement scenarios and receive responses of command execution.

The characteristics are described in detail as follows.

Table 2-1 PCS characteristics

Characteristic	UUID	Type	Support	Security	Property
TX	A6ED0202-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Notify
Setting	A6ED0203-D344-460A-8075-B9E8EC90D71B	128 bits	Mandatory	None	Write, Indicate

3 Initial Operation

This chapter introduces how to rapidly verify a PCP example in a GR551x SDK.

Note:

SDK_Folder is the root directory of GR551x SDK.

3.1 Preparation

Perform the following tasks before verifying and testing a PCP example.

- **Hardware preparation**

Table 3-1 Hardware preparation

Name	Description
J-Link debug probe	JTAG emulator launched by SEGGER. For more information, visit www.segger.com/products/debug-probes/j-link/.
Development board	GR5515 Starter Kit Board
Cable	Micro USB 2.0 cable
Keysight N6705C	DC power analyzer launched by Keysight

- **Software preparation**

Table 3-2 Software preparation

Name	Description
Windows	Windows 7/Windows 10
J-Link driver	A J-Link driver. Available at www.segger.com/downloads/jlink/.
Keil MDK5	An integrated development environment (IDE). Available at www.keil.com/download/product/.
GRToolbox (Android)	A Bluetooth LE debugging tool for GR551x. Available in `SDK_Folder\tools\GRToolbox`.
GProgrammer (Windows)	A GR551x programming tool. Available in `SDK_Folder\tools\GProgrammer`.
Keysight 14585A	Power control and analysis software launched by Keysight

3.2 Hardware Connection

1. Power on Keysight N6705C, and turn the knob as shown at Location B in Figure 3-1, to set the output channel voltage until the display at Location C in Figure 3-1 shows 3.3 V.
2. Connect the positive pole (“+”) at Location A in Figure 3-1 to VBAT pin (J10 Pin 2, as shown in Figure 3-2) on the GR5515 SK Board; connect the negative pole (“-”) to GND pin, to power the GR551x SoC on the board.

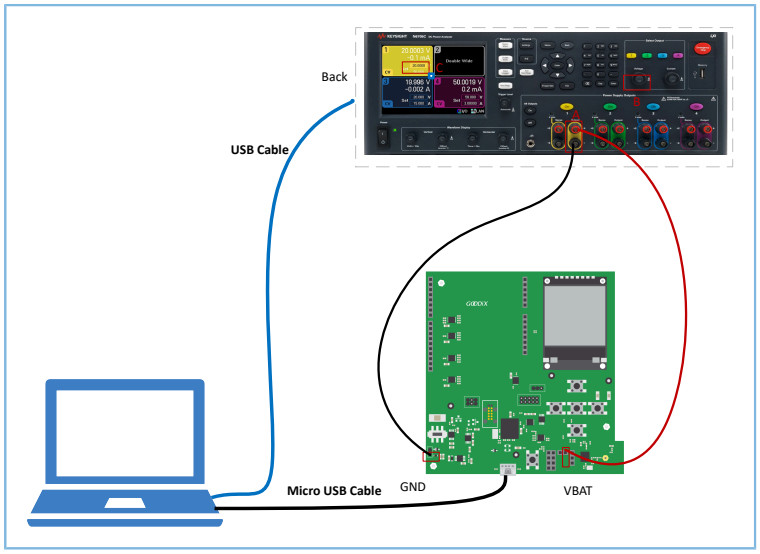

Figure 3-1 Hardware connection

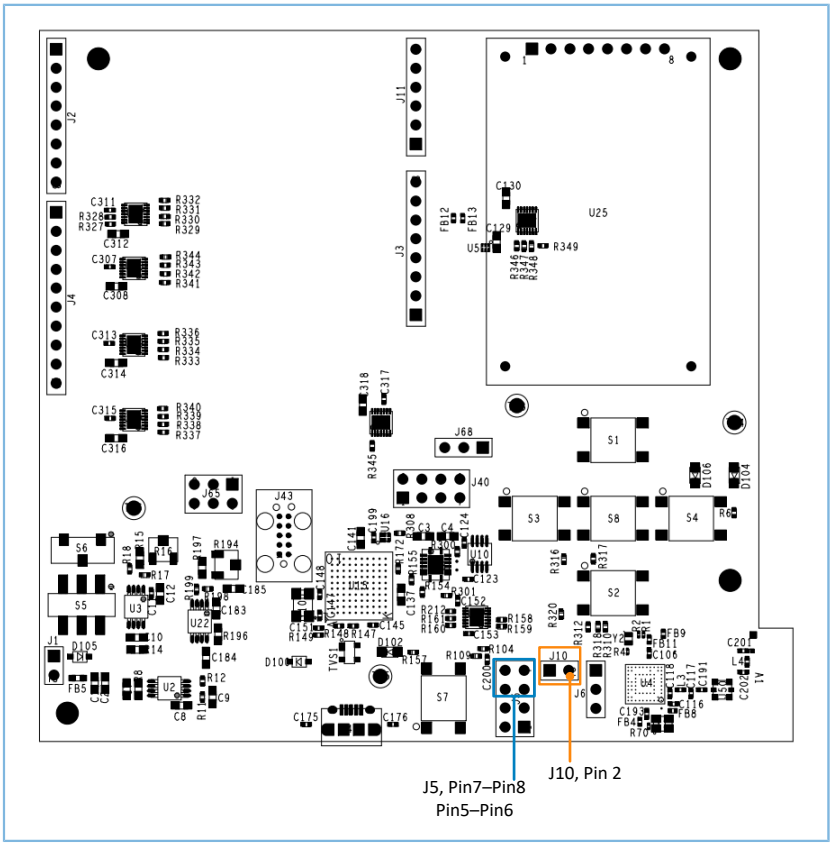

Figure 3-2 Hardware layout of GR5515 SK Board (top view)

3. Connect the GR5515 SK Board to any USB port of the PC with a Micro USB cable, to power the chip peripheral circuit on the board.

4. Connect Keysight N6705C to the PC with a USB cable.

Note:

Before power consumption measurement, remove the jumper caps on Pin 5–Pin 6 and Pin 7–Pin 8 of J5 (as shown in Figure 3-2), and remove the LCD in the upper-right corner on the board, to prevent abnormal measurements due to current leakage from VDDIO.

3.3 Firmware Download

Download *ble_app_pcs_fw.bin* to the GR5515 SK Board. For details, refer to *GProgrammer User Manual*.

Note:

The *ble_app_pcs_fw.bin* is in SDK_`Folder\projects\ble\ble_peripheral\ble_app_pcs\build`. SDK_Folder is the root directory of GR551x SDK.

3.4 Test and Verification

When the GR5515 SK Board downloaded with *ble_app_pcs_fw.bin* is powered on, the system enters Ultra Deep Sleep Mode. After reset, press **OK** on the board for more than 3 seconds to allow the system to initiate advertising, which lasts for 30 seconds. If the board is not connected to other devices, the system then enters Sleep Mode again due to advertising timeout; if the board is connected to other devices, the system also enters Sleep Mode when the board is in disconnected state, until the system wakes up. Press **OK** to wake the system up from Sleep Mode.

Note:

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

3.4.1 Setting of Measurement Scenarios

Set measurement scenarios by using the mobile tool GRToolbox. Detailed steps are shown as follows:

1. Run GRToolbox, and select **Application** > **PCS**.

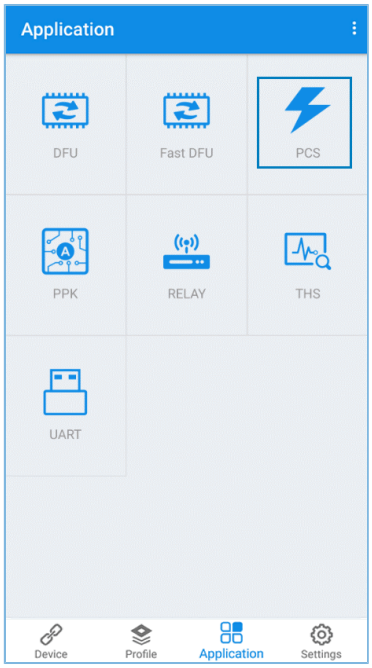

Figure 3-3 Choosing **PCS**

2. Tap **CONNECT** and then start scanning target devices. Discover a device with the advertising name **Goodix_Power** (the advertising name can be modified in the *user_app.c* file).

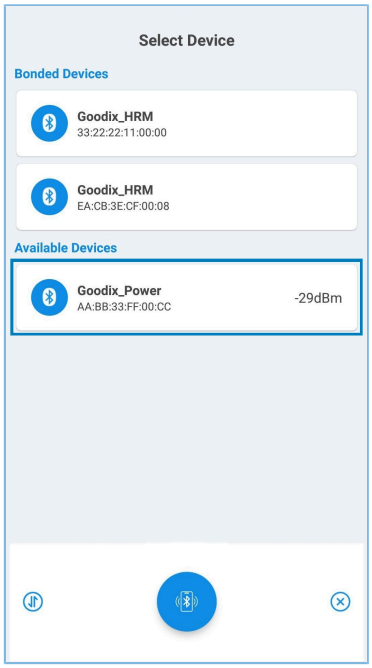

Figure 3-4 Discovering **Goodix_Power**

3. Tap **Goodix_Power** to establish connection, and then enter the setting page to set related power consumption measurement scenarios, including **Adv Interval**, **Adv Data**, **Connection Param**, **PHY Mode**, **Tx Power**, and **Enable Notify**, as shown in the figure below. Settings of **Adv Interval**, **Adv Data**, and **Tx Power** are valid only if advertising is restarted after the current connection is broken; settings of **Connection Param** and **PHY Mode** are

valid only for the current connection. If no advertising name or service UUID is discovered, tap **Last Connected Device** to search devices based on the previous device MAC address.

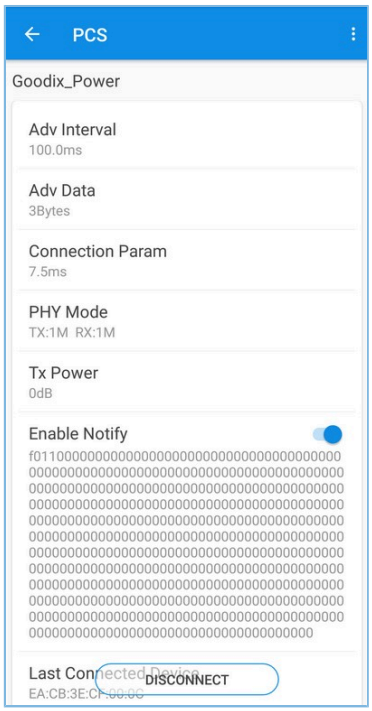

Figure 3-5 Setting page of power consumption measurement scenarios

4. If the connection is broken, press **OK** on the GR5515 SK Board, to allow the device to re-initiate advertising with the configured data length and connection interval.

3.4.2 GR551x Power Consumption Measurement

After measurement scenarios are set, measure GR551x power consumption in different scenarios with Keysight installed on the PC.

Scenario 1: advertising state at an interval of 1s

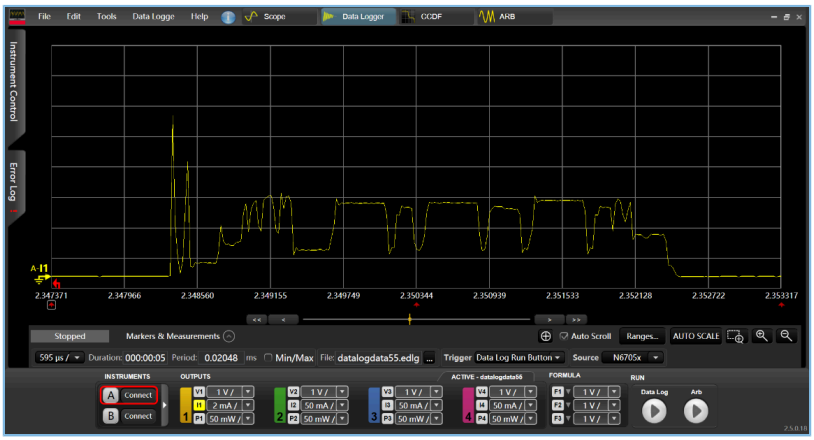

Figure 3-6 Power consumption measurement scenario 1

Scenario 2: connecting state at an interval of 200 ms

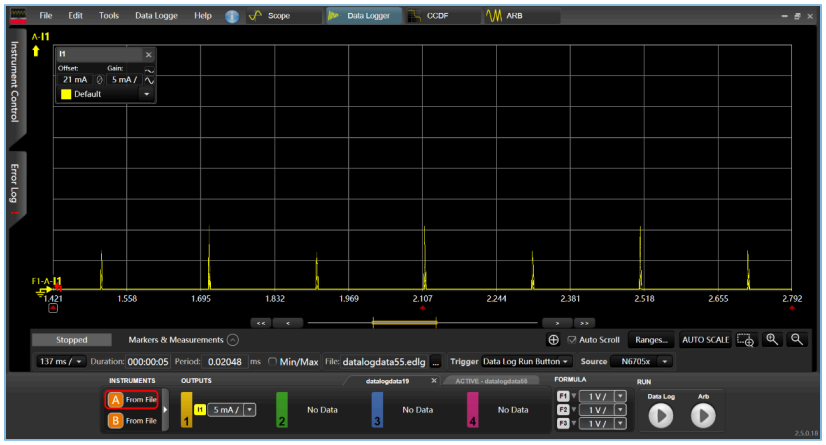

Figure 3-7 Power consumption measurement scenario 2

Users can set other power consumption measurement scenarios on demand.

4 Application Details

This chapter introduces the project directory and interactions of the PCP example, as well as part of the core code in the project.

4.1 Project Directory

The source code and project file of PCP applications are in SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs, and project file is in the Keil_5 folder.

Double-click the project file *ble_app_pcs.uvprojx*, to view the project directory structure of ble_app_pcs example in Keil. For related files, see the table below.

Table 4-1 File description of ble_app_pcs

Group	File	Description
gr_profiles	ble_prf_utils.c	This file contains profile-related operational tools.
	pcs.c	This is the source file of power consumption service.
user_callback	user_gap_callback.c	This file obtains connection, connection parameter update, and PHY update events.
user_platform	user_periph_setup.c	This file sets parameters, including device address and Sleep Mode.
user_app	main.c	This file contains the main() function.
	user_app.c	This file handles PCS registration and related events.

4.2 Implementation Procedures and Code

When the board downloaded with PCP example is powered on, the system initializes peripherals and power management, BLE Protocol Stack, and PCS successively. The main procedures are shown in the figure below:

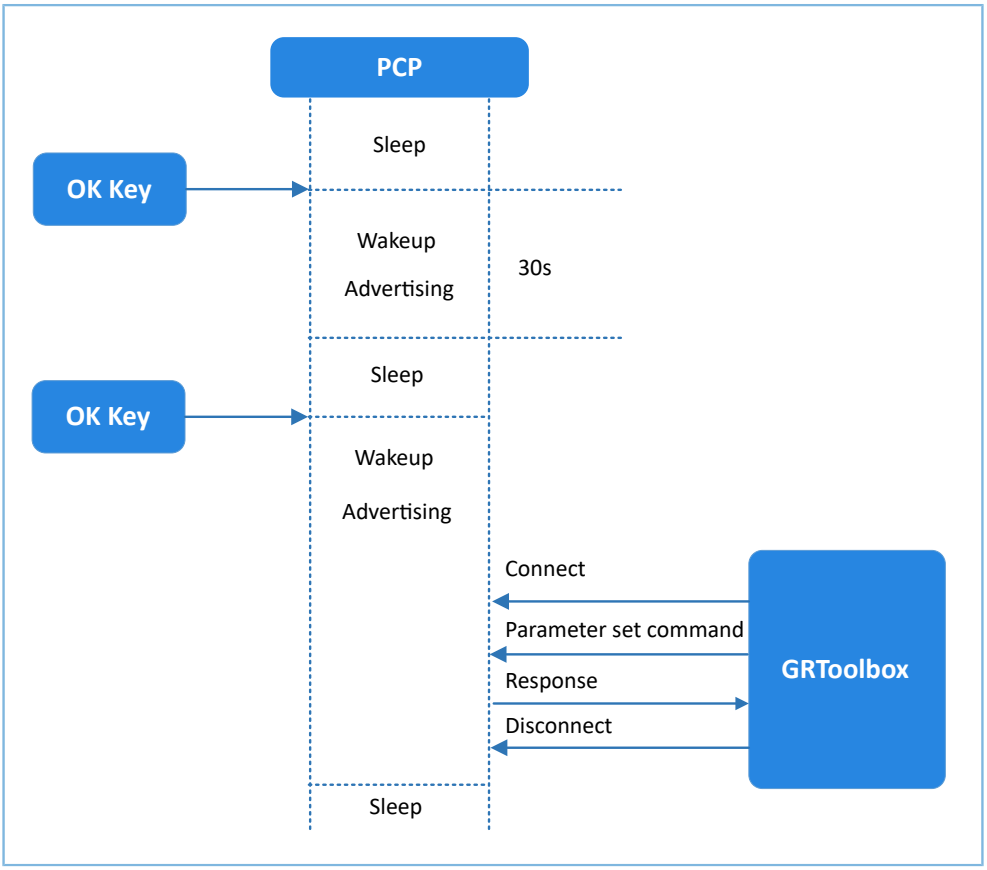

Figure 4-1 Implementation procedures

Note:

The main logical code of the PCP example is in:

user_periph_setup.c under user_platform in the Keil project directory tree.

main.c and *user_app.c* under user_app in the Keil project directory tree.

The following parts elaborate on the interactions between the GR5515 SK Board and GRToolbox.

1. Power management configuration

   (1). Configure the power management mode of the GR551x SoC to Sleep Mode (PMR_MGMT_SLEEP_MODE), and configure an external wakeup source (on-board **OK** button) to wake up the GR551x SoC and enable advertising. The code snippet is as follows.

```
static void wkup_key_init(void)
{
    s_gpiote_param.type          = KEY_OK_IO_TYPE;
    s_gpiote_param.mode          = KEY_TRIGGER_MODE;
    s_gpiote_param.pin           = KEY_OK_PIN;
    s_gpiote_param.pull          = APP_IO_PULLUP;
    s_gpiote_param.handle_mode   = APP_IO_ENABLE_WAKEUP;
    s_gpiote_param.io_evt_cb     = app_gpiote_event_handler;

    app_gpiote_init(&s_gpiote_param, 1);
}
```

```
void app_periph_init(void)
{
    SYS_SET_BD_ADDR(s_bd_addr);
    ble_rf_tx_mode_set(BLE_RF_TX_MODE_ULP_MODE);
    ble_rf_match_circuit_set(BLE_RF_MATCH_CIRCUIT_25OHM);
    wkup_key_init();
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

(2). The following function determines whether the system enters Ultra Deep Sleep Mode or implements normal task logics after booting. It is called in the main() function.

```
bool is_enter_ultra_deep_sleep(void)
{
    if (APP_IO_PIN_RESET ! = app_io_read_pin(APP_IO_TYPE_AON, KEY_OK_PIN))
    {
        return true;
    }

    return false;
}
```

(3). The main() function determines the procedure branches and low power management of the system after booting.

```
int main(void)
{
   app_periph_init();

   ble_stack_init(&s_app_ble_callback, &heaps_table);

   if (is_enter_ultra_deep_sleep())
   {
       pwr_mgmt_ultra_sleep(0);
   }

   while (1)
   {
       pwr_mgmt_schedule();
   }
}
```

2. Command parsing, execution, and response

When Setting Characteristic Value receives a command from the peer device, it reports events and related information to the application layer. The pcs_param_parse() function then can be used to parse, execute, and respond to the command. To set the scenarios for power consumption measurement, follow the steps below.

- Set advertising interval.

  Set the advertising interval value, and respond to the peer device. This value takes effect on the next advertising. For specific code, see processing code of PCS_SETTING_TYPE_ADV_INTERVAL event in the pcs_param_parse() function.

```
void pcs_param_parse(uint8_t conn_idx, uint8_t *p_data, uint16_t length)
{
```

```
    ...
    switch (p_data[0])
    {
        case PCS_SETTING_TYPE_ADV_INTERVAL:
            s_gap_adv_param.adv_intv_max = BUILD_U16(p_data[1], p_data[2]);
            s_gap_adv_param.adv_intv_min = BUILD_U16(p_data[1], p_data[2]);
            response[0] = PCS_SETTING_TYPE_ADV_INTERVAL;
            response[1] = PCS_SET_PARAM_SUCCESS;
            pcs_setting_reply(0, response, 2);
            break;

        ...

        default:
            break;
    }
}
```

- Set advertising data.

  Set advertising data, and respond to the peer device based on execution results. The data length can be set to 3, 10, 17, 24, or 31 bytes, which takes effect on the next advertising. For specific code, see processing code of PCS_SETTING_TYPE_ADV_DATA event in the pcs_param_parse() function.

Note:

You need to subtract 3-byte effective length when setting user advertising data by using ble_gap_adv_param_set(). This is because Advertising Type Flag occupies 3 bytes of the advertising data.

```
void pcs_param_parse(uint8_t conn_idx, uint8_t *p_data, uint16_t length)
{
    ...
    switch (p_data[0])
    {
        ...
        case PCS_SETTING_TYPE_ADV_DATA:
            response[0] = PCS_SETTING_TYPE_ADV_DATA;
            response[1] = PCS_SET_PARAM_SUCCESS;

            if (PCS_SET_ADV_DATA_3B == p_data[1])
            {
                s_adv_data_set.length = 0;     // 3 byte for adv type
            }
            else if (PCS_SET_ADV_DATA_10B == p_data[1])
            {
                memcpy(s_adv_data_set.adv_data, s_adv_data_10b, 7);
                s_adv_data_set.length   = 7;     // 3 byte for adv type
            }
            else if (PCS_SET_ADV_DATA_17B == p_data[1])
            {
                memcpy(s_adv_data_set.adv_data, s_adv_data_17b, 14);
                s_adv_data_set.length   = 14;     // 3 byte for adv type
            }
            else if (PCS_SET_ADV_DATA_24B == p_data[1])
            {
                memcpy(s_adv_data_set.adv_data, s_adv_data_24b, 21);
                s_adv_data_set.length   = 21;     // 3 byte for adv type
            }
            else if (PCS_SET_ADV_DATA_31B == p_data[1])
```

```
            {
                memcpy(s_adv_data_set.adv_data, s_adv_data_31b, 28);
                s_adv_data_set.length   = 28;     // 3 byte for adv type
            }
            else
            {
                response[1] = PCS_SET_PARAM_FAIL;
            }
            pcs_setting_reply(0, response, 2);
            break;
        ...
        default:
            break;
    }
}
```

- Set connection parameters.

  Set the connection interval, the slave device delay, and the monitoring timeout values for the current link. As an asynchronous function, ble_gap_conn_param_update responds to the peer device in the app_gap_connection_update_cb() callback based on the execution results.

  If parameters are successfully set, they become valid for the current connection. For specific code, see processing code of PCS_SETTING_TYPE_CONN_PARAM event in the pcs_param_parse() function.

```
void pcs_param_parse(uint8_t conn_idx, uint8_t *p_data, uint16_t length)
{
    ...
    switch (p_data[0])
    {
        ...
        case PCS_SETTING_TYPE_CONN_PARAM:
            gap_conn_param.interval_min  = BUILD_U16(p_data[1], p_data[2]);
            gap_conn_param.interval_max  = BUILD_U16(p_data[3], p_data[4]);
            gap_conn_param.slave_latency = BUILD_U16(p_data[5], p_data[6]);
            gap_conn_param.sup_timeout   = BUILD_U16(p_data[7], p_data[8]);

            if (SDK_SUCCESS ==
                ble_gap_conn_param_update(conn_idx, &gap_conn_param))
            {
                g_is_user_set_op = true;
            }
            break;
        ...
        default:
            break;
    }
}
```

- Set PHY mode.

  The PHY mode of the current link can be updated to 1M, 2M, or Coded PHY. As an asynchronous function, ble_gap_phy_update responds to the peer device in the app_gap_phy_update_cb() callback based on execution results. For specific code, see processing code of PCS_SETTING_TYPE_PHY event in the pcs_param_parse() function.

```
void pcs_param_parse(uint8_t conn_idx, uint8_t *p_data, uint16_t length)
{
```

```
    ...
    switch (p_data[0])
    {
        ...
        case PCS_SETTING_TYPE_PHY:
            tx_phys = p_data[1];
            rx_phys = p_data[2];
            phy_opt = p_data[3];
            if (SDK_SUCCESS == ble_gap_phy_update(0, tx_phys, rx_phys, phy_opt))
            {
                g_is_user_set_op = true;
            }
            break;
        ...
        default:
            break;
    }
}
```

- Set TX power.

  Set TX power for the current connection and the next advertising of the device, return execution results synchronously, and respond to the peer device. For specific code, see processing code of PCS_SETTING_TYPE_TX_POWER event in the pcs_param_parse() function.

```
void pcs_param_parse(uint8_t conn_idx, uint8_t *p_data, uint16_t length)
{
    ...
    switch (p_data[0])
    {
        ...
        case PCS_SETTING_TYPE_TX_POWER:
            if (0x01 == p_data[1])
            {
                tx_power_set = 0 - p_data[2];
            }
            else if (0x00 == p_data[1])
            {
                tx_power_set = p_data[2];
            }
            s_gap_adv_param.max_tx_pwr = tx_power_set;
           error_code = ble_gap_tx_power_set(GAP_TX_POWER_ROLE_CON, conn_idx, tx_power_set);
           response[0] = PCS_SETTING_TYPE_TX_POWER;
           response[1] = SDK_SUCCESS == error_code ? PCS_SET_PARAM_SUCCESS :
                                                     PCS_SET_PARAM_FAIL;
            pcs_setting_reply(0, response, 2);
            break;
        default:
            break;
    }
}
```

3. Enable Notify.

If the peer device enables Notify (writing the value 0x0001 to CCCD), the example application starts notifying data after it receives PCS_EVT_TX_ENABLE event; if one data transmission is completed, the example application notifies the data again after it receives PCS_EVT_DATA_SENT, and stops notifying the data when it receives PCS_EVT_TX_DISABLE.

```
static void pcs_service_event_process(pcs_evt_t *p_evt)
{
    switch (p_evt->evt_type)
    {
        case PCS_EVT_TX_ENABLE:
            s_is_notify_enable = true;
            pcs_tx_data_notify();
            break;
        case PCS_EVT_TX_DISABLE:
            s_is_notify_enable = false;
            break;
        case PCS_EVT_TX_DATA_SENT:
            if (s_is_notify_enable)
            {
                s_notify_counter++;
                pcs_tx_data_notify();
            }
            break;
        case PCS_EVT_PARAM_SET:
            pcs_param_parse(p_evt->conn_idx, p_evt->p_data, p_evt->length);
            break;
        case PCS_EVT_DISCONNECTED:
            break;
        default:
            break;
    }
}

static void pcs_tx_data_notify(void)
{
    uint8_t notify_data[PCS_MAX_DATA_LEN] = {0};
    notify_data[0] = LO_UINT32_T(s_notify_counter);
    notify_data[1] = L2_UINT32_T(s_notify_counter);
    notify_data[2] = L3_UINT32_T(s_notify_counter);
    notify_data[3] = HI_UINT32_T(s_notify_counter);
    pcs_tx_data_send(0, notify_data, PCS_MAX_DATA_LEN);
}
```