



## GR551x Power Consumption Profile示例手册

版本： 1.9

发布日期： 2020-12-15

版权所有 © 2020 深圳市汇顶科技股份有限公司。保留一切权利。

非经本公司书面许可，任何单位和个人不得对本手册内的任何部分擅自摘抄、复制、修改、翻译、传播，或将其全部或部分用于商业用途。

## 商标声明

**GOODIX** 和其他汇顶商标均为深圳市汇顶科技股份有限公司的商标。本文档提及的其他所有商标或注册商标，由各自的所有人持有。

## 免责声明

本文档中所述的器件应用信息及其他类似内容仅为您提供便利，它们可能由更新之信息所替代。确保应用符合技术规范，是您自身应负的责任。

深圳市汇顶科技股份有限公司（以下简称“GOODIX”）对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对其使用情况、质量、性能、适销性或特定用途的适用性的声明或担保。GOODIX对因这些信息及使用这些信息而引起的后果不承担任何责任。

未经GOODIX书面批准，不得将GOODIX的产品用作生命维持系统中的关键组件。在GOODIX知识产权保护下，不得暗或以其他方式转让任何许可证。

深圳市汇顶科技股份有限公司

总部地址：深圳市福田区腾飞工业大厦B座2层、13层

电话：+86-755-33338828      传真：+86-755-33338099

网址：[www.goodix.com](http://www.goodix.com)

## 前言

### 编写目的

本文档介绍如何使用和验证GR551x SDK中的功耗测试示例，旨在帮助用户快速进行二次开发。

### 读者对象

本文适用于以下读者：

- GR551x用户
- GR551x开发人员
- GR551x测试人员
- 开发爱好者
- 文档工程师

### 版本说明

本文档为第7次发布，对应的产品系列为GR551x。

### 修订记录

版本	日期	修订内容
1.0	2019-12-08	首次发布
1.3	2020-03-16	更新了文档页脚版本时间
1.5	2020-05-30	更新了文档页眉图标
1.6	2020-06-30	基于SDK刷新版本
1.7	2020-09-25	“3.2 硬件连接”章节，补充开始功耗测量前需先移除J5上跳线帽的说明、增加开发板硬件布局图
1.8	2020-11-09	更新“3.4.1 设置测量应用场景”图3-3
1.9	2020-12-15	更新GRToolbox软件界面截图

目录

前言..... I

1 简介..... 1

2 Profile概述..... 2

3 初次运行..... 3

    3.1 准备工作..... 3

    3.2 硬件连接..... 3

    3.3 下载固件..... 4

    3.4 测试验证..... 5

        3.4.1 设置测量应用场景..... 5

        3.4.2 观测GR551x功耗..... 7

4 应用详解..... 8

    4.1 工程目录..... 8

    4.2 实现流程及代码介绍..... 8

## 1 简介

GR551x Power Consumption Profile（以下简称PCP）示例基于GR551x SDK和GR5515 Starter Kit开发板（以下简称“SK板”），通过手机端实时设置参数，实现GR551x功耗测量场景配置。

本文档将介绍如何使用及验证GR551x SDK中的Goodix自定义PCP示例。在进行操作前，建议参考如表 1-1 所示文档。

表 1-1 文档参考

名称	描述
应用及自定义GR551x Sample Service	介绍实现自定义Service的相关知识
GR551x开发者指南	GR551x软硬件介绍、快速使用及资源总览
Bluetooth Core Spec v5.1	Bluetooth官方标准核心规范5.1： <a href="https://www.bluetooth.com/specifications/bluetooth-core-specification/">https://www.bluetooth.com/specifications/bluetooth-core-specification/</a>
Bluetooth GATT Spec	Bluetooth Profile和服务的详细信息： <a href="http://www.bluetooth.com/specifications/gatt">www.bluetooth.com/specifications/gatt</a>
J-Link用户指南	J-Link使用说明： <a href="http://www.segger.com/downloads/jlink/UM08001_JLink.pdf">www.segger.com/downloads/jlink/UM08001_JLink.pdf</a>
Keil用户指南	Keil详细操作说明： <a href="http://www.keil.com/support/man/docs/uv4/">www.keil.com/support/man/docs/uv4/</a>

## 2 Profile概述

PCP中定义了功耗服务（Power Consumption Service，PCS）。该服务由Goodix自定义，专属128位UUID为A6ED0501-D344-460A-8075-B9E8EC90D71B，用于数据发送、指令发送以及接收回应。

PCS包含两个特征：

- TX Characteristic：发送数据。
- Setting Characteristic：发送指令定制的功耗测试场景、接收指令的执行回应。

Characteristic的具体描述如表 2-1 所示。

表 2-1 PCS Characteristic

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

### 3 初次运行

本章介绍如何快速验证GR551x SDK中的PCP示例。

 说明:

SDK\_Folder为GR551x SDK的根目录。

#### 3.1 准备工作

验证并测试PCP示例之前，需要完成以下准备工作。

- 硬件准备

表 3-1 硬件准备

名称	描述
J-Link工具	SEGGER公司推出的JTAG仿真器，如需更多了解，请访问 <a href="http://www.segger.com/products/debug-probes/j-link/">www.segger.com/products/debug-probes/j-link/</a>
开发板	GR5515 Starter Kit开发板
数据线	Micro USB 2.0数据线
Keysight N6705C	Keysight公司推出的直流功率分析仪

- 软件准备

表 3-2 软件准备

名称	描述
Windows	Windows 7/Windows 10操作系统
J-Link Driver	J-Link驱动程序，下载网址: <a href="http://www.segger.com/downloads/jlink/">www.segger.com/downloads/jlink/</a>
Keil MDK5	IDE工具，下载网址: <a href="http://www.keil.com/download/product/">www.keil.com/download/product/</a>
GRTtoolbox（Android）	GR551x BLE调试工具，位于SDK_Folder\tools\GRTtoolbox
GProgrammer（Windows）	GR551x Programming工具，位于SDK_Folder\tools\GProgrammer
Keysight 14585A	Keysight公司推出的用于电源控制与分析的软件

#### 3.2 硬件连接

1. 打开Keysight N6705C，用图 3-1中B处的按钮将通道的电压输出调整到3.3 V，即图 3-1中C处显示3.3 V。
2. 将图 3-1中A处的“+”连接到GR5515 SK板的VBAT端（开发板上的J10 Pin 2，如图 3-2所示）；“-”连接到GR5515 SK板的GND端，为开发板上GR551x 芯片供电。

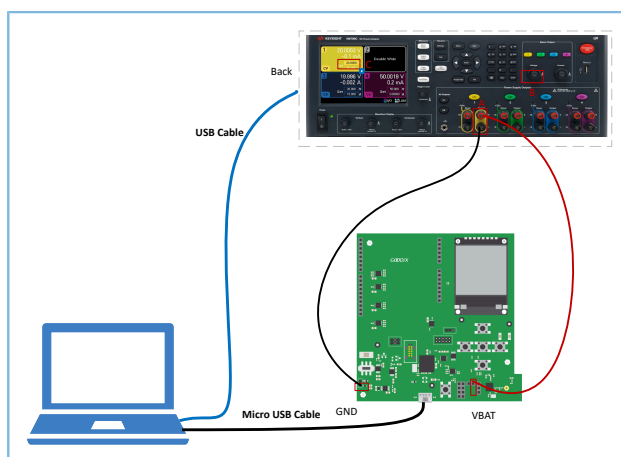


图 3-1 硬件连接示意图

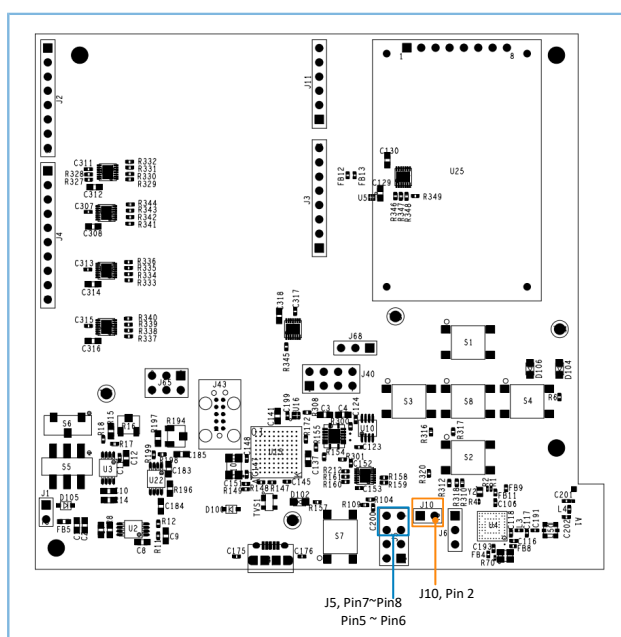


图 3-2 GR5515 SK板的硬件布局（顶层图）

3. 将GR5515 SK板Micro USB接口接到PC机的任意USB端口，为开发板芯片外围电路供电。
4. 通过USB线将Keysight N6705C与PC机连接。

#### 说明:

在GR5515 SK板测量功耗前，需要先将如图 3-2所示J5上Pin 5 ~ Pin 6、Pin 7 ~ Pin 8上的跳线帽移除，并取下SK板右上方的LCD屏幕。避免额外电流从VDDIO电源泄露，导致测量睡眠功耗异常。

## 3.3 下载固件

下载ble\_app\_pcs\_fw.bin固件至开发板的具体操作方法，请参考 [《GProgrammer用户手册》](#)。



#### 说明:

`ble_app_pcs_fw.bin`位于`SDK_Folder\projects\ble\ble_peripheral\ble_app_pcs\build`，其中`SDK_Folder`为GR551x SDK的根目录。

## 3.4 测试验证

下载有`ble_app_pcs_fw.bin`固件的GR5515 SK开发板上电后，将进入Ultra Deep Sleep状态。复位后，长按开发板上的“OK”键3秒以上，系统将发起持续时长为30秒的广播。此后，如果SK板没被其他设备连接，将会广播超时再次进入Sleep状态；如果SK板被其他设备连接，在断连后，也会进入Sleep状态，直至被再次唤醒。按“OK”键可将SK板从Sleep状态唤醒。

#### 说明:

GR5515 Starter Kit开发板按键的详细信息，请参考[《GR5515 Starter Kit用户指南》](#)“7 按键和指示灯”。

### 3.4.1 设置测量应用场景

利用手机端工具GRToolbox设置测量场景，具体操作步骤如下：

1. 打开GRToolbox APP，选择“应用 > PCS”。

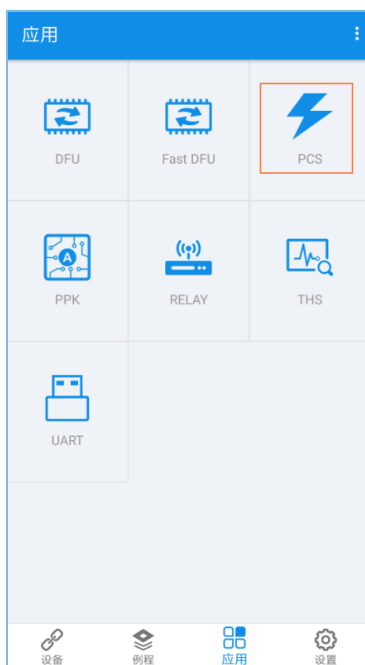


图 3-3 选择“PCS”

2. 连接后开始扫描目标设备。发现广播名为“Goodix\_Power”的设备（广播名可在`user_app.c`文件中进行修改）。

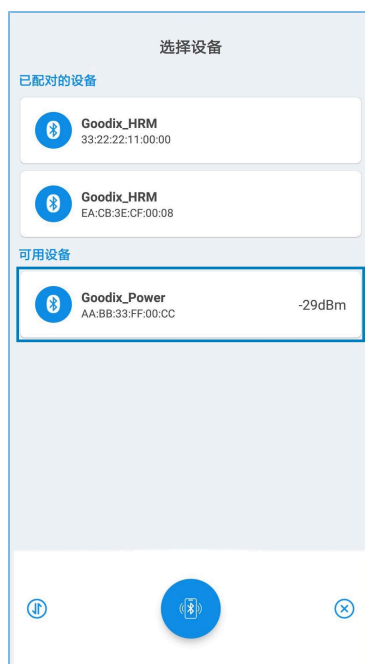


图 3-4 发现 “Goodix\_Power”

3. 点击 “Goodix\_Power” 选项并连接，进入功耗测试相关场景设置页面，包括广播间隔、广播数据、连接参数、传输模式、传输功率和开启通知，如图 3-5 所示。其中，广播间隔、广播数据和传输功率的设置断连后，重启广播时生效，而连接参数和传输模式的设置仅在当前连接生效。“最近连接设备”用于在无法通过广播名和服务UUID搜索到设备时根据前一次的设备MAC地址进行搜索。



图 3-5 功耗测试场景设置

4. 断开连接后，按下开发板上的 “OK” 键，设备会以设置好的数据长度和连接间隔重新发起广播。

### 3.4.2 观测GR551x功耗

测量场景设置完成后，可利用PC端工具Keysight，观测不同场景下GR551x功耗。

场景一：1s间隔的广播态。

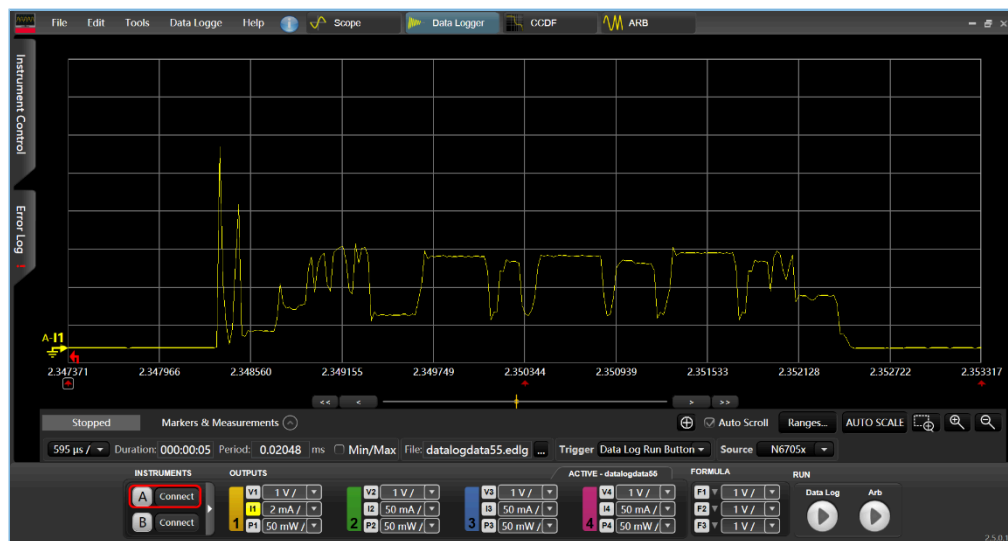


图 3-6 功耗测试场景一

场景二：200 ms间隔的连接态。

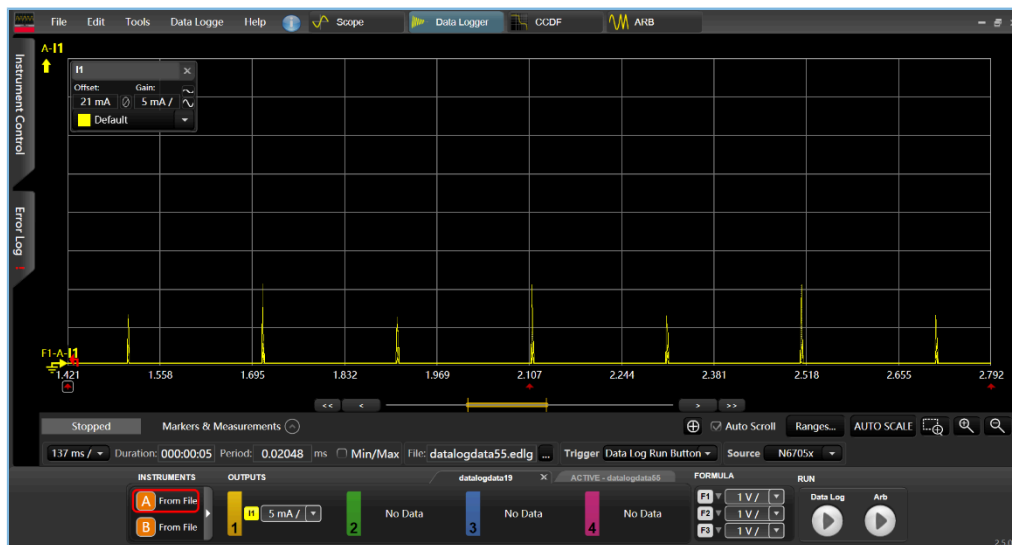


图 3-7 功耗测试场景二

其他场景的功耗测试在此不再赘述，请根据需要进行设置。

## 4 应用详解

本章主要介绍PCP示例的工程目录、交互流程，以及工程中的部分核心代码。

### 4.1 工程目录

PCP应用的源代码和工程文件位于SDK\_Folder\projects\ble\ble\_peripheral\ble\_app\_pcs，其中工程文件位于Keil\_5文件夹。

双击工程文件`ble_app_pcs.uvprojx`，在Keil中查看`ble_app_pcs`示例的工程目录结构，相关文件说明如表 4-1 所示。

表 4-1 ble\_app\_pcs文件说明

Group	文件	描述
gr_profiles	ble_prf_utils.c	Profile相关操作工具
	pcs.c	功耗Services
user_callback	user_gap_callback.c	获取连接、连接参数更新和PHY更新事件
user_platform	user_periph_setup.c	设备地址及Sleep等参数设置
user_app	main.c	main()入口函数
	user_app.c	功耗Service注册及相关事件处理

### 4.2 实现流程及代码介绍

下载PCP示例的开发板上电后，会依次执行外设及电源管理初始化、BLE协议栈初始化、PCS初始化等相关操作，主要流程如图 4-1所示：

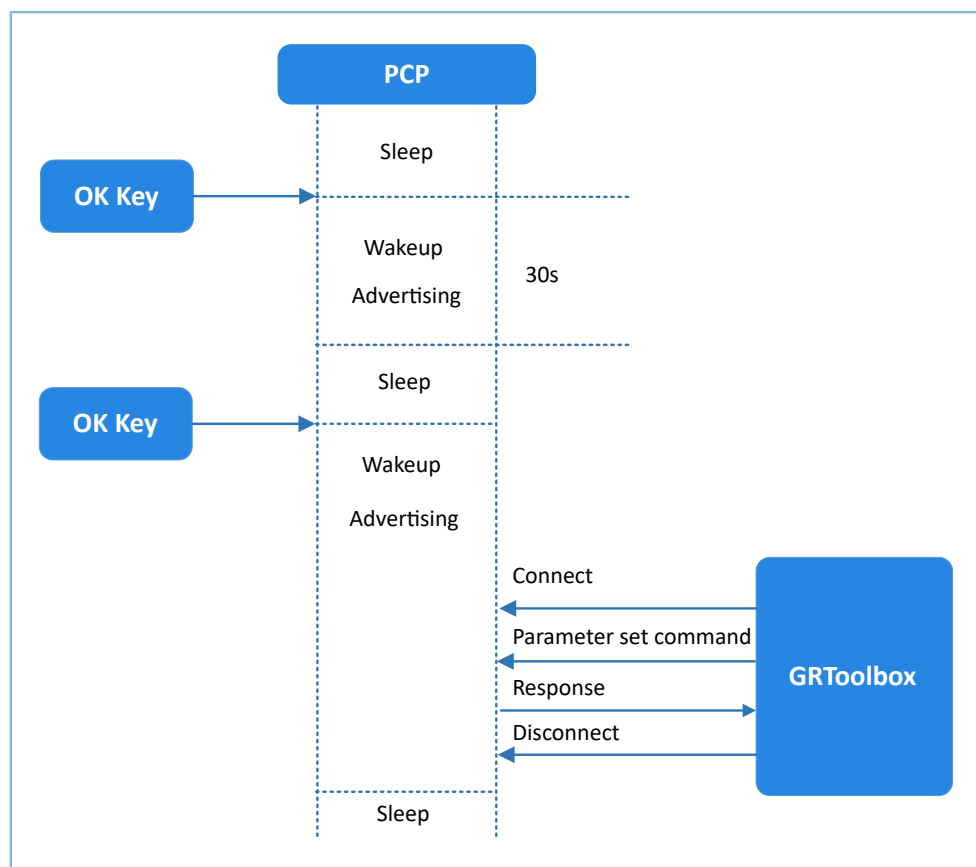


图 4-1 实现流程图

#### 说明:

PCP示例的主要逻辑代码位于:

Keil工程目录树中, user\_platform下的user\_periph\_setup.c文件。

Keil工程目录树中, user\_app下的main.c和user\_app.c文件。

下文详细描述了GR5515 Starter Kit开发板与GRTtoolbox的交互过程。

#### 1. 电源管理配置。

(1) 配置GR551x电源管理模式为睡眠模式 (PMR\_MGMT\_SLEEP\_MODE) 及外部唤醒源 (板载“OK”键), 用于唤醒GR551x开启广播。代码片段如下所示。

```
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_250HM);
    wkup_key_init();
    pwr_mgmt_mode_set(PMR_MGMT_SLEEP_MODE);
}
```

(2) 用于判断系统启动后是进入Ultra Deep Sleep，还是进行正常业务逻辑，在main()函数中调用。

```
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) main()函数判断系统启动后流程分支，以及低功耗管理。

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

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

    ble_stack_init(&s_app_ble_callback, &heaps_table);

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

## 2. 指令解析、执行与回应。

当Setting Characteristic Value接收到对端发送的指令数据时，会将事件与相关信息上报至应用层，可利用pcs\_param\_parse()函数对其指令进行解析、执行和回应。功耗测试相关场景设置，说明如下。

- 设置广播间隔

设置广播间隔值，然后回应对端设备。该参数值将在下次开启广播时生效。具体代码见pcs\_param\_parse函数中对PCS\_SETTING\_TYPE\_ADV\_INTERVAL事件的处理代码。

```

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;
    }
}

```

- 设置广播数据

设置广播数据，并根据执行结果回应对端设备。可设置的长度分别为3、10、17、24和31字节。该设置在下次开启广播时生效。具体代码见pcs\_param\_parse函数中对PCS\_SETTING\_TYPE\_ADV\_DATA事件的处理代码。

#### 说明:

通过ble\_gap\_adv\_param\_set()设置广播数据时，由于Advertising Type Flag会占用3个字节的广播数据长度，因此在设置用户广播数据时需减去3个字节有效长度。

```

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;
}
}

```

- 设置连接参数

设置当前链路的连接间隔、从设备时延和监控超时值。**ble\_gap\_conn\_param\_update**为异步函数，会在**app\_gap\_connection\_update\_cb()**回调中根据执行结果回应对端设备。

若设置成功，则在当前连接即可生效。具体代码见**pcs\_param\_parse()**函数中对**PCS\_SETTING\_TYPE\_CONN\_PARAM**事件的处理代码。

```

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;
}
}

```

- 设置传输模式

当前连接链路可更新为1M、2M或Coded PHY。ble\_gap\_phy\_update为异步函数，会在app\_gap\_phy\_update\_cb()回调中根据执行结果回应对端设备。具体代码见pcs\_param\_parse()函数中对PCS\_SETTING\_TYPE\_PHY事件的处理代码。

```

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;
    }
}

```

- 设置传输功率

设置设备的当前连接发射功率，以及下次广播的发射功率，同步返回执行结果，并回应对端设备。具体代码见pcs\_param\_parse()函数中对PCS\_SETTING\_TYPE\_TX\_POWER事件的处理代码。

```

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. 开启通知。

当对端设备开启通知（即对其CCCD写值0x0001），示例应用在收到PCS\_EVT\_TX\_ENABLE事件后开始发送Notify数据；一次数据发送完成后，示例应用收到PCS\_EVT\_DATA\_SENT后再次Notify数据，直到收到PCS\_EVT\_TX\_DISABLE才停止Notify数据。

```

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);  
}
```