BLE Profile 设计说明

发布 ***1.0.0***

2018 年 **11** 月 **02** 日

目录

CHAPTER 1

概述

1.1 文档目的

通过一个简单的示例介绍 BLE Profile 的编写方法，供用户学习参考。

1.2 术语说明

表 1.1: 术语说明

术语	说明
BLE	蓝牙低功耗技术
GAP	通用访问配置文件
GATT	通用属性配置文件
PROFILE	本文特指通用属性配置文件
SERVER	服务器
CLIENT	客户端
BAS	电池服务
ZEPHYR	一款开源的实时操作系统

1.3 参考文档

- https://www.bluetooth.com/specifications/gatt

1.4 版本历史

表 1.2: 版本历史

日期	版本	注释	作者
2018-08-24	1.0	初始版本	ZS110A 项目组

CHAPTER 2

BLE Profile 概述

2.1 GATT 简介

通常说的 BLE Profile 即为 GATT profile，全名 Generic Attribute profile，中文名通用属性配置文件。该规程定义了服务、特征及其描述符被发现、继而用于允许 BLE 设备传输数据的标准途径。该配置文件以 GATT 功能为基础，描述了用例、角色和一般行为。

GATT 定义了两种角色：GATT Server 和 GATT Client。GATT Server 是向 GATT Client 提供数据服务的设备，GATT Client 是从 GATT Server 读写应用数据的设备。

2.2 GATT 的层次结构

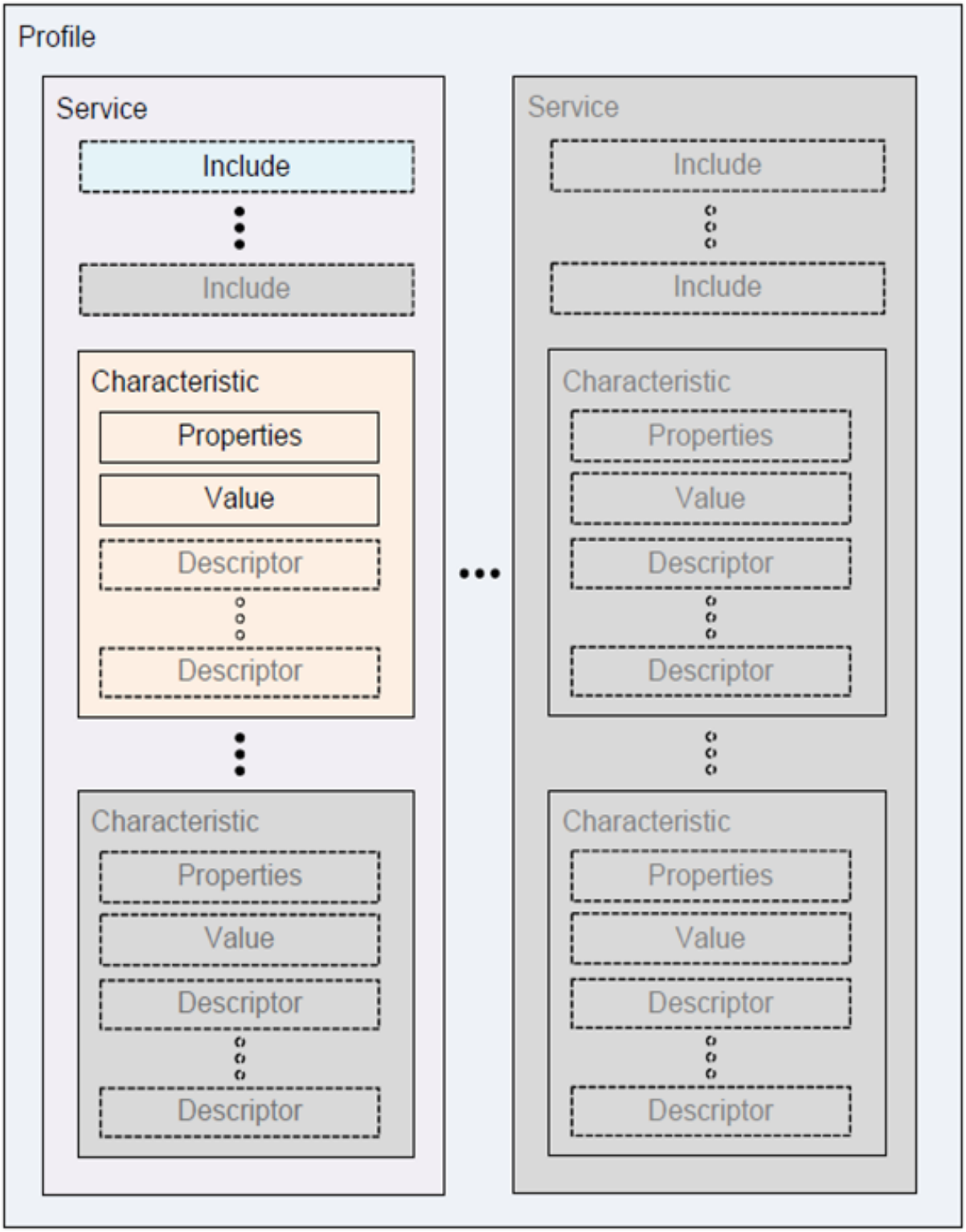


图 2.1: GATT 的层次结构

层次结构的最顶层为 Profile，由完成用例时必需的一个或多个 Service 组成。一个 Service 由 Characteristic 和对其他服务的 Reference 组成。Characteristic 包括 type (表

现为 UUID)、value 和一组指示 Characteristic 所支持操作的 Properties 以及一组与安全相关的 Permissions。Characteristic 还可能包括一个或多个 Descriptors, 即与所拥有 Characteristic 相关的元数据或配置标志。

更多信息请登录 Bluetooth 官网:

- GATT Profile: https://www.bluetooth.com/specifications/gatt
- GATT Service: https://www.bluetooth.com/specifications/gatt/services

CHAPTER 3

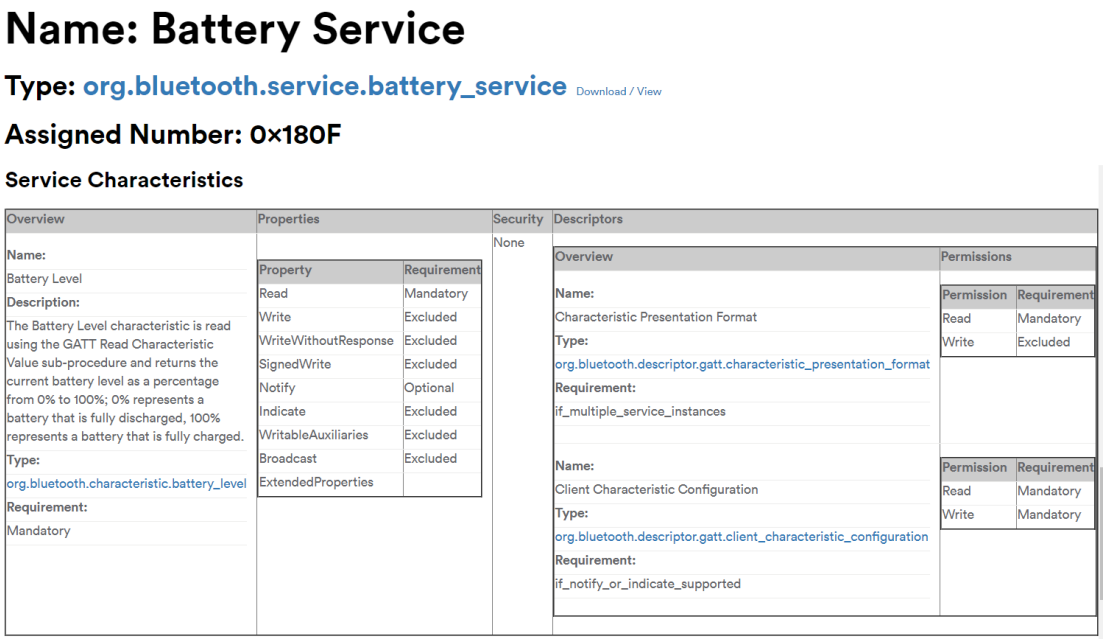
BLE Profile 编写

下面以电池服务 (BAS) 说明标准的 BLE Profile 的编写方法。

3.1 准备工作

- 下载 BAS Profile 规格: https://www.bluetooth.com/specifications/gatt
- 查看 BAS Service : https://www.bluetooth.com/specifications/gatt/services

下面主要以 BAS Service 讲述 BLE Profile 的编写，更为详细的信息请自行阅读 BAS Profile 规格《BAS_SPEC_V10.pdf》。



Name: Battery Service

Type: org.bluetooth.service.battery_service Download / View

Assigned Number: 0x180F

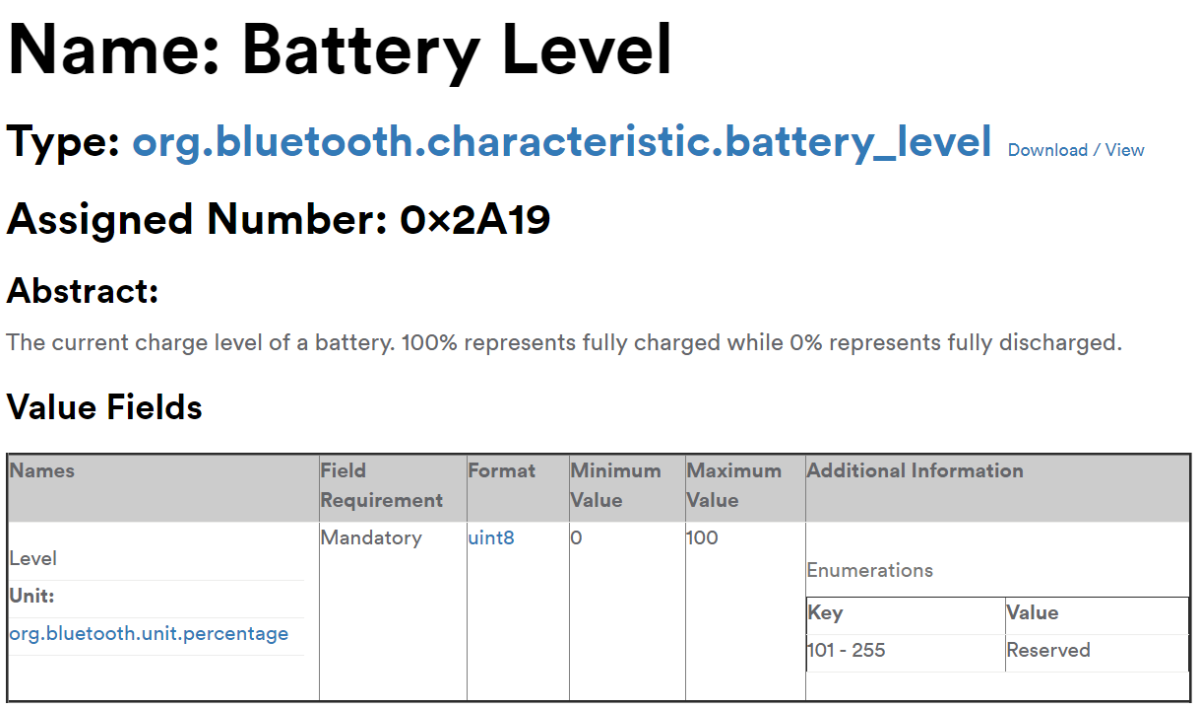
Service Characteristics

Overview	Properties	Security	Descriptors
Name: Battery Level **Description:** The Battery Level characteristic is read using the GATT Read Characteristic Value sub-procedure and returns the current battery level as a percentage from 0% to 100%; 0% represents a battery that is fully discharged, 100% represents a battery that is fully charged. **Type:** org.bluetooth.characteristic.battery_level **Requirement:** Mandatory	Property / Requirement Read — Mandatory Write — Excluded WriteWithoutResponse — Excluded SignedWrite — Excluded Notify — Optional Indicate — Excluded WritableAuxiliaries — Excluded Broadcast — Excluded ExtendedProperties	None	Overview: **Name:** Characteristic Presentation Format; **Type:** org.bluetooth.descriptor.gatt.characteristic_presentation_format; **Requirement:** if_multiple_service_instances Permissions: Permission / Requirement — Read — Mandatory; Write — Excluded Overview: **Name:** Client Characteristic Configuration; **Type:** org.bluetooth.descriptor.gatt.client_characteristic_configuration; **Requirement:** if_notify_or_indicate_supported Permissions: Permission / Requirement — Read — Mandatory; Write — Mandatory

图 3.1: BAS Service

查看 BAS Service 可知它的 UUID 是 0x180F，根据 Service Characteristics 一栏可知 BAS 仅有一个特性 Battery Level，该特性有一个强制的 Read 属性和一个可选的 Notify 属性。因此我们必须实现特性的 Read 函数，可选的实现特性的 Notify。

点击 org.bluetooth.characteristic.battery_level 查看特性的类型:



Name: Battery Level

Type: org.bluetooth.characteristic.battery_level Download / View

Assigned Number: 0x2A19

Abstract:

The current charge level of a battery. 100% represents fully charged while 0% represents fully discharged.

Value Fields

Names	Field Requirement	Format	Minimum Value	Maximum Value	Additional Information
Level **Unit:** org.bluetooth.unit.percentage	Mandatory	uint8	0	100	Enumerations Key: 101 - 255 — Value: Reserved

图 3.2: battery level 类型

由以上信息可知，Battery Level 特性的 UUID 是 0x2A19，类型是 uint8，单位是百分比，最小值 0，最大值是 100，值 101~255 为保留。

再查看 Descriptor 一栏，Battery Level 特性有两个描述符，一个 Characteristic Presentation Format，一个 Client Characteristic Configuration。根据 Requirement，Characteristic Presentation Format 用于多服务实例，此处未用到，Client Characteristic Configuration 用于 notify 或 indicate，当前电池服务电量需要自动上报，所以使用到该描述符，该描述符有一个强制的读属性和一个强制的写属性。

点击 org.bluetooth.descriptor.gatt.client_characteristic_configuration 查看 Client Characteristic Configuration 的类型，得到其 UUID 是 0x2902，值是 16 位数据类型，bit 0 表示 Notifications 禁止/开启，bit 1 表示 Indications 禁止/开启。

3.2 代码编写

所有 BLE Profile 编写大致都分为两个过程，声明服务和注册服务，特性的声明和操作函数包含在声明服务的过程中，对于 BAS，有如下两个过程:

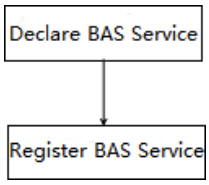


图 3.3: BAS 流程

1. 声明 BAS 服务

   根据前面的分析，BAS 服务 UUID 是 0x180F，该服务有一个 Battery Level 的特性，它的 UUID 是 0x2A19，值的类型是 uint8，它有一个读属性和通知属性，因此需要实现一个读函数，且当电池电量发生改变时，若使能了通知，还要上报电池电量。该特性还包括两个描述符，一个是 Characteristic Presentation Format，另一个是 Client Characteristic Configuration，这里只用到 Client Characteristic Configuration，它的作用是开启或禁止 Notification/Indication。

   在 Zephyr 中声明一个 GATT 服务很简单，只需要构造一个 struct bt_gatt_attr 的数组，并将需要声明的服务和该服务所包含的特性及其描述符以数组项的形式加入即可。

   在此过程中要用到如下几个宏：

   - BT_GATT_PRIMARY_SERVICE：声明一个主服务
   - BT_GATT_SECONDARY_SERVICE：声明一个次要服务
   - BT_GATT_CHARACTERISTIC：声明一个特性
   - BT_GATT_DESCRIPTOR：声明一个描述符
   - BT_GATT_CCC：声明一个客户端特性描述符 (CCC)

   更多 GATT 属性声明及其使用方法请参考 include/bluetooth/gatt.h。

   根据以上信息，就可写出如下代码：

```
#define BT_UUID_BAS                       BT_UUID_DECLARE_
↪16(0x180f)
#define BT_UUID_BAS_VAL                   0x180f
#define BT_UUID_BAS_BATTERY_LEVEL         BT_UUID_DECLARE_
↪16(0x2a19)
#define BT_UUID_BAS_BATTERY_LEVEL_VAL 0x2a19

static struct bt_gatt_ccc_cfg  blvl_ccc_cfg[BT_GATT_CCC_
↪MAX] = {};
static u8_t simulate_blvl;
static u8_t battery = 100;

/* 当客户端写入的值是 BT_GATT_CCC_NOTIFY(该宏值为 1) 时，开
启通知，否则关闭通知 */
static void blvl_ccc_cfg_changed(const struct bt_gatt_
↪attr *attr,  u16_t value) {
        simulate_blvl = (value == BT_GATT_CCC_NOTIFY) ?␣
↪1 : 0;
}

/* Battery Level 读操作函数，返回当前的电量 */
static ssize_t read_blvl(struct bt_conn *conn, const␣
↪struct bt_gatt_attr *attr,
        void *buf, u16_t len, u16_t offset) {
        const char *value = attr->user_data;
        return bt_gatt_attr_read(conn, attr, buf, len,␣
↪offset, value,  sizeof(*value));
}

/* Battery Service Declaration */
static struct bt_gatt_attr attrs[] = {
        /* 声明 BAS 为服务 */
        BT_GATT_PRIMARY_SERVICE(BT_UUID_BAS),

        /* 声明特性 Battery Level，属性为读和通知 */
        BT_GATT_CHARACTERISTIC(BT_UUID_BAS_BATTERY_
↪LEVEL,
                BT_GATT_CHRC_READ | BT_GATT_CHRC_
↪NOTIFY),

        /* 声明特性值描述符，有读属性权限，read_blvl 是读操
作函数，battery 是当前电量 */
        BT_GATT_DESCRIPTOR(BT_UUID_BAS_BATTERY_LEVEL,␣
↪BT_GATT_PERM_READ,
                read_blvl, NULL, &battery),

        /* 声明客户端特性描述符，当客户端发起 ccc 写操作时，
blvl_ccc_cfg_changed 会被调用 */
        BT_GATT_CCC(blvl_ccc_cfg,  blvl_ccc_cfg_
↪changed),
```

(continues on next page)

(续上页)

```
};
```

2. 注册服务

   服务声明完成后，需要通过调用 bt_gatt_service_register 完成注册:

```
void bas_init(void)
{
        bt_gatt_service_register(&bas_svc);
}
```

3. 电量通知

   电量通知，即在电池电量改变时，主动上报电池电量，而无需客户端手动去读取电池电量。下面的示例程序通过软件模拟的方式模拟电池电量，并发送通知，进行上报。

```
void bas_notify(void)
{
    /* simulate_blvl 在通知使能时被置 1 */
    if (!simulate_blvl) {
        return;
    }
    /* 软件模拟电池电量 */
    battery--;
    if (!battery) {
        battery = 100;
    }
    /* 发送通知，上报电量 */
    bt_gatt_notify(NULL, &attrs[2], &battery,
↪sizeof(battery));
}
```

4. 使用服务

   现在 BAS 服务已编写完成，剩下的工作就是在合适的地方调用 bas_init 函数将服务注册进蓝牙协议栈，并在电池电量改变时调用 bas_notify 实现电量的上报。下面贴上 samples\bluetooth\peripheral_bas\src\main.c 代码方便理解:

```
#define DEVICE_NAME                CONFIG_BT_DEVICE_NAME
↪ /* 设备名 */
#define DEVICE_NAME_LEN        (sizeof(DEVICE_NAME) - 1)
struct bt_conn *default_conn;

/* 广播数据 */
static const struct bt_data ad[] = {
    BT_DATA_BYTES(BT_DATA_FLAGS, (BT_LE_AD_GENERAL |
    BT_LE_AD_NO_BREDR)),
    BT_DATA_BYTES(BT_DATA_UUID16_ALL, 0x0f, 0x18), /*
↪bas */
```

(continues on next page)

(续上页)

```
};

/* 扫描回应数据 */
static const struct bt_data sd[] = {
    BT_DATA(BT_DATA_NAME_COMPLETE, DEVICE_NAME, DEVICE_
↪NAME_LEN),
};

/* connected 回调 */
static void connected(struct bt_conn *conn, u8_t err)
{
    if (err) {
        printk("Connection failed (err %x)\n", err);
    } else {
        default_conn = bt_conn_ref(conn);
        printk("Connected\n");
    }
}

/* disconnected 回调 */
static void disconnected(struct bt_conn *conn, u8_t␣
↪reason)
{
    printk("Disconnected (reason %x)\n", reason);
    if (default_conn) {
            bt_conn_unref(default_conn);
            default_conn = NULL;
    }
}

/* 连接回调结构 */
static struct bt_conn_cb conn_callbacks = {
        .connected = connected,
        .disconnected = disconnected,
};

u8_t bt_ready_flag;
__init_once_text static void bt_ready(int err)
{
        if (err) {
                printk("Bluetooth init failed (err %d)\n
↪", err);
                return;
        }
        bt_ready_flag = 1;
        printk("Bluetooth initialized\n");

        /* 初始化电池服务 */
        bas_init();
```

(continues on next page)

(续上页)

```
        /* 启动广播 */
        err = bt_le_adv_start(BT_LE_ADV_CONN, ad, ARRAY_
↪SIZE(ad),
                              sd, ARRAY_SIZE(sd));
        if (err) {
                printk("Advertising failed to start␣
↪(err %d)\n", err);
                return;
        }
        printk("Advertising successfully started\n");
}

/* 应用程序主入口 */
void app_main(void)
{
        int err;

        /* 蓝牙使能 */
        err = bt_enable(bt_ready);
        if (err) {
                printk("Bluetooth init failed (err %d)\n
↪", err);
                return;
        }

        /* 注册连接回调 */
        bt_conn_cb_register(&conn_callbacks);

        /* 每秒上报一次电池电量 */
        while (1) {
                k_sleep(MSEC_PER_SEC);  // 休眠 1s
                bas_notify();  // 发送通知
        }
}
```

CHAPTER 4

BLE Profile 验证

打开写好的 BAS 服务，路径:samples\bluetooth\peripheral_bas 编译，烧写完成后，重启开发板，系统将输出如下信息:

```
shell> BLE startup event
resetReason 03
flags 0002
maxSize 256
hwId 00000001
hwRev 02040105
swRev 00002001
swVerStr 20180524-g1f8bda7
complete
hci_vs_configuration_set_init completed
hci_vs_test_set_init completed
hci_vs_extension_set_init completed
le_read_buffer_size_complete le_max_num:4,(4,7)
[bt] [INF] show_dev_info: Identity: 11:22:33:77:55:66 (public)
[bt] [INF] show_dev_info: HCI: version 4.2 (0x08) revision 0x0120, manufacturer 0x01af
[bt] [INF] show_dev_info: LMP: version 4.2 (0x08) subver 0x0120
Bluetooth initialized
Advertising successfully started
```

图 4.1: BAS 串口输出信息

打开 LightBlue，找到 Actions BAS 并连接该设备。连接成功后，使能通知，电池量将每秒上报一次, 如图所示:

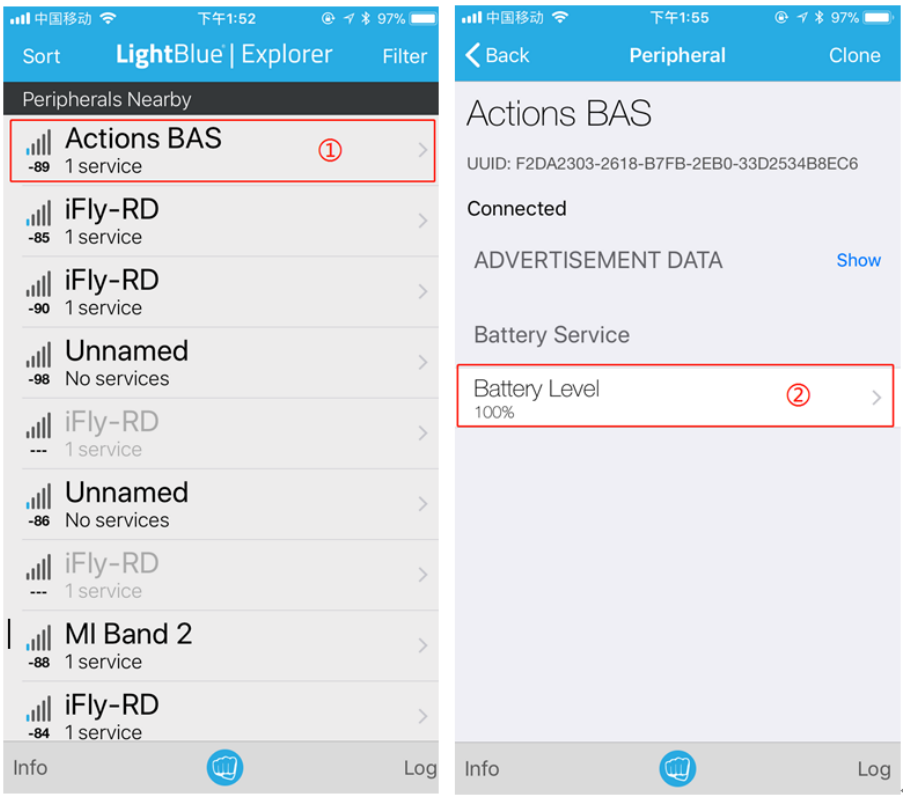


图 4.2: BAS APP 验证 - 1

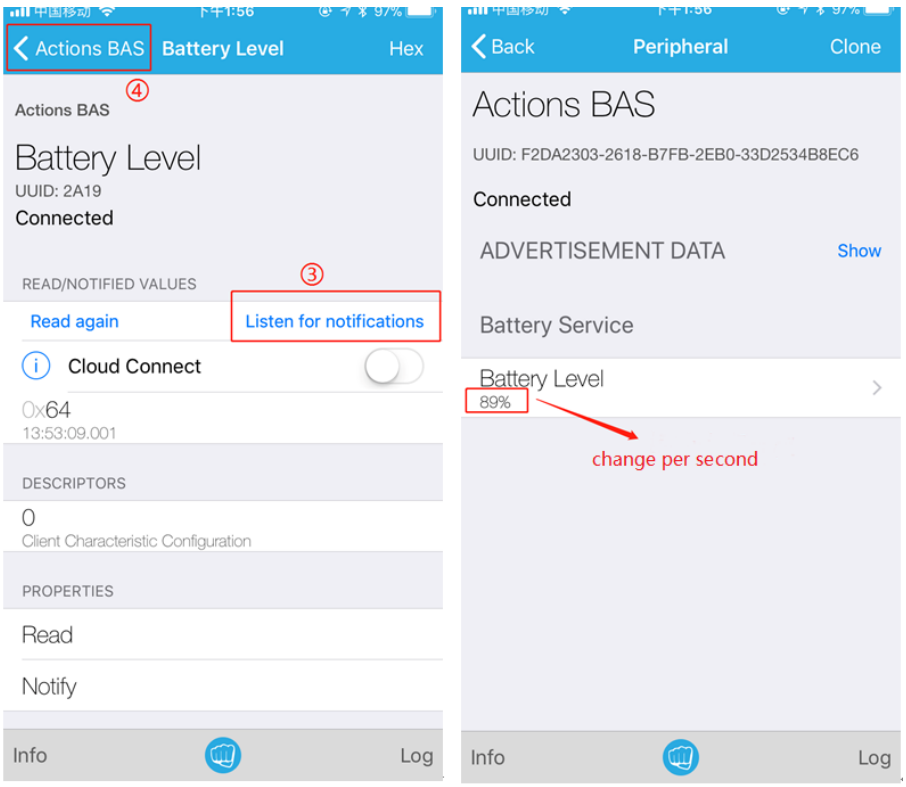


图 4.3: BAS APP 验证 - 2

List of Figures

List of Tables