青莲云蓝牙 BLE MESH SDK 开发使用文档

版本	编写/修订说明	修订人	修订日期	备注
1.0.0	创建文档	张乐嘉	20190919	

目录

1 概要

青莲云作为领先的物联网安全服务商，为了让开发者不必关心数据的加密传输、网络链路的安全通信，适配了一系列 wifi、蓝牙、NB-IOT 模组。

我们提供了不同平台的嵌入式 SDK，开发者可以通过简单的调试，立即拥有强大的后端云能力，专注于具体业务研发。

本 SDK 提供以下功能的接口说明：

1) 连接状态报告

2) 网络时间同步

3) 实时数据上报

4) 实时发布数据（BLE MESH 特性）

5) 实时获取命令

6) 实时自定义消息推送

7) 针对 BLE MESH 特性的极速传输模式

8) 本地存储/加载数据

2 sdk 目录结构

-\SDK

-\lib

-\libiot_ble_mesh_platform_os.a 正式版 sdk

-\libiot_ble_ mesh_platform_os_debug.a debug 版 sdk

-\include

-\iot_ble_ mesh_interface.h	云端对外接口
-\iot_cjson.h	cjson 库
-\iot_md5.h	md5 库
-\iot_base64.h	base64 库
-\src	
-\ iot_ble_mesh_demo.c	使用样例
-\iot_ble_ mesh_interface.c	用户回调函数实现
-\makefile	
-readme.txt	相关说明

3 系统函数

3.1 初始化 SDK

初始化设备与 APP 交互的上下文环境。注意，填写秘钥时，需将官网的一串字符串转换成相应的十六进制编码，即在每个字节前增加 0x 作为开头。如字符串是 5668，转换时应改为 0x56，0x68。

iot_s32 iot_start(struct iot_context* ctx);

struct iot_context 结构体的内容如下：

参数	长度	说明
product_id	4	产品 ID，云平台生成，4 字节的无符号整型数字
product_key	16	产品秘钥，云平台生成，16 字节的十六进制编码
mcu_version	5	mcu 固件版本，"xx.xx"，0≤x≤9
recvbuf_size	4	接收数据 buffer 大小，范围 128-377
sendbuf_size	4	发送数据 buffer 大小，范围 128-377
返回值	4	0 ：成功； -1：失败

3.2 设置设备运行状态

```
void        iot_status_set(     DEV_STATUS_T   dev_status,
                                iot_u32        timeout     )
```

参数	说明
dev_status	DEV_STA_UNBIND：设置解绑，设备会解除与 APP 端的绑定关系。 操作成功后自动调用 3.3 节的回调函数
timeout	设备运行状态保持时长，单位为秒; timeout = 0 表示不设置超时状态。

下表为 3.2 中设置参数与 3.3 中回调参数的对应关系。

设置运行状态	设置参数（3.2 章节）	回调参数（3.3 章节）
解除绑定	DEV_STA_UNBIND	解绑成功： DEV_STA_UNBIND （4）

3.3 设备运行状态报告

当设备运行状态发生改变时，sdk 自动调用此函数。回调函数中不可执行耗时的代码。

```
void        iot_status_cb (     DEV_STATUS_T   dev_status,
                                iot_u32        timestamp   )
```

参数	说明

dev_status	DEV_STA_BLE_MESH_PROV_RESET ：设备配置被 provisioner 重置 DEV_STA_BLE_MESH_PROV_COMP ：设备 provision 完成 DEV_STA_BLE_APP_CONFIRM ：与 APP 注册认证通过 DEV_STA_UNBIND ：解绑成功
timestamp	状态改变的时间点

3.4 获取网络时间

iot_u32 iot_get_onlinetime(void)

参数	说明
返回值	0：时间无效 >0：实时网络时间戳

3.5 打印输出函数

此函数用于输出日志信息，需用户自行实现，可根据需要重定向到串口、屏幕、文件等位置。

void iot_print (const char * str)

参数	说明
str	输出的日志内容

3.6 发送数据回调函数

发送数据成功后，sdk 自动调用下面这个回调函数。当调用 4.1 节上传数据、6.1 节透传自定义数据。回调函数中不可执行太多耗时代码。

void iot_upload_data_cb (iot_u32 data_seq)

参数	说明
data_seq	某条数据的序列号，如不关心何时上传成功，可不做处理

3.7 获取时间结构体

void iot_parse_timestamp (iot_u32 tick , struct s_time* st);

参数	说明
tick	需要被转换的时间戳
st	转换后的时间结构体

时间结构体定义如下:

```
struct s_time {
    int sec;                    //秒
    int min;                    //分
    int hour;                   //时
    int day;                    //日
    int mon;                    //月
    int year;                   //年
    int week;                   //周
};
```

4 传输数据

发送/接收数据的最大长度与 iot_ctx 中的 buffer 的大小有关。

如：buffer 设置为 1024，以整型类型数据点为例，最多可以一次性发送 70 个整型数据。

4.1 上传数据

- 开发者需明确每个数据点的 dpid、类型(通过云平台获取)。根据数据点的类型，调用不同的函数将数据点 id、对应数值添加到发送队列中。
- 目前支持的类型包括整数型、布尔型、枚举型、浮点型、字符型、故障型、二进制。
- 上传数据时，需保证在云端创建的数据点是可上报的。

① 添加数据点到发送队列

```
iot_s32   dp_up_add_int   (   iot_u8   dpid,     iot_s32  value)
iot_s32   dp_up_add_bool (   iot_u8   dpid,     iot_u8   value)
iot_s32   dp_up_add_enum(   iot_u8   dpid,     iot_u8   value)
iot_s32   dp_up_add_float (   iot_u8   dpid,     iot_f32  value)
iot_s32   dp_up_add_string(   iot_u8   dpid,
                              const    char*    str,
                              iot_u32  str_len   )
iot_s32   dp_up_add_fault (   iot_u8   dpid,
                              const    char*    fault,
                              iot_u32  fault_len      )
iot_s32   dp_up_add_binary(   iot_u8   dpid,
                              const    iot_u8*  bin,
```

iot_u32 bin_len)

② 上传一条数据，设备功能发生改变时将最新数据上传，一条数据可包含多个数据点。

iot_s32 iot_upload_dps(iot_u16* data_seq)

参数	说明
data_seq	传出参数，本条数据的序列号，如果需要确定数据何时上传成功，可记录此发送序列号，与收到的进行对比。
返回值	0 ：成功； -1：失败

4.2 发布数据

- publish 是 BLE MESH 网络中的一种特殊传输数据的形式，具体表现为某个设备可以设置 publish 地址例如 0xc0001，其他多个设备 subscribe 地址 0xc0001，如果该设备向其 publish 地址发送消息，那么所有 subscribe 该地址的设备都会接收到 publish 消息。
- 在我们的 BLE MESH SDK 中建议是接收到 APP 发送的数据后设备主动调用 publish 接口。
- 目前该 publish 接口限制发送的数据不超过 7 字节，否则传输速率会很慢。

① 添加数据点到发送队列

请参考 4.1 章节说明

② 发布一条数据

iot_s32 iot_publish_dps(iot_u16* data_seq);

参数	说明
data_seq	传出参数，本条数据的序列号，如果需要确定数据何时上传成功，可记录此发送序列号，与收到的进行对比。
返回值	0 ： 成功； -1：失败

4.3 接收数据

- 开发者需明确每个数据点的 dpid、类型(通过云平台获取)。根据数据点的类型，调用不同的转换函数转换成需要的数值。
- 目前支持的类型包括整数型、布尔型、枚举型、浮点型、字符型、故障型、二进制。
- 接收数据时，需保证在云端创建的数据点是可下发的。
- 接收数据时会判断是否是 APP 下发的命令，通常 APP 下发的命令需要做 publish 操作。

① 填写以下数据结构

```
iot_download_dps_t      iot_down_dps[]  =
{       云端数据点 ID              数据点类型              处理函数
    { DP_ID_DP_SWITCH ,    DP_TYPE_BOOL   ,   dp_down_handle_switch    },
};
```

③ 定义针对某个数据点的处理函数

```
函数类型为   typedef dp_down__handle*(  void*    indata,
                                        iot_u32 inlen,
                                        iot_u8   is_app   )
```

如数据点 switch 的类型为 bool，则接受其数据的处理函数实现如下：

```
void      dp_down_handle_switch (      iot_u8*  in_data,
                                       iot_u32  inlen,
                                       iot_u8    is_app    )
{
    iot_u8    dp_switch    =    bytes_to_bool(    in_data  );    //转换成对应数值
    if(    dp_switch    ==  0    )
    {      }
    else
    {      }
    //如果是 APP 发送过来的数据则将该数据进行 publish
    if(is_app) {
        dp_up_add_bool(DP_ID_DP_SWITCH, dp_switch);
        iot_publish_dps(&seq);
    }
    //硬件操作完成后，应将更新后的状态上报一次
    dp_up_add_bool(DP_ID_DP_SWITCH, dp_switch);
    //上报自身的开关值
    iot_upload_dps(&seq);
}
```

③ 处理函数要根据数据点的类型，调用不同的转换函数

```
iot_s32        bytes_to_int (     const iot_u8        bytes[4]       );

iot_u8         bytes_to_bool (    const iot_u8        bytes[1]       );

iot_u8         bytes_to_enum (    const iot_u8        bytes[1]       );

iot_f32        bytes_to_float (   const iot_u8        bytes[4]       );
```

其他包括字符串、故障、二进制类型请直接对原始数据进行处理。

④ 接收到合法数据后，sdk 会自动调用数据点对应的处理函数，处理函数一般会操作改变设备状态。

比如收到开/关命令，处理函数中应先执行设备开/关，接着将最新的设备开关状态通过调用 4.1 节接口上传至云端。如果 is_app 等于 1 表示接受到的数据是 APP 发送的，那么就要将接受的数据 publish 出去。

5 高级功能

5.1 透传自定义数据

◆ 可透传任意格式自定义数据，请与 app 开发者自行约定

```
iot_s32  iot_tx_data (iot_u32*      data_seq,
                      iot_u8*       data,
                      iot_u32       data_len        )
```

参数	说明
data_seq	传出参数，本条数据的序列号，如果需要确定数据何时上传成功，可记录此发送序列号，与收到的进行对比。
data	自定义字符，支持字符串、二进制数据，特殊字符数据
data_len	自定义数据长度
返回值	0 ： 成功； -1： 失败

5.2 接收自定义数据

◆ 接收来自 app 的透传数据，格式请与 app 开发者自行约定

◆ 回调函数中不可执行太多耗时代码。

```
void      iot_rx_data_cb (iot_u8*       data,
                          iot_u32       data_len        )
```

参数	说明
data	自定义数据，仅支持字符型
data_len	自定义数据长度
is_app	表示数据是否是 APP 发送来的，1 表示是 APP 发送的，0 表示不是。

5.3 上传数据（极速版本）

- 功能和 4.1 差不多，因为 BLE MESH 是针对小数据量的网络协议，如果使用 4.1 的接口对于非代理节点的设备传输数据有很大的延迟，因此如果对于速度要求比较高的功能建议使用极速版本。
- 极速版本一次只能上传一个数据点的数据
- 目前支持的类型包括整数型、布尔型、枚举型、浮点型。

```
iot_s32  iot_ble_mesh_fast_tx_dp_data(    iot_u8    dp_id,
                                          const     iot_u8*  data,
                                          iot_u32  data_len      );
```

参数	说明
dp_id	数据点
data	该数据点的具体内容
data_len	数据长度

5.4 接收数据（极速版本）

- 功能和 4.3 差不多，其他请参考 5.3 说明

```
void    iot_ble_mesh_fast_rx_dp_data    (    iot_u8    dp_id,
                                             iot_u8*   data,
                                             iot_u32   data_len,
                                             iot_u8    is_app    );
```

参数	说明
dp_id	数据点
data	该数据点的具体内容
data_len	数据长度
is_app	表示数据是否是 APP 发送来的，1 表示是 APP 发送的，0 表示不是。

5.5 发布数据（极速版本）

◆ 功能和 4.2 差不多，其他请参考 5.3 说明

```
iot_s32  iot_ble_mesh_fast_publish_dp_data    (    iot_u8    dp_id,
                                                   const     iot_u8*   data,
                                                   iot_u32   data_len      );
```

参数	说明
dp_id	数据点
data	该数据点的具体内容
data_len	数据长度

5.6 保存数据至本地

iot_s32 iot_local_save (iot_u32 data_len, const void * data)

参数	说明
data_len	自定义数据长度，范围 1-4064
data	需要保存的自定义数据
返回值	0 ： 成功； -1：失败

5.7 加载本地数据

iot_s32 iot_local_load (iot_u32 data_len, void * data)

参数	说明
data_len	需要加载的数据长度，需小于实际保存的数据长度
data	需要加载的自定义数据
返回值	0 ： 成功； -1：失败，加载数据出错或 data_len 大于实际数据长度

5.8 清空本地数据

iot_s32 iot_local_reset (void)

参数	说明
返回值	0 ： 成功； -1：失败