

青连云蓝牙 BLE 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 | 20190308 | |
| 1.0.1 | 更新 OTA 相关功能 | 张乐嘉 | 20190531 | |

目录

| | | |
|-----|----------------|----|
| 1 | 概要..... | 4 |
| 2 | SDK 目录结构..... | 5 |
| 3 | 系统函数..... | 6 |
| 3.1 | 初始化 SDK..... | 6 |
| 3.2 | 设置设备运行状态..... | 7 |
| 3.3 | 设备运行状态报告..... | 7 |
| 3.4 | 获取网络时间..... | 8 |
| 3.5 | 打印输出函数..... | 8 |
| 3.6 | 发送数据回调函数..... | 9 |
| 3.7 | 获取时间结构体..... | 9 |
| 4 | 传输数据..... | 10 |
| 4.1 | 上传数据..... | 10 |
| 4.2 | 接收数据..... | 11 |
| 5 | OTA 固件升级..... | 13 |
| 5.1 | 设置 OTA 属性..... | 13 |
| 5.2 | 接收固件数据块..... | 13 |
| 5.3 | 接收升级指令..... | 14 |
| 6 | 高级功能..... | 15 |
| 6.1 | 透传自定义数据..... | 15 |
| 6.2 | 接收自定义数据..... | 15 |
| 6.3 | 保存数据至本地..... | 16 |
| 6.4 | 加载本地数据..... | 16 |

| | | |
|-----|-------------|----|
| 6.5 | 清空本地数据..... | 16 |
|-----|-------------|----|

1 概要

青连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。

本地版本和云端版本：

本文档详细介绍了蓝牙 SDK 的各个接口。蓝牙 SDK 分为“本地”和“云端”两个类型，类型是在 APP 侧来控制的，因此设备上使用相同的 SDK 代码即可。

但是两个类型的 SDK 在设备上仍然有一些差异，目前的区别仅仅在于是否支持 OTA 功能，其他功能全部一样。由于 OTA 功能是 APP 侧主动触发的，因此用户在设备上也不用做额外的处理。

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) 实时获取命令
- 5) 实时自定义消息推送
- 6) 本地存储/加载数据
- 7) 配置设备信息
- 8) OTA 升级
- 9) 获取设备信息

2 sdk 目录结构

-\SDK

 -\lib

 -\libiot_ble_platform_os.a 正式版 sdk

 -\libiot_ble_platform_os_debug.a debug 版 sdk

 -\include

 -\iot_ble_interface.h 云端对外接口

 -\iot_cjson.h cjson 库

 -\iot_md5.h md5 库

 -\iot_base64.h base64 库

 -\src

 -\main.c 使用样例

 -\iot_ble_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，填写密钥时，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 | 产品密钥, 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 512-2048 |
| sendbuf_size | 4 | 发送数据 buffer 大小, 范围 512-2048 |
| 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) |
|--------|------------------------|--|
| 恢复出厂 | DEV_STA_BLE_DISCONNECT | 设备 BLE 断开连接通知: DEV_STA_BLE_DISCONNECT (0) |
| 允许绑定 | DEV_STA_BLE_CONNECTED | 设备 BLE 连接成功通知: DEV_STA_BLE_CONNECTED (1) |
| 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太多耗时代码。

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

| 参数 | 说明 |
|------------|--|
| dev_status | DEV_STA_BLE_DISCONNECT (0) : 设备 BLE 断开连接 |

| | |
|-----------|--|
| | DEV_STA_BLE_CONNECTED (1) : 设备 BLE 连接成功 DEV_STA_BLE_APP_CONFIRM (2) : 与 APP 注册认证通过 DEV_STA_UNBIND (3) : 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 接收设备信息

```
void iot_info_cb ( INFO_TYPE_T info_type, void* info )
```

| 参数 | 说明 |
|-----------|---|
| info_type | INFO_TYPE_OTA : OTA 下载固件的信息, 包括固件类型、固件总大小、固件版本号 |
| info | 请求消息类型的数据内容 |

回调函数中不可执行太多耗时代码。该函数用来处理接收的设备信息数据, 接收的数据根据

info_type 的值数据格式不同

① 当 info_type= INFO_TYPE_OTA 时，表示获取的是 OTA 信息，数据结构体如下

```
typedef struct ota_info {
    iot_u8  owner;                //固件类型，0: wifi 固件，1: MCU 固件
    iot_u32 flen;                //固件文件大小
    char    ota_ver[6];          //OTA 固件版本号
}ota_info_t;
```

3.7 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_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，可不作处理 |

3.8 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

```
void dp_up_add_int (   iot_u8 dpid, iot_s32 value)
void dp_up_add_bool ( iot_u8 dpid, iot_u8  value)
void dp_up_add_enum(  iot_u8 dpid, iot_u8  value)
```

```

void dp_up_add_float (  iot_u8  dpid,  iot_f32  value)
void dp_up_add_string(  iot_u8  dpid,  const char  * str,  iot_u32  str_len)
void dp_up_add_fault (  iot_u8  dpid,  const char  * fault,  iot_u32  fault_len)
void 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( 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 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。

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  },
};
    
```

② 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)。

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 dp_switch = bytes_to_bool( in_data ); //转换成对应数值  
  
    if( dp_switch == 0 )  
  
        { }  
  
    else  
  
        { }  
  
}
```

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。

5 OTA 固件升级

如有远程升级需求，请参考具体 ota 升级流程文档及本小节内容进行实现，目前 OTA 固件升级功能仅“云端”版本支持。

5.1 设置 OTA 属性

```
iot_s32 iot_ota_option_set ( iot_u32 chunk_size )
```

| 参数 | 说明 |
|------------|--|
| chunk_size | APP 端会将固件按 chunk_size 分块，分块后一次下发一块。 2 的 n 次幂，范围 256-2048，默认 512 |
| 返回值 | 0：设置成功 -1：设置失败，参数错误 |

5.2 接收固件数据块

```
iot_s32 iot_ota_chunk_cb ( iot_u8 chunk_is_last,
                           iot_u32 chunk_offset,
                           iot_u32 chunk_size,
                           const iot_s8* chunk )
```

| 参数 | 说明 |
|---------------|-------------------------------------|
| chunk_is_last | 0: 不是最后一个分块 1: 是最后一个分块，固件传输结束 |
| chunk_offset | 本数据块相对于完整固件的偏移量 即，第一个数据块，此参数值是 0 |

| | |
|------------|----------------------------|
| chunk_size | 本数据块的长度 |
| chunk | 固件数据块 |
| 返回值 | 0 : 写入固件块成功 -1: 写入固件块失败 |

5.3 接收升级指令

收到此命令，重启，运行新版本固件。回调函数中不可执行太多耗时代码。

```
void iot_ota_upgrade_cb ( OTA_FINISH_STATUS_T finish_status )
```

| 参数 | 说明 |
|---------------|--|
| finish_status | OTA_FINISH_OK : OTA 传输正常结束 OTA_FINISH_ERR: OTA 传输异常结束 |
| 返回值 | 无 |

6 高级功能

6.1 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 |

6.2 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
- ◆ 设备在线，云端收到 app 数据后，直接透传至设备
- ◆ 设备离线，云端最多会保存 20 条 app 数据，待设备上线时发送，发送后清空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 |
|----------|---------|

6.3 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 |

6.4 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 |

6.5 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 |
|--|--------|