涂鸦 ble sdk 说明

##

概述

Tuya BLE SDK主要封装了和涂鸦智能手机 App 之间的通信协议以及实现了简易事件调度机制，使用该 SDK 的设备无须关心具体的通信协议实现细节，通过调用 SDK 提供的 API 和 Call Back 即可与涂鸦智能 App 互联互通。

SDK下载地址

SDK架构

系统架构

如 下图 所示，系统架构包括几个主要的部分：

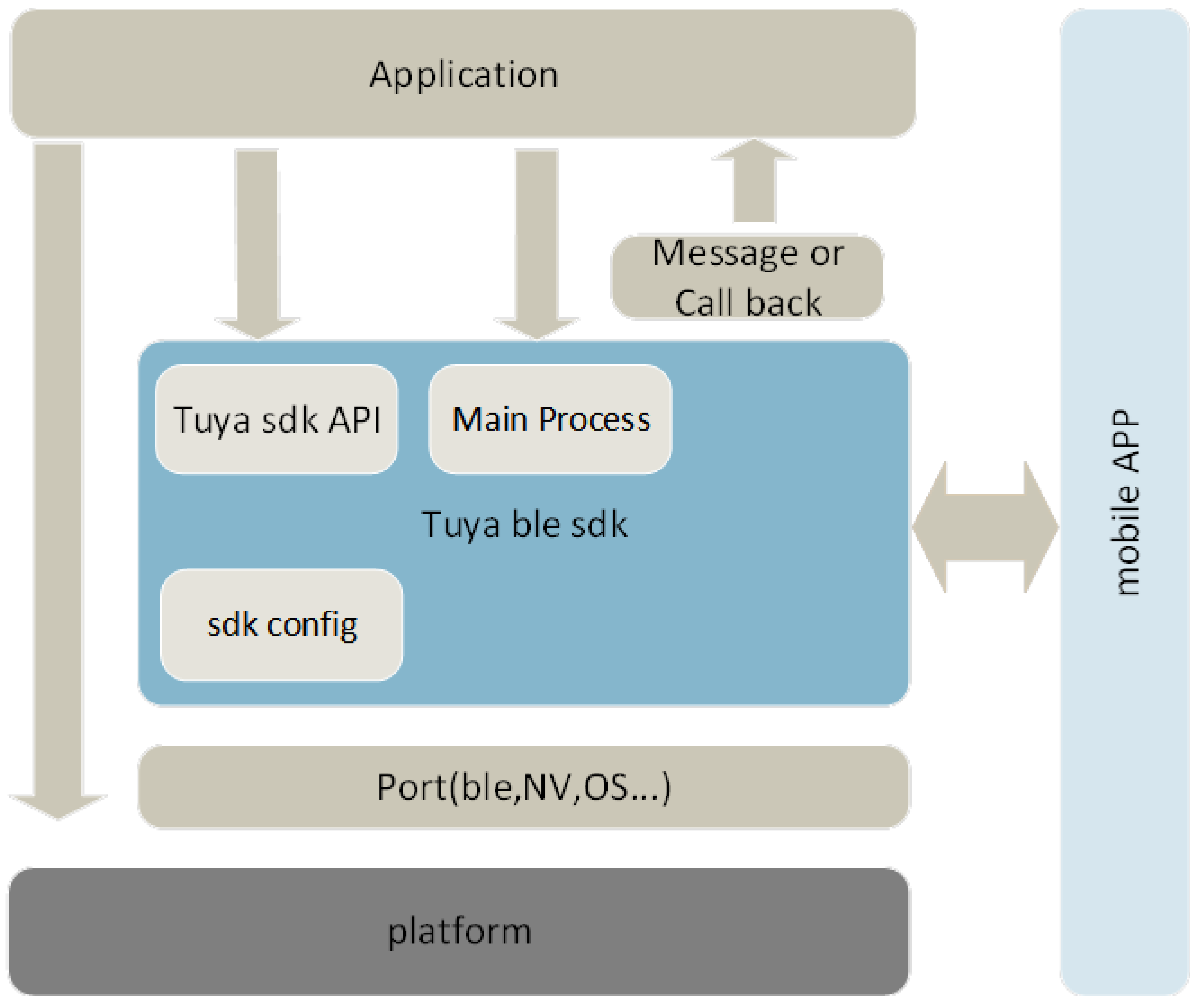
- Platform：所使用的芯片平台，芯片 + 协议栈 由芯片公司维护。
- Port：Tuya BLE SDK 所需要的接口抽象，需要用户根据具体的芯片平台移植实现。
- Tuya BLE SDK ： SDK 封装了涂鸦 BLE 通信协议，提供构建涂鸦 BLE 应用所需的服务接口。
- Application：基于Tuya BLE SDK 构建的应用。
- Tuya BLE SDK API：

  SDK 提供相关 API 用于设备实现 BLE 相关的管理、通信等，如果使用 OS，API 的调用将采用基于消息的异步机制，API 的执行结果将会以 Message 或者 Call Back 的方式通知给设备的 Application，如果是非 OS，API 的返回值即为执行结果。
- SDK config：

  SDK 可裁剪可配置，通过 config 文件中的宏定义操作，例如配置 SDK 适用于多协议设备的通用配网模式，蓝牙单点设备、基于 ECDH 秘钥协商加密模式、是否使用 OS 等。
- Main process function：

  为 SDK 的主引擎，设备 Application 需要一直调用，如果 Platform 基于 OS，SDK 会基于 Port 层提供的 OS相关 API 自动创建一个任务用于执行 Main process function，如果是非 OS 平台，需要设备 Application循环调用。
- Message or Call Back：

  SDK 通过 Message 或者设备 App 注册的Call Back函数向设备 App 发送数据（状态、数据等）。



OS支持

Tuya BLE SDK 可运行在基于 RTOS 的芯片平台下。如果使用 OS，API 的调用将采用基于消息的异步机制，初始化 SDK 时，SDK 将会根据 tuya_ble_config.h 文件的相关配置自动创建一个任务用于处理 SDK 的核心逻辑，同时自动创建一个消息队列用于接收 API 的执行请求，API 的执行结果也将会以 Message 的方式通知给设备的 Application，所以用户 Application 需要创建一个消息队列并在调用 tuya_ble_sdk_init() 后调用tuya_ble_callback_queue_register() 将消息队列注册至 SDK 中。

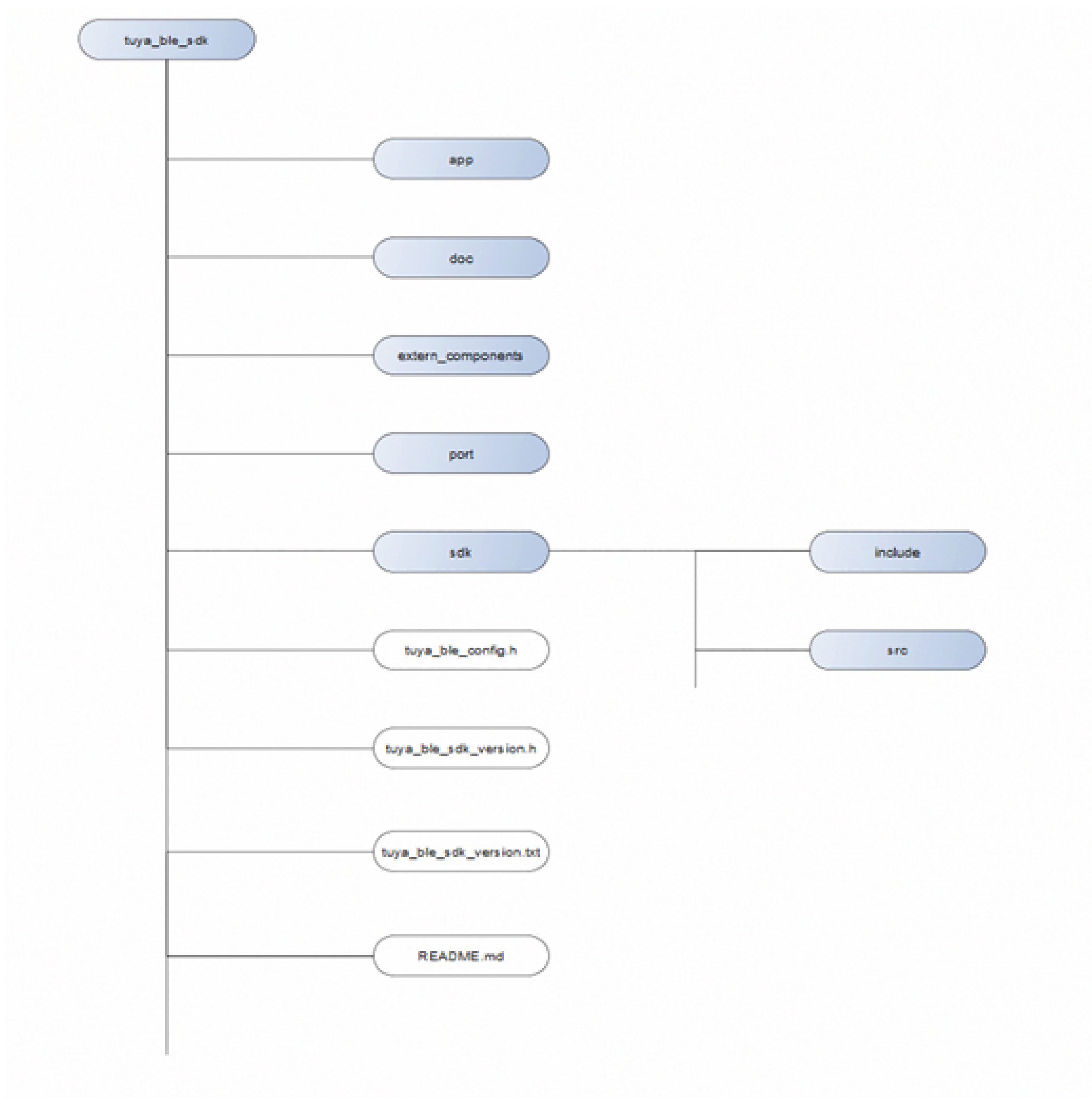
事件队列

先进先出，用于缓存设备 Application 以及 Platform 层发送来消息事件（API 调用、BLE 底层数据接收等），Main process function 模块循环查询消息队列并取出处理。

SDK目录

```
    |- app

    |- doc

    |- extern_components
```

```
|- port
|- sdk
|       |- include
|       |- lib
|       |- src
|- tuya_ble_config.h
|- tuya_ble_sdk_version.h
|- tuya_ble_sdk_version.txt
|- README.md
```



Directory	Description
app	存放sdk之上的一些应用，例如产测、通用对接等。
doc	说明文档
extern_components	一些外部组件，例如安全相关算法实现
port	各平台移植代码
sdk	Tuya BLE SDK 核心代码
tuya_ble_config.h	BLE SDK 配置文件
tuya_ble_sdk_version.h	SDK 版本 .h 文件
README.md	说明文件

TUYA BLE SERVICE

概述

Tuya BLE SDK 不提供初始化 Service 相关接口，Application 需要在初始化 SDK 前实现 SERVICE 章节所定义的 Sevice Characteristics，当然，Application除了定义Tuya BLE SDK 所需的 Service 外，也可以定义其他 Service。蓝牙广播数据的初始话内容见广播内容章节，否则 SDK 将不能正常工作。

SERVICE

Tuya BLE SDK 使用的 Service UUID 和 Characteristic UUID 如下表所示：

Service UUID	Characteristic UUID	Properties	Security Permissions	Maximum length of the characteristic value
FD50	00000001-0000-1001-8001-00805F9B07D0	Write without response	None	ATT MTU - 3
	00000002-0000-1001-8001-00805F9B07D0	Notify	None	ATT MTU - 3
	00000003-0000-1001-8001-00805F9B07D0	Read	None	512

Read Characteristic uuid 只有在使能 LINK 层加密时才需要（TUYA_BLE_LINK_LAYER_ENCRYPTION_SUPPORT_ENABLE 配置为 1 时），否则不需要定义。

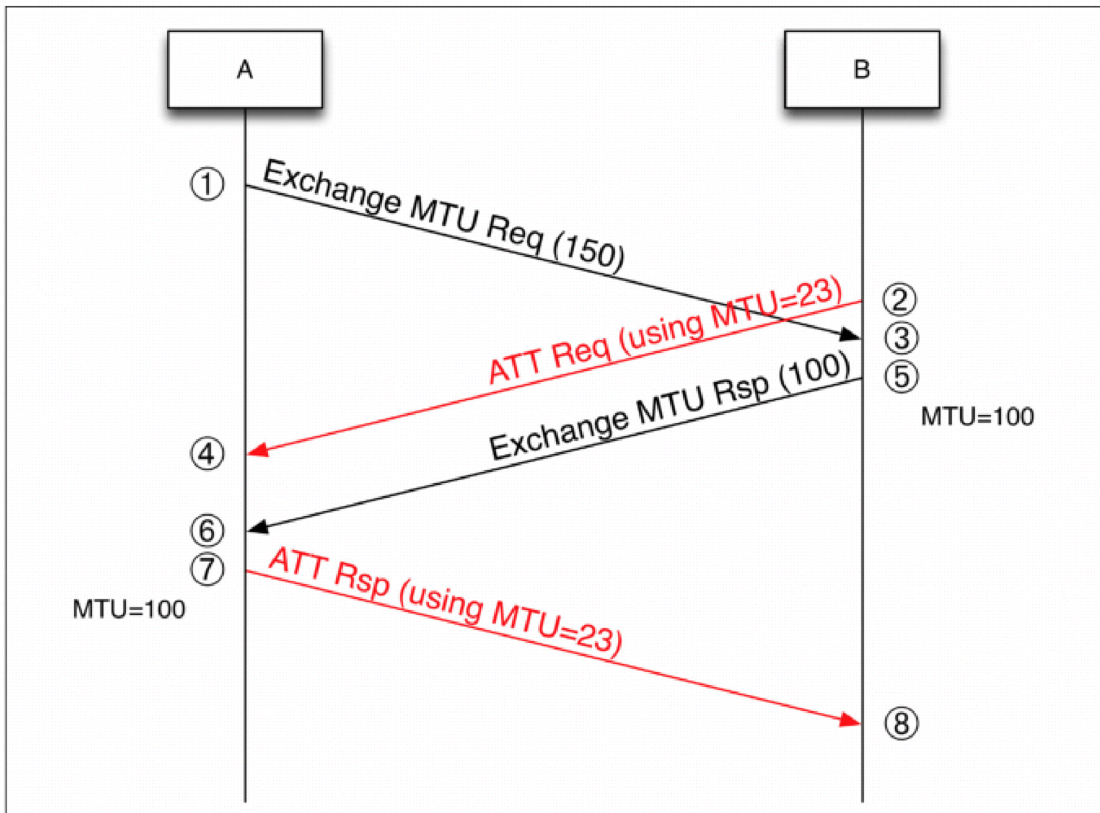
MTU

MTU: 最大传输单元（MAXIMUM TRANSMISSION UNIT），指在一个 PDU （Protocol Data Unit：协议数据单元）能够传输的最大数据量。

蓝牙4.0定义了 ATT 的默认 MTU 为 23 个 bytes，除去 ATT 的 opcode 一个字节以及 ATT 的 handle 2 个字节之后，剩下的 20 个字节便是留给 GATT 的，这就是我们单包数据最大长度为 20 字节的原因。

从蓝牙4.2开始支持 MTU 交换，MTU 交换是为了在主从双方设置一个 PDU 中最大能够交换的数据量，通过 MTU的交换和双方确认，主从双方约定每次在做数据传输时不超过这个最大数据单元。

MTU交换通常发生在主从双方建立连接关系后，如下图所示：



通过 MTU 交换，主从双方选择一个较低的值作为本次连接 MTU。

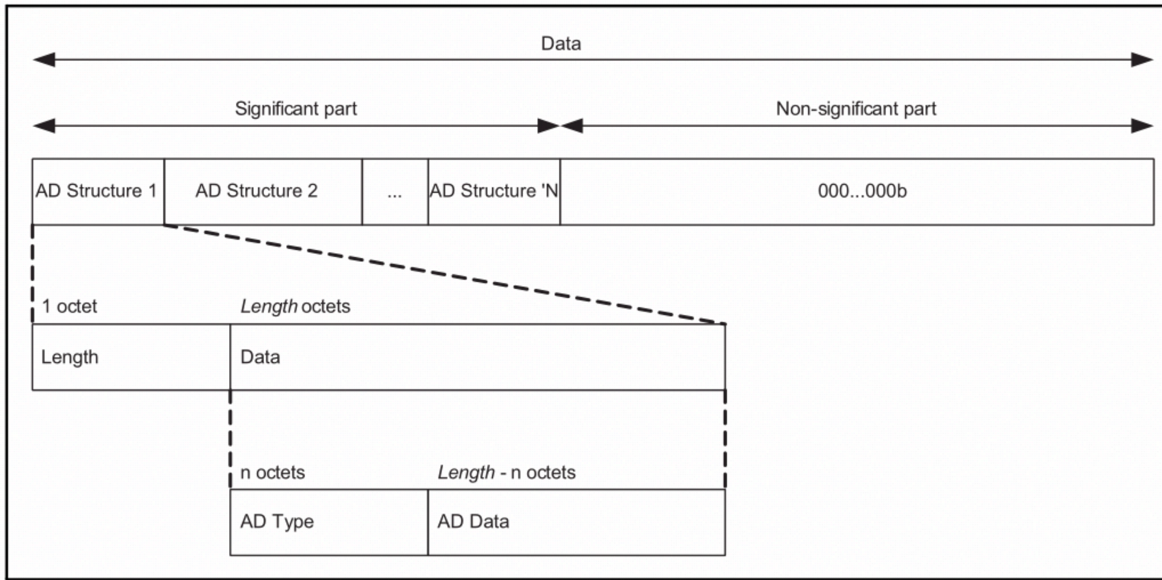
手机端蓝牙MTU交换在蓝牙建立连接后会自动进行，无需上层应用干预，但是上层应用需要拿到交换的结果ATT MTU 值，然后通过获取设备信息指令将（ATT MTU - 3）发送给设备，之后的通信就采用该值作为每次写操作的最大值。

设备端定义的 ATT MTU 当前不能超过 247 。

广播数据

BLE广播包数据结构

BLE 广播包数据结构如下图所示：



Tuya BLE SDK 在初始化完成后会自动更新蓝牙广播内容，但是设备应用程序在初始化蓝牙广播时需要将广播内容初始化为如下所示的数据。

Tuya ble adv data

广播数据段描述	类型	说明
Flags	0x01	长度：0x02 类型：0x01 数据：0x06
Service UUID	0x02	长度：0x03 类型：0x02 数据： UUID：0xFD50
Service Data	0x16	长度：0x17 类型：0x16 数据： UUID: 0xFD50 DATA：20 个字节的 0x00

Tuya ble scan response data

广播数据段描述	类型	说明
Manufacturer data	0xFF	长度：0x17 类型：0xFF 数据： COMPANY ID:0x07D0 DATA：20 个字节的 0x00
Custom Complete Local Name	0x09	长度：Length（最大6个字节） 类型：0x09 数据： Name: (Length - 1)个字节长度的自定义名字

PORT和CONFIG介绍

Port说明

如下图所示，tuya_ble_port.h 和 tuya_ble_port_peripheral.h 里定义的所有接口都需要用户根据具体的芯片平台移植实现，如果用户平台是非 OS 的，OS 相关接口不需要实现。tuya_ble_port.c 和 tuya_ble_port_peripheral.c 是对 tuya_ble_port.h 和 tuya_ble_port_peripheral.h 所定义接口的弱实现，用户不能在该文件里实现具体的平台接口，应该新建一个 .c文件，例如新建一个 tuya_ble_port_nrf52832.c 文件。

名称	修改日期	类型	大小
bk	2019/9/10 10:30	文件夹	
cypress	2019/9/10 10:30	文件夹	
nordic	2019/9/14 16:23	文件夹	
realtek	2019/9/14 16:23	文件夹	
telink	2019/9/10 21:57	文件夹	
tuya_ble_port.c	2019/10/3 16:16	C Source File	12 KB
tuya_ble_port.h	2019/10/18 16:54	C/C++ Header F...	19 KB
tuya_ble_port_peripheral.c	2019/9/16 11:05	C Source File	1 KB
tuya_ble_port_peripheral.h	2019/9/16 11:08	C/C++ Header F...	1 KB

Port 接口介绍

TUYA_BLE_PRINTF

函数名	TUYA_BLE_PRINTF
函数原型	void TUYA_BLE_PRINTF(const char *format,...)
功能概述	格式化输出
参数	format[in] : 格式控制符; ...[in]： 可变参数；
返回值	无
备注	通过宏定义的方式移植实现，例如 `#define TUYA_BLE_PRINTF(...) log_d(__VA_ARGS__)`

TUYA_BLE_HEXDUMP

函数名	TUYA_BLE_HEXDUMP
函数原型	void TUYA_BLE_HEXDUMP(uint8_t *p_data , uint16_t len)
功能概述	16进制打印
参数	p_data[in]: 要打印的数据指针; len[in]： 数据长度；
返回值	无
备注	通过宏定义的方式移植实现，例如 `#define TUYA_BLE_HEXDUMP(...) elog_hexdump("", 8, __VA_ARGS__)`

tuya_ble_gap_advertising_adv_data_update

函数名	tuya_ble_gap_advertising_adv_data_update
函数原型	tuya_ble_status_t tuya_ble_gap_advertising_adv_data_update(uint8_t const * p_ad_data, uint8_t ad_len)
功能概述	BLE 广播包数据更新
参数	p_ad_data[in]：新的广播数据; ad_len[in]：数据长度；
返回值	TUYA_BLE_SUCCESS：成功; 其他：失败。
备注	

tuya_ble_gap_advertising_scan_rsp_data_update

函数名	tuya_ble_gap_advertising_scan_rsp_data_update
函数原型	tuya_ble_status_t tuya_ble_gap_advertising_scan_rsp_data_update(uint8_t const *p_sr_data, uint8_t sr_len)
功能概述	BLE 扫描响应包数据更新
参数	p_sr_data[in]：新的扫描响应包数据； sr_len[in]：数据长度；
返回值	TUYA_BLE_SUCCESS：成功； 其他：失败。
备注	

tuya_ble_gap_disconnect

函数名	tuya_ble_gap_disconnect
函数原型	tuya_ble_status_t tuya_ble_gap_disconnect(void)
功能概述	断开 BLE 连接
参数	无
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	

tuya_ble_gatt_send_data

函数名	tuya_ble_gatt_send_data
函数原型	tuya_ble_status_t tuya_ble_gatt_send_data(const uint8_t *p_data,uint16_t len)
功能概述	ble gatt 发送数据（notify）
参数	p_data[in] : 要发送的数据指针； len[in] : 发送的数据长度（目前不能超过20字节）。
返回值	TUYA_BLE_SUCCESS: 成功；其他：失败。
备注	1、必须是 notify 方式发送； 2、如果是返回成功，必须是真正的发出去了，因为 Tuya BLE SDK 会基于该返回值判断是否需要缓存和重发。

tuya_ble_device_info_characteristic_value_update

函数名	tuya_ble_device_info_characteristic_value_update
函数原型	tuya_ble_status_t tuya_ble_device_info_characteristic_value_update(uint8_t const *p_data, uint8_t data_len)
功能概述	更新 Read Characteristic Value (UUID : 00000003-0000-1001-8001-00805F9B07D0)
参数	p_data[in] : 要发送的数据指针； len[in] : 发送的数据长度（目前不能超过20字节）。
返回值	TUYA_BLE_SUCCESS: 成功；其他：失败。
备注	1、只有在 `TUYA_BLE_LINK_LAYER_ENCRYPTION_SUPPORT_ENABLE` 定义为 1 时 才需要实现该函数。 2、主要应用于 需要 开启蓝牙 LINK 层配对绑定机制的场景。

tuya_ble_timer_create

函数名	tuya_ble_timer_create
函数原型	tuya_ble_status_t tuya_ble_timer_create(void** p_timer_id,uint32_t timeout_value_ms, tuya_ble_timer_mode mode, tuya_ble_timer_handler_t timeout_handler)
功能概述	创建一个定时器
参数	p_timer_id[out] :创建的定时器指针； timeout_value_ms[in]:定时时间，单位 ms； mode[in] : TUYA_BLE_TIMER_SINGLE_SHOT - 单一模式，TUYA_BLE_TIMER_REPEATED-重复模式。 timeout_handler : 定时器回调函数。
返回值	TUYA_BLE_SUCCESS: 成功；其他：失败。
备注	

tuya_ble_timer_delete

函数名	tuya_ble_timer_delete
函数原型	tuya_ble_status_t tuya_ble_timer_delete(void* timer_id)
功能概述	删除一个定时器
参数	timer_id [in] : 定时器id;
返回值	TUYA_BLE_SUCCESS: 成功; TUYA_BLE_ERR_INVALID_PARAM: 无效的参数; 其他：失败。
备注	

tuya_ble_timer_start

函数名	tuya_ble_timer_start
函数原型	tuya_ble_status_t tuya_ble_timer_start(void* timer_id)
功能概述	启动一个定时器
参数	timer_id [in] :定时器id;
返回值	TUYA_BLE_SUCCESS: 成功; TUYA_BLE_ERR_INVALID_PARAM: 无效的参数; 其他：失败。
备注	如果定时器已经启动，执行该函数后会重新启动。

tuya_ble_timer_restart

函数名	tuya_ble_timer_restart
函数原型	tuya_ble_status_t tuya_ble_timer_restart(void* timer_id,uint32_t timeout_value_ms)
功能概述	以新的定时时间重新启动一个定时器
参数	timer_id [in] :定时器id; timeout_value_ms[in]：定时时间，单位ms。
返回值	TUYA_BLE_SUCCESS: 成功; TUYA_BLE_ERR_INVALID_PARAM: 无效的参数; 其他：失败。
备注	

tuya_ble_timer_stop

函数名	tuya_ble_timer_stop
函数原型	tuya_ble_status_ttuya_ble_timer_stop(void* timer_id)
功能概述	停止一个定时器
参数	timer_id [in] :定时器id；
返回值	TUYA_BLE_SUCCESS: 成功； TUYA_BLE_ERR_INVALID_PARAM: 无效的参数； 其他：失败。
备注	

tuya_ble_device_delay_ms

函数名	tuya_ble_device_delay_ms
函数原型	void tuya_ble_device_delay_ms(uint32_t ms)
功能概述	ms 级延时函数
参数	ms [in] : 延时毫秒数；
返回值	无
备注	如果是在 os 平台下，必须为无阻塞延时。

tuya_ble_device_delay_us

函数名	tuya_ble_device_delay_us
函数原型	void tuya_ble_device_delay_us(uint32_t us)
功能概述	us 级延时函数
参数	us [in] : 延时微妙数；
返回值	无
备注	如果是在 os 平台下，必须为无阻塞延时。

tuya_ble_device_reset

函数名	**tuya_ble_device_reset**
函数原型	tuya_ble_status_t tuya_ble_device_reset(void)
功能概述	设备重启
参数	无
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	

tuya_ble_gap_addr_get

函数名	**tuya_ble_gap_addr_get**
函数原型	tuya_ble_status_t tuya_ble_gap_addr_get(tuya_ble_gap_addr_t *p_addr);
功能概述	获取设备 mac 地址
参数	p_addr [out] : mac 地址指针。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	

```
typedef enum

{

TUYA_BLE_ADDRESS_TYPE_PUBLIC, // public address

TUYA_BLE_ADDRESS_TYPE_RANDOM, // random address

} tuya_ble_addr_type_t;

typedef struct

{

tuya_ble_addr_type_t addr_type;

uint8_t addr[6];

}tuya_ble_gap_addr_t;
```

tuya_ble_gap_addr_set

函数名	tuya_ble_gap_addr_set
函数原型	tuya_ble_status_t tuya_ble_gap_addr_set(tuya_ble_gap_addr_t *p_addr);
功能概述	设置更新设备 mac 地址
参数	p_addr [in] : mac 地址指针。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	

tuya_ble_device_enter_critical

函数名	tuya_ble_device_enter_critical
函数原型	void tuya_ble_device_enter_critical(void)
功能概述	进入临界区
参数	无
返回值	无
备注	通过宏定义方式定义，例如在 nordic 平台下定义如下所示。

```
#define tuya_ble_device_enter_critical() \
{                                      \
    uint8_t __CR_NESTED = 0;           \
    app_util_critical_region_enter(&__CR_NESTED);

#define tuya_ble_device_exit_critical()  \
    app_util_critical_region_exit(__CR_NESTED); \
}
```

tuya_ble_device_exit_critical

函数名	tuya_ble_device_exit_critical
函数原型	void tuya_ble_device_exit_critical(void)
功能概述	退出临界区
参数	无
返回值	无
备注	通过宏定义方式定义，例如在 nordic 平台下定义如下所示。

```
#define tuya_ble_device_enter_critical() \
{                                     \
    uint8_t __CR_NESTED = 0;          \
    app_util_critical_region_enter(&__CR_NESTED);

#define tuya_ble_device_exit_critical()  \
    app_util_critical_region_exit(__CR_NESTED); \
}
```

tuya_ble_rand_generator

函数名	tuya_ble_rand_generator
函数原型	tuya_ble_status_t tuya_ble_rand_generator(uint8_t* p_buf, uint8_t len)
功能概述	随机数生成
参数	p_buf [out]: 生成的随机数数组指针； len[in] : 随机数字节数。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	

tuya_ble_rtc_get_timestamp

函数名	tuya_ble_rtc_get_timestamp
函数原型	tuya_ble_status_t tuya_ble_rtc_get_timestamp(uint32_t *timestamp,int32_t *timezone);
功能概述	获取 unix 时间戳
参数	timestamp [out]: 时间戳； timezone [out] : 时区（有符号型整数，真实时区的100倍）。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	数据来自应用程序维护的 RTC 实时时钟。

tuya_ble_rtc_set_timestamp

函数名	tuya_ble_rtc_set_timestamp
函数原型	tuya_ble_status_ttuya_ble_rtc_set_timestamp(uint32_t timestamp,int32_t timezone)
功能概述	更新 unix 时间戳
参数	timestamp [in]:unix 时间戳； timezone [in] : 时区（有符号型整数，真实时区的100倍）。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	SDK 通过调用此函数更新应用程序维护的 RTC 实时时钟。

tuya_ble_nv_init

函数名	tuya_ble_nv_init
函数原型	tuya_ble_status_ttuya_ble_nv_init(void)
功能概述	NV 初始化。
参数	无
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	配合 config 文件定义的 NV 空间地址使用，SDK 调用 nv 相关函数来存储和管理授权信息和其他信息。

tuya_ble_nv_erase

函数名	tuya_ble_nv_erase
函数原型	tuya_ble_status_ttuya_ble_nv_erase(uint32_t addr,uint32_t size)
功能概述	NV 擦除函数。
参数	addr[in] : 要擦除 NV 区域的起始地址； size[in] : 要擦除的大小（单位：字节）。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	配合 config 文件定义的 NV 空间地址使用，SDK 调用 nv 相关函数来存储和管理授权信息和其他信息。

tuya_ble_nv_write

函数名	tuya_ble_nv_write
函数原型	tuya_ble_status_ttuya_ble_nv_write(uint32_t addr,const uint8_t * p_data, uint32_t size)
功能概述	NV写数据函数。
参数	addr[in] : 要写入数据的起始地址； p_data[in] : 要写入数据的起始地址。 Size[in] : 要写入数据的大小（单位：字节）。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	配合 config 文件定义的 NV 空间地址使用，SDK 调用 nv 相关函数来存储和管理授权信息和其他信息。

tuya_ble_nv_read

函数名	tuya_ble_nv_read
函数原型	tuya_ble_status_t tuya_ble_nv_read(uint32_t addr,uint8_t * p_data, uint32_t size)
功能概述	NV读数据函数。
参数	addr[in] : 要读取数据的NV起始地址； p_data[out] : 读取数据的地址； Size[in] : 要读取数据的大小（单位：字节）。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	配合 config 文件定义的 NV 空间地址使用，SDK 调用 nv 相关函数来存储和管理授权信息和其他信息。

tuya_ble_common_uart_init

函数名	tuya_ble_common_uart_init
函数原型	tuya_ble_status_t tuya_ble_common_uart_init(void)
功能概述	Uart初始化函数。
参数	无
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	1、该 UART 主要用于产测授权使用。 2、如果应用代码在初始化 BLE SDK 之前已经初始化过uart，则不用移植实现该函数。

tuya_ble_common_uart_send_data

函数名	tuya_ble_common_uart_send_data
函数原型	tuya_ble_status_t tuya_ble_common_uart_send_data(const uint8_t *p_data,uint16_t len)
功能概述	Uart 发送数据函数。
参数	p_data[in] : 要发送的数据指针； len[in] : 要发送的数据长度。
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	主要用于授权产测使用。

tuya_ble_os_task_create

函数名	tuya_ble_os_task_create
函数原型	bool tuya_ble_os_task_create(void **pp_handle, const char *p_name, void(*p_routine)(void *),void *p_param, uint16_t stack_size, uint16_t priority)
功能概述	Task create。
参数	pp_handle [out] :Used to pass back a handle by which the created task can be referenced； p_name[in] : Adescriptive name for the task； p_routine [in] : Pointerto task routine function that must be implemented to never return； p_param[in] :Pointer parameter passed to the task routine function; stack_size[in] :The size of the task stack that is specified as the number of bytes； priority[in] : Thepriority at which the task should run. Higher priority task has higherpriority value
返回值	True : Task wascreated successfully and added to task ready list. False：Task was failed to create.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_task_delete

函数名	tuya_ble_os_task_delete
函数原型	bool tuya_ble_os_task_delete(void *p_handle)
功能概述	Remove a task fromRTOS's task management. The task being deleted will be removed from RUNNING,READY or WAITING state
参数	pp_handle [in] : Thehandle of the task to be deleted.
返回值	True : Task wasdeleted successfully False：Task was failed to delete.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_task_suspend

函数名	tuya_ble_os_task_suspend
函数原型	bool tuya_ble_os_task_suspend(void *p_handle)
功能概述	Suspend the task.The suspended task will not be scheduled and never get any microcontrollerprocessing time.
参数	pp_handle [in] : Thehandle of the task to be suspend.
返回值	True : Task wassuspend successfully. False：Task was failed to suspend.
备注	该接口函数只需在基于os的平台下实现。

tuya_ble_os_task_resume

函数名	tuya_ble_os_task_resume
函数原型	bool tuya_ble_os_task_resume(void *p_handle)
功能概述	Resume thesuspended task.
参数	pp_handle [in] : Thehandle of the task to be resumed.
返回值	True : Task wasresumed successfully. False：Task was failed to resume.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_msg_queue_create

函数名	tuya_ble_os_msg_queue_create
函数原型	bool tuya_ble_os_msg_queue_create(void **pp_handle, uint32_t msg_num, uint32_tmsg_size)
功能概述	Creates a messagequeue instance. This allocates the storage required by the new queue andpasses back a handle for the queue.
参数	pp_handle [out] :Used to pass back a handle by which the message queue can be referenced. msg_num [in] : Themaximum number of items that the queue can contain. msg_size [in] : Thenumber of bytes each item in the queue will require.
返回值	True : Messagequeue was created successfully. False：Message queue was failed to create.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_msg_queue_delete

函数名	tuya_ble_os_msg_queue_delete
函数原型	bool tuya_ble_os_msg_queue_delete(void *p_handle)
功能概述	Creates a messagequeue instance. This allocates the storage required by the new queue andpasses back a handle for the queue.
参数	pp_handle [in] : Thehandle to the message queue being deleted.
返回值	True : Messagequeue was deleted successfully. False：Message queue was failed to delete.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_msg_queue_peek

函数名	tuya_ble_os_msg_queue_peek
函数原型	bool tuya_ble_os_msg_queue_peek(void *p_handle, uint32_t *p_msg_num)
功能概述	Peek the number ofitems sent and resided on the message queue.
参数	pp_handle [in] : Thehandle to the message queue being peeked. p_msg_num[out] : Usedto pass back the number of items residing on the message queue.
返回值	True : Messagequeue was peeked successfully. False：Message queue was failed to peek.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_msg_queue_send

函数名	tuya_ble_os_msg_queue_send
函数原型	bool tuya_ble_os_msg_queue_send(void *p_handle, void *p_msg, uint32_t wait_ms)
功能概述	Send an item to theback of the specified message queue.
参数	pp_handle [in] : Thehandle to the message queue on which the item is to be sent. p_msg[in] : Pointerto the item that is to be sent on the queue. wait_ms[in] : Themaximum amount of time in milliseconds that the task should block waiting forthe item to sent on the queue. * 0 No blocking and returnimmediately. * 0xFFFFFFFF Block infinitely until the item sent. * others The timeout value in milliseconds.
返回值	True : Message itemwas sent successfully. False：Message item was failed to send.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_os_msg_queue_recv

函数名	tuya_ble_os_msg_queue_recv
函数原型	bool tuya_ble_os_msg_queue_recv(void *p_handle, void *p_msg, uint32_t wait_ms)
功能概述	Receive an itemfrom the specified message queue.
参数	pp_handle [in] : Thehandle to the message queue from which the item is to be received. p_msg[out] : Pointerto the buffer into which the received item will be copied. wait_ms[in] : Themaximum amount of time in milliseconds that the task should block waiting forthe item to received on the queue. * 0 No blocking and return immediately. * 0xFFFFFFFF Block infinitely until the item received. * others The timeout value in milliseconds.
返回值	True : Message itemwas received successfully. False：Message item was failed to receive.
备注	该接口函数只需在基于 os 的平台下实现。

tuya_ble_event_queue_send_port

函数名	tuya_ble_event_queue_send_port
函数原型	bool tuya_ble_event_queue_send_port(tuya_ble_evt_param_t *evt, uint32_t wait_ms)
功能概述	If undefineTUYA_BLE_SELF_BUILT_TASK ,Application should provide the task to SDKtoprocess the event. SDK will use this port to send event to the task ofprovided by Application.
参数	evt [in] : themessage data point to be send. wait_ms[in] : Themaximum amount of time in milliseconds that the task should block waiting forthe item to sent on the queue. * 0 No blocking and returnimmediately. * 0xFFFFFFFF Block infinitely until the item sent. * others The timeout value in milliseconds.
返回值	True : Message itemwas sent successfully. False： Message item was failed to send.
备注	该接口函数只需在基于os的平台下实现。

tuya_ble_aes128_ecb_encrypt

函数名	tuya_ble_aes128_ecb_encrypt
函数原型	bool tuya_ble_aes128_ecb_encrypt(uint8_t *key,uint8_t *input,uint16_tinput_len,uint8_t *output)
功能概述	128 bit AES ECBencryption on speicified plaintext and keys
参数	key [in] : keys toencrypt the plaintext In_put[in] : specifedplain text to be encypted. in_put_len[in] : bytelength of the data to be encrypted, must be multiples of 16. Out_put[out] : outputbuffer to store encrypted data.
返回值	True : successful. False： fail.
备注	

tuya_ble_aes128_ecb_decrypt

函数名	tuya_ble_aes128_ecb_decrypt
函数原型	bool tuya_ble_aes128_ecb_decrypt(uint8_t *key,uint8_t *input,uint16_tinput_len,uint8_t *output)
功能概述	128 bit AES ECBdecryption on speicified encrypted data and keys
参数	key [in] : keys to decryptthe plaintext In_put[in] : specifedencypted data to be decypted in_put_len[in] : bytelength of the data to be descrypted, must be multiples of 16. Out_put[out] : outputbuffer to store decrypted data.
返回值	True : successful. False：fail.
备注	

tuya_ble_aes128_cbc_encrypt

函数名	tuya_ble_aes128_cbc_encrypt
函数原型	bool tuya_ble_aes128_cbc_encrypt(uint8_t *key,uint8_t *iv,uint8_t input,uint16_t input_len,uint8_toutput)
功能概述	128 bit AES CBCencryption on speicified plaintext and keys.
参数	key [in] : keys toencrypt the plaintext. iv[in] : initializationvector (IV) for CBC mode. In_put[in] : specifedplain text to be encypted. in_put_len[in] : bytelength of the data to be encrypted, must be multiples of 16. Out_put[out] : outputbuffer to store encrypted data.
返回值	True : successful. False：fail.
备注	

tuya_ble_aes128_cbc_decrypt

函数名	tuya_ble_aes128_cbc_decrypt
函数原型	bool tuya_ble_aes128_cbc_decrypt(uint8_t*key,uint8_t *iv,uint8_t *input,uint16_t input_len,uint8_t *output)
功能概述	128 bit AES CBC descryption on speicified plaintext and keys.
参数	key [in] : keys to decryptthe plaintext. Iv[in] : initializationvector (IV) for CBC mode In_put[in] : specifedencypted data to be decrypted. in_put_len[in] : bytelength of the data to be descrypted, must be multiples of 16. Out_put[out] : outputbuffer to store decrypted data.
返回值	True : successful. False： fail.
备注	

tuya_ble_md5_crypt

函数名	tuya_ble_md5_crypt
函数原型	booltuya_ble_md5_crypt(uint8_t *input,uint16_t input_len,uint8_t *output)
功能概述	MD5 checksum.
参数	In_put[in] : specifedplain text to be encypted. in_put_len[in] : ytelength of the data to be encypted. Out_put[out] : outputbuffer to store md5 result data,output data len is always 16.
返回值	True : successful. False： fail.
备注	

tuya_ble_hmac_sha1_crypt

函数名	**tuya_ble_hmac_sha1_crypt**
函数原型	bool tuya_ble_hmac_sha1_crypt(constuint8_t *key, uint32_t key_len, const uint8_t *input, uint32_t input_len,uint8_t *output)
功能概述	calculates the fullgeneric HMAC on the input buffer with the provided key.
参数	key [in] : The HMACsecret key. key_len[in] : Thelength of the HMAC secret key in Bytes. In_put[in] : specifedplain text to be encypted. in_put_len[in] : bytelength of the data to be encypted. Out_put[out] : outputbuffer to store the result data.
返回值	True : successful. False： fail.
备注	

tuya_ble_hmac_sha256_crypt

函数名	**tuya_ble_hmac_sha1_crypt**
函数原型	bool tuya_ble_hmac_sha256_crypt(const uint8_t *key, uint32_t key_len, const uint8_t *input, uint32_t input_len, uint8_t *output)
功能概述	calculates the fullgeneric HMAC on the input buffer with the provided key.
参数	key [in] : The HMACsecret key. key_len[in] : Thelength of the HMAC secret key in Bytes. In_put[in] : specifedplain text to be encypted. in_put_len[in] : bytelength of the data to be encypted. Out_put[out] : outputbuffer to store the result data.
返回值	True : successful. False： fail.
备注	

tuya_ble_storage_private_data

函数名	tuya_ble_storage_private_data
函数原型	tuya_ble_status_t tuya_ble_storage_private_data(tuya_ble_private_data_type private_data_type, uint8_t *p_data, uint32_t data_size)
功能概述	存储密钥数据，主要用于 基于 安全芯片加密的高级加密模式
参数	private_data_type[in] : 数据类型 p_data[in] : 要写入数据的起始地址 data_size[in] : 数据长度
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_get_dev_crt_len

函数名	tuya_ble_get_dev_crt_len
函数原型	uint32_t tuya_ble_get_dev_crt_len(void)
功能概述	获取存储在安全芯片内的证书长度
参数	无
返回值	证书长度
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_get_dev_crt_der

函数名	tuya_ble_get_dev_crt_der
函数原型	tuya_ble_status_t tuya_ble_get_dev_crt_der(uint8_t *p_der, uint32_t der_len)
功能概述	读取安全芯片内的设备证书数据
参数	p_der[out] : 读取的证书数据存放地址 der_len[in] : 要读取的证书数据长度
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_ecc_keypair_gen_secp256r1

函数名	tuya_ble_ecc_keypair_gen_secp256r1
函数原型	tuya_ble_status_t tuya_ble_ecc_keypair_gen_secp256r1(uint8_t *public_key,uint8_t *private_key)
功能概述	生成ECDH密钥对，ECC曲线：secp256r1
参数	public_key[out] : 生成的公钥 private_key[out] : 生成的私钥
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_ecc_shared_secret_compute_secp256r1

函数名	tuya_ble_ecc_shared_secret_compute_secp256r1
函数原型	tuya_ble_status_t tuya_ble_ecc_shared_secret_compute_secp256r1(uint8_t *public_key,uint8_t *private_key,uint8_t *secret_key)
功能概述	计算共享密钥
参数	public_key[in] : 公钥 private_key[in] : 私钥 secret_key[out]：共享密钥
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_ecc_sign_secp256r1

函数名	tuya_ble_ecc_sign_secp256r1
函数原型	tuya_ble_status_t tuya_ble_ecc_sign_secp256r1(const uint8_t *p_sk, const uint8_t * p_data, uint32_t data_size,uint8_t *p_sig)
功能概述	计算ECDSA签名
参数	p_sk[in] : 签名私钥 p_data[in] : 要签名的数据 data_size[in]：要签名的数据长度 p_sig[out]：计算的签名数据
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_ecc_verify_secp256r1

函数名	tuya_ble_ecc_verify_secp256r1
函数原型	tuya_ble_status_t tuya_ble_ecc_verify_secp256r1(const uint8_t *p_pk, const uint8_t * p_data, uint32_t data_size,const uint8_t *p_sig)
功能概述	验证ECDSA签名
参数	p_pk[in] : 验签公钥 p_data[in] : 验证签名的数据 data_size[in]：验证签名的数据长度 p_sig[in]：验证的签名数据
返回值	TUYA_BLE_SUCCESS: 成功； 其他：失败。
备注	用于基于安全芯片加密的高级加密模式，否则无需实现。

tuya_ble_port_malloc

函数名	tuya_ble_port_malloc
函数原型	void *tuya_ble_port_malloc(uint32_t size)
功能概述	Allocate a memoryblock with required size.
参数	Size[in] : Required memory size.
返回值	The address of theallocated memory block. If the address is NULL, the memory allocation failed.
备注	该接口函数只需在 TUYA_BLE_USE_PLATFORM_MEMORY_HEAP==1 时移植实现。

tuya_ble_port_free

函数名	tuya_ble_port_free
函数原型	void tuya_ble_port_free(void *pv)
功能概述	Free a memory blockthat had been allocated.
参数	pv[in]：The address of memoryblock being freed.
返回值	None.
备注	该接口函数只需在 TUYA_BLE_USE_PLATFORM_MEMORY_HEAP==1时移植实现

Config说明

通过 `tuya_ble_config.h` 里的相关配置项，可将SDK 配置成不同的应用场景，例如多协议设备的通用配网、平台是否使用OS，设备通信能力，SDK 是否自管理授权信息等，必须新建一个 `custom_tuya_ble_config.h` 文件定义 SDK 需要的各种参数，然后在`tuya_ble_config.h`文件包含新建的`custom_tuya_ble_config.h`。

各配置项说明如下所示：

CUSTOMIZED_TUYA_BLE_CONFIG_FILE

宏定义	CUSTOMIZED_TUYA_BLE_CONFIG_FILE
依赖项	无
描述	客户自定义自配置文件，该定义文件会覆盖 `tuya_ble_config.h` 文件中的默认配置，应用程序必须新建一个自定义名称的应用配置文件，然后将该文件名字赋值给 `CUSTOMIZED_TUYA_BLE_CONFIG_FILE`，例如：keil 下：`CUSTOMIZED_TUYA_BLE_CONFIG_FILE = <custom_tuya_ble_config.h>`
备注	

CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE

宏定义	CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE
依赖项	无
描述	用户应用程序自定义的生产测试代码头文件引用
备注	例如，`#define CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE "custom_app_product_test.h"`，其中 `custom_app_product_test.h` 就是用户应用程序自定义的生产测试代码文件的头文件，该文件主要实现用户应用程序自定义的测试项目以及需要依赖具体芯片平台的涂鸦标准测试项目。

CUSTOMIZED_TUYA_BLE_APP_UART_COMMON_HEADER_FILE

宏定义	CUSTOMIZED_TUYA_BLE_APP_UART_COMMON_HEADER_FILE
依赖项	无
描述	
备注	应用无需关心

TUYA_BLE_PORT_PLATFORM_HEADER_FILE

宏定义	TUYA_BLE_PORT_PLATFORM_HEADER_FILE
依赖项	无
描述	芯片平台移植代码头文件
备注	举例: 1、在 nordic 平台移植适配 Tuya BLE SDK ,新建 `tuya_ble_port_nrf.c` 和 `tuya_ble_port_nrf.h` 代码源文件 用于实现 port 所有接口。 2、在新建的 `custom_tuya_ble_config.h` 文件中定义如下: `#define TUYA_BLE_PORT_PLATFORM_HEADER_FILE "tuya_ble_port_nrf.h"`

TUYA_BLE_USE_OS

宏定义	TUYA_BLE_USE_OS
依赖项	无
描述	如果芯片平台架构是基于 OS 的（例如 FreeRTOS）： #define TUYA_BLE_USE_OS 1 否则: #define TUYA_BLE_USE_OS 0
备注	

TUYA_BLE_SELF_BUILT_TASK

宏定义	TUYA_BLE_SELF_BUILT_TASK
依赖项	`#define TUYA_BLE_USE_OS 1`

描述	RTOS 下 Tuya BLE SDK 是否需要自建 task 来处理消息，如果是:
	`#define TUYA_BLE_SELF_BUILT_TASK 1`
	否则:
	`#define TUYA_BLE_SELF_BUILT_TASK 0`

备注	如果 SDK 需要自建 task 来处理消息，那么还需要配置 `TUYA_BLE_TASK_PRIORITY` 和 `TUYA_BLE_TASK_STACK_SIZE` 。

TUYA_BLE_TASK_PRIORITY

宏定义	TUYA_BLE_TASK_PRIORITY
依赖项	`#define TUYA_BLE_USE_OS 1` `#define TUYA_BLE_SELF_BUILT_TASK 1`
描述	RTOS 下 Tuya BLE SDK 自建 task 的 PRIORITY 优先级，例如: `#define TUYA_BLE_TASK_PRIORITY 1`
备注	具体的PRIORITY需要根据芯片平台所使用的的 RTOS 来定义。

TUYA_BLE_TASK_STACK_SIZE

宏定义	TUYA_BLE_TASK_STACK_SIZE
依赖项	`#define TUYA_BLE_USE_OS 1` `#define TUYA_BLE_SELF_BUILT_TASK 1`
描述	RTOS下Tuya BLE SDK 自建 task 的 stack 大小，例如: `#define TUYA_BLE_TASK_STACK_SIZE 256*10`
备注	

TUYA_BLE_DEVICE_COMMUNICATION_ABILITY

宏定义	TUYA_BLE_DEVICE_COMMUNICATION_ABILITY
依赖项	无

描述	设备能力定义：
	参数说明：
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_BLE`
	--- 是否支持 BLE；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_REGISTER_FROM_BLE`
	--- 是否通过 BLE 注册设备；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_MESH`
	--- 是否支持MESH；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_24G`
	--- 是否支持2.4G Wi-Fi；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_5G`
	--- 是否支持5G Wi-Fi；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_ZIGBEE`
	--- 是否支持zigbee；
	`TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_NB`
	--- 是否支持NB。

备注	如果是单 BLE 设备 需要定义如下： `#define TUYA_BLE_DEVICE_COMMUNICATION_ABILITY (TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_BLE |TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_REGISTER_FROM_BLE)`

TUYA_BLE_DEVICE_SHARED

宏定义	TUYA_BLE_DEVICE_SHARED
依赖项	无
描述	是否是共享类设备，目前主要用于共享类体脂秤产品
备注	

TUYA_BLE_DEVICE_UNBIND_MODE

宏定义	TUYA_BLE_DEVICE_UNBIND_MODE
依赖项	无
描述	如果定义为1，则在解绑设备时 SDK 需要删除存储的绑定信息； 如果定义为0，则不需要删除绑定信息， 主要用于共享类设备。
备注	如果不是共享类设备，必须定义为 1 。

TUYA_BLE_WIFI_DEVICE_REGISTER_MODE

宏定义	TUYA_BLE_WIFI_DEVICE_REGISTER_MODE
依赖项	设备能力必须支持wifi
描述	通过BLE对WIFI进行配网时，是否首先发送查询配网状态指令，如果是： #define TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 1 否则： #define TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 0
备注	目前暂不使用

TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT

宏定义	TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT
依赖项	无

描述	Tuya BLE SDK 是否需要管理授权信息，如果是：
	`#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 1`
	否则：
	`#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 0`

备注	对于不具备 Wi-Fi 能力的单 BLE 设备，该值必须定义为 1

TUYA_BLE_SECURE_CONNECTION_TYPE

宏定义	TUYA_BLE_SECURE_CONNECTION_TYPE
依赖项	无

描述	BLE 通信安全加密方式，定义如下：
	`TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY`
	--- encrypt with auth key；
	`TUYA_BLE_SECURE_CONNECTION_WITH_ECC`
	--- encrypt with ECDH；
	`TUYA_BLE_SECURE_CONNECTION_WTIH_PASSTHROUGH`
	--- no encrypt；
	`TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY_ADVANCED_ENCRYPTION`
	--- advanced encrypt(security chip) with auth key;

备注	目前仅支持 `TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY` 和 `TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY_ADVANCED_ENCRYPTION` 一般定义如下值即可： `#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY`

TUYA_BLE_ADVANCED_ENCRYPTION_AUTH_ON_CONNECT

宏定义	TUYA_BLE_ADVANCED_ENCRYPTION_AUTH_ON_CONNECT
依赖项	`#define TUYA_BLE_SECURE_CONNECTION_TYPE TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY_ADVANCED_ENCRYPTION`
描述	定义为 1 表示 高级安全加密 下每次蓝牙连接都需要进行证书认证。 定义为 0 表示 高级安全加密下每次蓝牙连接不需要进行证书认证。
备注	

TUYA_BLE_INCLUDE_CJSON_COMPONENTS

宏定义	TUYA_BLE_INCLUDE_CJSON_COMPONENTS
依赖项	`#define TUYA_BLE_SECURE_CONNECTION_TYPE TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY_ADVANCED_ENCRYPTION`
描述	是否需要包含 CJSON 库。
备注	CJSON 库主要用于 高级安全加密。

TUYA_BLE_DEVICE_MAC_UPDATE

宏定义	TUYA_BLE_DEVICE_MAC_UPDATE
依赖项	无
描述	是否需要更新 mac 地址为涂鸦提供的地址。
备注	

TUYA_BLE_DEVICE_MAC_UPDATE_RESET

宏定义	TUYA_BLE_DEVICE_MAC_UPDATE_RESET
依赖项	无

描述	芯片平台更新MAC地址后是否需要重启生效，如果是：
	`#define TUYA_BLE_DEVICE_MAC_UPDATE_RESET 1`
	否则：
	`#define TUYA_BLE_DEVICE_MAC_UPDATE_RESET 0`

备注	

TUYA_BLE_USE_PLATFORM_MEMORY_HEAP

宏定义	TUYA_BLE_USE_PLATFORM_MEMORY_HEAP
依赖项	无

描述	是否需要 Tuya BLE SDK 使用自己的内存堆，如果是：
	`#define TUYA_BLE_USE_PLATFORM_MEMORY_HEAP 0`
	否则：
	`#define TUYA_BLE_USE_PLATFORM_MEMORY_HEAP 1`

备注	如果定义为 1，需要 port 层移植实现内存管理函数供 Tuya BLE SDK 使用。

TUYA_BLE_DATA_MTU_MAX

宏定义	TUYA_BLE_DATA_MTU_MAX
依赖项	无
描述	设备定义的 MTU 值（ATT MTU - 3）。
备注	nordic 平台示例： `#define TUYA_BLE_DATA_MTU_MAX (NRF_SDH_BLE_GATT_MAX_MTU_SIZE - 3)`

TUYA_BLE_LOG_ENABLE

宏定义	TUYA_BLE_LOG_ENABLE
依赖项	无

描述	是否开启sdk log，如果开启：
	`#define TUYA_BLE_LOG_ENABLE 1`
	否则：
	`#define TUYA_BLE_LOG_ENABLE 0`

备注	开启LOG会增加一定的ROM占用，建议debug版本开启，release版关掉。

TUYA_BLE_LOG_COLORS_ENABLE

宏定义	TUYA_BLE_LOG_COLORS_ENABLE
依赖项	`#define TUYA_BLE_LOG_ENABLE 1`

描述	是否开启sdk log多颜色显示，如果开启：
	`#define TUYA_BLE_LOG_COLORS_ENABLE 1`
	否则：
	`#define TUYA_BLE_LOG_COLORS_ENABLE 0`

备注	并不是所有的 log 显示终端都支持，例如使用 J-Link 的 RTT 时不支持。

TUYA_BLE_LOG_LEVEL

宏定义	TUYA_BLE_LOG_LEVEL
依赖项	`#define TUYA_BLE_LOG_ENABLE 1`

描述	定义sdk log的显示等级，分为如下几个等级：
	`TUYA_BLE_LOG_LEVEL_ERROR`
	`TUYA_BLE_LOG_LEVEL_WARNING`
	`TUYA_BLE_LOG_LEVEL_INFO`
	`TUYA_BLE_LOG_LEVEL_DEBUG`
	如果只需要打印错误信息，则定义为：
	`#define TUYA_BLE_LOG_LEVEL TUYA_BLE_LOG_LEVEL_ERROR`

备注	

TUYA_APP_LOG_ENABLE

宏定义	TUYA_APP_LOG_ENABLE
依赖项	无

描述	是否开启app log，如果开启：
	`#define TUYA_APP_LOG_ENABLE 1`
	否则：
	`#define TUYA_APP_LOG_ENABLE 0`

备注	开启LOG会增加一定的ROM占用，建议debug版本开启，release版关掉。

TUYA_APP_LOG_COLORS_ENABLE

宏定义	TUYA_APP_LOG_COLORS_ENABLE
依赖项	`#define TUYA_APP_LOG_ENABLE 1`

描述	是否开启app log多颜色显示，如果开启：
	`#define TUYA_APP_LOG_COLORS_ENABLE 1`
	否则：
	`#define TUYA_APP_LOG_COLORS_ENABLE 0`

备注	并不是所有的app log 显示终端都支持，例如使用 J-Link 的 RTT 时不支持。

TUYA_APP_LOG_LEVEL

宏定义	TUYA_APP_LOG_LEVEL
依赖项	`#define TUYA_APP_LOG_ENABLE 1`

描述	定义app log的显示等级，分为如下几个等级：
	`TUYA_APP_LOG_LEVEL_ERROR`
	`TUYA_APP_LOG_LEVEL_WARNING`
	`TUYA_APP_LOG_LEVEL_INFO`
	`TUYA_APP_LOG_LEVEL_DEBUG`
	如果只需要打印错误信息，则定义为：
	`#define TUYA_APP_LOG_LEVEL TUYA_APP_LOG_LEVEL_ERROR`

备注	

TUYA_BLE_BEACON_KEY_ENABLE

宏定义	TUYA_BLE_BEACON_KEY_ENABLE
依赖项	无
描述	是否使能 beacon key ， 如果应用程序需要使用 beacon 发送 dp 数据，那么就需要开启。
备注	

TUYA_BLE_FEATURE_WEATHER_ENABLE

宏定义	TUYA_BLE_FEATURE_WEATHER_ENABLE
依赖项	无
描述	是否使能 天气服务
备注	开启天气服务，必须联系涂鸦项目经理开通，否则只从 sdk 端使能没有作用。

TUYA_BLE_LINK_LAYER_ENCRYPTION_SUPPORT_ENABLE

宏定义	TUYA_BLE_LINK_LAYER_ENCRYPTION_SUPPORT_ENABLE
依赖项	无
描述	是否使能蓝牙 LINK 层配对加密机制，如果使能 应用程序移植sdk时就需要实现 port接口中的 `tuya_ble_device_info_characteristic_value_update()` ,同时还需要定义 Read Characteristic uuid
备注	

TUYA_BLE_AUTO_REQUEST_TIME_CONFIGURE

宏定义	TUYA_BLE_AUTO_REQUEST_TIME_CONFIGURE
依赖项	无
描述	Tuya BLE SDK 在每次和 APP 建立连接后是否需要主动请求一次时间; 0 --- 不请求; 1 --- 请求一次云端时间; 2 --- 请求一次手机本地时间。
备注	

TUYA_NV_ERASE_MIN_SIZE

宏定义	TUYA_NV_ERASE_MIN_SIZE
依赖项	无
描述	给BLE SDK 分配的NV（flash）最小擦除单位，例如: `#define TUYA_NV_ERASE_MIN_SIZE 4096`
备注	根据port层对NV接口的实现方式来定义。

TUYA_NV_WRITE_GRAN

宏定义	TUYA_NV_WRITE_GRAN
依赖项	无
描述	给BLE SDK 分配的NV（flash）写入粒度，例如： `#define TUYA_NV_WRITE_GRAN 4` 只能以4字节字方式写入。
备注	根据port层对NV接口的实现方式来定义。

TUYA_NV_START_ADDR

宏定义	TUYA_NV_START_ADDR
依赖项	无
描述	给 BLE SDK 分配的 NV（flash）起始地址，例如： `#define TUYA_NV_START_ADDR 0x1000`
备注	

TUYA_NV_AREA_SIZE

宏定义	TUYA_NV_AREA_SIZE
依赖项	无
描述	给BLE SDK 分配的NV（flash）大小，例如： `#define TUYA_NV_AREA_SIZE (4*TUYA_NV_ERASE_MIN_SIZE)` 必须是`TUYA_NV_ERASE_MIN_SIZE`的倍数。
备注	

TUYA_BLE_APP_VERSION_STRING

宏定义	TUYA_BLE_APP_VERSION_STRING
依赖项	无
描述	基于 Tuya BLE SDK 开发的应用固件版本号，字符串格式。

备注	假设 设备应用程序定义了固件版本号为
	`#define TY_APP_VER_NUM 0x0101`
	`#define TY_APP_VER_STR "1.1"`
	那么在`custom_tuya_ble_config.h`文件中做如下定义：
	`#define TUYA_BLE_APP_VERSION_STRING TY_APP_VER_STR`

TUYA_BLE_APP_BUILD_FIRMNAME_STRING

宏定义	TUYA_BLE_APP_BUILD_FIRMNAME_STRING
依赖项	无
描述	基于 Tuya BLE SDK 开发的应用固件指纹标识，字符串格式。
备注	定义方法同 `TUYA_BLE_APP_VERSION_STRING`

TUYA_BLE_APP_FIRMWARE_KEY

宏定义	TUYA_BLE_APP_FIRMWARE_KEY
依赖项	无
描述	基于 Tuya BLE SDK 开发的应用固件key， 字符串格式。
备注	定义方法同 `TUYA_BLE_APP_VERSION_STRING`

API介绍

Tuya BLE SDK 提供封装好的 API 用于设备应用程序实现 BLE 相关的管理、通信等，API 函数定义在 `tuya_ble_api.c` 和 `tuya_ble_api.h` 文件中，客户无需更改，有兴趣可以阅读源码理解实现原理，下面对各个 API 做个介绍。

tuya_ble_main_tasks_exec

函数名	tuya_ble_main_tasks_exec
函数原型	void tuya_ble_main_tasks_exec(void)
功能概述	非 OS 架构下 BLE SDK 的消息事件主调度器，应用程序必须在主循环中调用。
参数	无
返回值	无
备注	必须在主循环中调用

示例：例如在 nrf52832 平台下的调用位置如下所示：

```
/**@brief Function for handling the idle state (main loop).
 *
 * @details If there is no pending log operation, then sleep until next the next
event occurs.
 */
static void idle_state_handle(void)
{
```

```
    ret_code_t err_code;
    err_code = nrf_ble_lesc_request_handler();
    APP_ERROR_CHECK(err_code);

    tuya_ble_main_tasks_exec();

    if ((NRF_LOG_PROCESS() == false)&&(tuya_ble_sleep_allowed_check()))
    {
        nrf_pwr_mgmt_run();
    }

}
```

tuya_ble_gatt_receive_data

函数名	tuya_ble_gatt_receive_data
函数原型	tuya_ble_status_t tuya_ble_gatt_receive_data(uint8_t*p_data,uint16_t len);
功能概述	通过调用该函数将蓝牙收到的 gatt 数据发送至 SDK 。
参数	p_data[in] : 指向要发送的数据; len[in] : 要发送的数据长度，不能超过 `TUYA_BLE_DATA_MTU_MAX`。
返回值	TUYA_BLE_SUCCESS： 发送成功; TUYA_BLE_ERR_INTERNAL： 发送失败。
备注	该api用于将蓝牙底层收到的数据发送给Tuya BLE SDK，必须在 Tuya Write Characteristic UUID `00000001-0000-1001-8001-00805F9B07D0` 的写回调函数中调用。

示例（nrf52832示例demo，基于nordic）：

```
/**@brief Function for handling the data from the Nordic UART Service.
 *
 * @details This function will process the data received from the Nordic UART
BLE Service and send
 *          it to the UART module.
 *
 * @param[in] p_evt       Nordic UART Service event.
 */
/**@snippet [Handling the data received over BLE] */
static void nus_commdata_handler(ble_nus_evt_t * p_evt)
{

    if (p_evt->type == BLE_NUS_EVT_RX_DATA)
    {
        tuya_ble_gatt_receive_data((uint8_t*)(p_evt-
>params.rx_data.p_data),p_evt->params.rx_data.length);

    }

```

```
}
```

tuya_ble_common_uart_receive_data

函数名	tuya_ble_common_uart_receive_data
函数原型	tuya_ble_status_t tuya_ble_common_uart_receive_data (uint8_t *p_data,uint16_t len);
功能概述	通过调用该函数将 uart 接收数据发送至 SDK，uart 接口主要用于产测授权。
参数	p_data[in] : 指向要发送的数据； len[in] : 要发送的数据长度。
返回值	TUYA_BLE_SUCCESS： 发送成功； 其他： 发送失败。
备注	该函数内部有调用 malloc，如果应用配置使用的是平台提供的 malloc 接口，要确认平台 malloc 是否支持中断调用，如果不支持，一定不能在 UART 中断里调用 `tuya_ble_common_uart_receive_data()`。

tuya_ble_common_uart_send_full_instruction_received

函数名	tuya_ble_common_uart_send_full_instruction_received
函数原型	tuya_ble_status_t tuya_ble_common_uart_send_full_instruction_received (uint8_t *p_data,uint16_t len);
功能概述	通过调用该函数将 uart 收到并解析好的完整涂鸦 uart 指令（包含 cmd/data/checksum）发送至 BLE SDK。
参数	p_data[in] : 指向要发送的完整指令数据； len[in] : 要发送的指令数据长度。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_PARAM： 参数有误； TUYA_BLE_ERR_NO_MEM： 内存申请失败； TUYA_BLE_ERR_BUSY： BLE SDK内部忙。
备注	该函数内部有调用 malloc，如果应用配置使用的是平台提供的 malloc 接口，要确认平台 malloc 是否支持中断调用，如果不支持，一定不能在 UART 中断里调用 `tuya_ble_common_uart_send_full_instruction_received()`。

说明：

1、Tuya BLE SDK 集成产测授权功能模块，产测授权模块通过uart和PC端的产测授权工具进行通信，uart通信有一套完整的指令格式，具体参照《蓝牙通用产测授权协议》。

2、Tuya BLE SDK 包含 uart 通信指令解析功能，应用只需要在收到uart的数据的地方调用 `tuya_ble_common_uart_receive_data()` 函数即可，当然应用也可以自己解析出完整的uart通信指令，然后通过调用 `tuya_ble_common_uart_send_full_instruction_received()` 函数发送完整指令给 Tuya BLE SDK。

tuya_ble_device_update_product_id

函数名	tuya_ble_device_update_product_id
函数原型	tuya_ble_status_t tuya_ble_device_update_product_id (tuya_ble_product_id_type_t type, uint8_t len, uint8_t* p_buf);
功能概述	更新 product id 函数。
参数	type [in] : id 类型； len[in] : id 长度； p_buf[in] : id 数据。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数有误； TUYA_BLE_ERR_INTERNAL：内部错误。
备注	在基于 BLE SDK 的 Soc 开发方案中一般不需要调用此函数，因为 product id 一般不会变。

tuya_ble_device_update_login_key

函数名	tuya_ble_device_update_login_key
函数原型	tuya_ble_status_t tuya_ble_device_update_login_key(uint8_t* p_buf, uint8_t len);
功能概述	更新 login key 函数。
参数	len[in] : 长度。 p_buf[in] : loginkey。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数有误； TUYA_BLE_ERR_INTERNAL：内部错误。
备注	该api主要用于Wi-Fi/BLE双协议设备中（通过BLE发送配网信息给Wi-Fi,设备通过Wi-Fi向云端注册设备并将注册成功后的login key调用此函数发送给BLE SDK，同时也需要更新绑定标志位）。

tuya_ble_device_update_beacon_key

函数名	tuya_ble_device_update_beacon_key
函数原型	tuya_ble_status_t tuya_ble_device_update_beacon_key(uint8_t* p_buf, uint8_t len);
功能概述	更新 beacon key 函数。
参数	len[in] : 长度。 p_buf[in] : loginkey。
返回值	TUYA_BLE_SUCCESS ： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数有误； TUYA_BLE_ERR_INTERNAL：内部错误。
备注	该api主要用于Wi-Fi/BLE双协议设备中（通过BLE发送配网信息给Wi-Fi,设备通过Wi-Fi向云端注册设备并将注册成功后的beacon key调用此函数发送给BLE SDK，同时也需要更新绑定标志位和 login key等）。

tuya_ble_device_update_bound_state

函数名	tuya_ble_device_update_bound_state
函数原型	tuya_ble_status_t tuya_ble_device_update_bound_state(uint8_t state);
功能概述	更新注册绑定状态。
参数	state[in] : 1-设备已注册绑定，0-设备未注册绑定。
返回值	TUYA_BLE_SUCCESS ： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数有误； TUYA_BLE_ERR_INTERNAL：内部错误。
备注	该api主要用于 Wi-Fi/BLE 双协议设备中（设备通过 Wi-Fi 链路注册绑定成功后调用此函数更新绑定状态给 BLE SDK，同时也需要更新 login key）。

tuya_ble_device_update_mcu_version

函数名	tuya_ble_device_update_mcu_version
函数原型	tuya_ble_status_t tuya_ble_device_update_mcu_version(uint32_t mcu_firmware_version, uint32_t mcu_hardware_version);
功能概述	更新外部mcu版本号。
参数	mcu_firmware_version [in] : MCU 固件版本号，例如0x010101表示v1.1.1； mcu_hardware_version [in] : MCU 硬件版本号（PCBA版本号）；
返回值	TUYA_BLE_SUCCESS： 发送成功；其他： 失败。
备注	该 api 主要用于开发蓝牙 BLE 模组，SOC 开发方案不需要使用。

tuya_ble_sdk_init

函数名	tuya_ble_sdk_init
函数原型	tuya_ble_status_t tuya_ble_sdk_init(tuya_ble_device_param_t * param_data);
功能概述	Tuya BLE SDK初始化函数。
参数	param_data [in] : 初始化参数。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数无效。
备注	BLE SDK初始化函数，应用程序必须调用此函数初始化SDK，否则SDK不能运行。

参数说明:

tuya_ble_device_param_t 结构体如下所示:

```
typedef struct {
    uint8_t use_ext_license_key;     /**< If use the license key stored by the SDK,initialized to 0, Otherwise 1.*/
    uint8_t device_id_len;           /**<if 20,Compressed into 16.*/
    uint8_t device_id[DEVICE_ID_LEN_MAX];
    uint8_t auth_key[AUTH_KEY_LEN];
    tuya_ble_gap_addr_t mac_addr;
    uint8_t mac_addr_string[MAC_STRING_LEN];

    tuya_ble_product_id_type_t p_type;
    uint8_t product_id_len;
    uint8_t product_id[TUYA_BLE_PRODUCT_ID_MAX_LEN];
    uint8_t adv_local_name_len;
    uint8_t adv_local_name[TUYA_BLE_ADV_LOCAL_NAME_MAX_LEN];    /**< Only supported when TUYA_BLE_PROTOCOL_VERSION_HIGN >= 4. */
```

```
    uint32_t firmware_version;                          /**< 0x00010102 :
v1.1.2 */
    uint32_t hardware_version;

    uint8_t device_vid[DEVICE_VIRTUAL_ID_LEN];
    uint8_t login_key[LOGIN_KEY_LEN];
    uint8_t beacon_key[BEACON_KEY_LEN];
    uint8_t bound_flag;

    uint8_t reserve_1;
    uint8_t reserve_2;
} tuya_ble_device_param_t;
```

各成员变量含义:

1、use_ext_license_key：是否使用应用程序传入的 license (device id 、 auth key 和 mac地址)，`TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT`定义为1时，如果use_ext_license_key为1，Tuya BLE SDK 将会使用传入的 device id 、auth key以及mac地址（必须传入，否则将不能被涂鸦app绑定），但是仍然会自行管理和存储其他绑定信息，如果use_ext_license_key为 0，Tuya BLE SDK 将会使用自行存储管理的 license （通过涂鸦产测工具授权获得） 和 绑定信息，此时不需要传入 license, device_id_len赋值为0即可。

`TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT`定义为0时表示 应用程序自行管理 license 和 绑定信息，sdk不做任何管理，初始化 sdk 时必须传入这些信息， use_ext_license_key 也就必须赋值为1 。

2、device_id 、auth_key、mac_addr 、mac_addr_string ： 是 tuya iot 分配给设备的唯一 license，在产测授权时通过产测工具写入，并且 一一对应，单BLE设备经过产测授权后，Tuya BLE SDK 会自动管理授权信息（license），device id 和 auth key 上面已有介绍，该license 的使用 请参照下面的代码示例。

3、product id ： 产品 id 简称 pid，是在 涂鸦 iot 平台新建产品时自动生成的，生成后不会改变，需要应用代码以常量的形式保存并且在初始化 Tuya BLE SDK 时传入。

4、p_type ： product id 的类型，有 TUYA_BLE_PRODUCT_ID_TYPE_PID 和 TUYA_BLE_PRODUCT_ID_TYPE_PRODUCT_KEY 两种类型，目前只支持 TUYA_BLE_PRODUCT_ID_TYPE_PID 类型。

5、adv_local_name ： adv_local_name 和 adv_local_name_len 是应用程序自定义蓝牙广播名字的变量，如果adv_local_name_len 为 0 ，那么Tuya BLE SDK 将默认使用 “TY” 作为蓝牙广播名字，应用程序在初始化蓝牙GAP 时设置的蓝牙名字必须和 传给 SDK 的 adv_local_name 一致，最多支持5个字符。

6、firmware_version 和 hardware_version ： 固件版本号和PCBA硬件版本号，四字节，例如 0x00010102表示v1.1.2，0x0101 表示 v1.1 。

7、device_vid ： 设备虚拟 id，设备注册绑定后由 iot 云端生成，主要作用是设备绑定解绑再绑定时通过该 id 来查找云端对该设备的历史数据记录，对于单 BLE 设备来说，赋值为 0 即可，对于 Wi-Fi / BLE 多协议设备需要代入。

8、login key 、beacon key 和 bound flag ： 对于单 BLE 设备赋值为 0 即可，对于双协议设备需要代入。

nrf52832平台的Tuya BLE SDK初始化示例:

```
static const char auth_key_test[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
static const char device_id_test[] = "yyyyyyyyyyyyyyyy";
```

```
static const char mac_test[]  = "112233445566"; //The actual MAC address is :
11:22:33:44:55:66
static const char device_local_name[] = "SDK20";  //Maximum support 5 characters

#define APP_PRODUCT_ID  "vvvvvvvv"

#define TY_APP_VER_NUM       0x0101
#define TY_APP_VER_STR       "1.1"

#define TY_HARD_VER_NUM      0x0100
#define TY_HARD_VER_STR      "1.0"

void tuya_ble_app_init(void)
{
    tuya_ble_device_param_t device_param = {0};

    device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
    device_param.product_id_len = 8;
    memcpy(device_param.product_id,APP_PRODUCT_ID,8);
    device_param.firmware_version = TY_APP_VER_NUM;
    device_param.hardware_version = TY_HARD_VER_NUM;
    device_param.adv_local_name_len = strlen(device_local_name);

memcpy(device_param.adv_local_name,device_local_name,device_param.adv_local_name
_len);

    device_param.use_ext_license_key = 1; //If use the license stored by the
SDK,initialized to 0, Otherwise 1.

    if(device_param.use_ext_license_key==1)
    {
        device_param.device_id_len = 16;
        memcpy(device_param.auth_key,(void *)auth_key_test,32);
        memcpy(device_param.device_id,(void *)device_id_test,16);
        memcpy(device_param.mac_addr_string,mac_test,12);
        device_param.mac_addr.addr_type = TUYA_BLE_ADDRESS_TYPE_RANDOM;
    }

    tuya_ble_sdk_init(&device_param);
    tuya_ble_callback_queue_register(tuya_cb_handler);

    tuya_ota_init();
}

/*nrf52832示例*/
int main(void)
{
    bool erase_bonds;

    // Initialize.
    uart_init();
    app_log_init();
    timers_init();
    buttons_leds_init(&erase_bonds);
    power_management_init();
    ble_stack_init();
    gap_params_init();
    gatt_init();
```

```
    services_init();
    advertising_init();
    conn_params_init();

    tuya_ble_app_init();  //
    advertising_start();



    // Enter main loop.
    for (;;)
    {
        idle_state_handle();
    }
}
```

tuya_ble_dp_data_send

函数名	tuya_ble_dp_data_send
函数原型	tuya_ble_status_t tuya_ble_dp_data_send(uint32_t sn,tuya_ble_dp_data_send_type_t type,tuya_ble_dp_data_send_mode_t mode,tuya_ble_dp_data_send_ack_t ack,uint8_t *p_dp_data,uint32_t dp_data_len)
功能概述	发送dp数据。
参数	sn[in] : 发送序号; type[in] : 发送类型，分为主动发送和应答查询发送; mode[in] : 发送模式; ack[in] : 是否需要应答标志; p_dp_data [in] : dp数据; dp_data_len[in] : 数据长度，最大不能超过 TUYA_BLE_SEND_MAX_DATA_LEN-7 ,其中 TUYA_BLE_SEND_MAX_DATA_LEN 可配置。
返回值	TUYA_BLE_SUCCESS ： 发送成功; TUYA_BLE_ERR_INVALID_PARAM：参数无效; TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开; TUYA_BLE_ERR_NO_MEM：内存申请失败; TUYA_BLE_ERR_INVALID_LENGTH：数据长度错误； TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	应用程序通过调用该函数上报 DP 数据到 涂鸦 App。

说明:

1、涂鸦 IoT平台是以 dp 模型管理数据，任何设备产生的数据都需要抽象为 dp 数据形式，一个 dp 数据由四部分组成（具体参考 涂鸦IoT 平台上的相关介绍）：

Dp_id： 1个字节，在开发平台注册的 dp_id 序号。

Dp_type：1 个字节，dp点类型。

`#define DT_RAW   0` raw类型；

`#define DT_BOOL   1` 布尔类型；

`#define DT_VALUE 2` 数值类型，其范围在iot平台注册时指定；

`#define DT_STRING 3` 字符串类型；

`#define DT_ENUM   4` 枚举类型；

`#define DT_BITMAP 5` 位映射类型；

Dp_len：两个字节，每个Dp数据类型的最大数据长度在涂鸦 IoT平台定义时指定。

Dp_data：数据，dp_len 个字节。

2、该 函数的参数 p_dp_data 指向的数据必须以下表格式组装：

Dp1的数据	~				Dpn的数据			
1	2	3 - 4	5~	~	n	n+1	n+2 - n+3	n+4~
Dp_id	Dp_type	Dp_len	Dp_data	~	Dp_id	Dp_type	Dp_len	Dp_data

3、一次可发送多个dp数据，只要总长度不超过限制即可，最大长度为 `TUYA_BLE_SEND_MAX_DATA_LEN-7` ,其中 `TUYA_BLE_SEND_MAX_DATA_LEN` 可配置。

4、sn 是由应用程序自行定义管理的序号，一般从0开始，每发送一次 递增1 ，用于应用程序统计发送次数以及管理涂鸦智能 app 的发送响应，典型应用场景是 应用程序定时或者连续调用多次该函数发送dp数据后，在 SDK 回调函数中通过该 sn 来判断哪一次发送是成功的，sn 可以和下面的 `tuya_ble_dp_data_with_time_send()` 共用，因为 `tuya_ble_dp_data_with_time_send()` 和 `tuya_ble_dp_data_send()` 使用不同的回调事件ID。

5、type表示本次发送是应用程序的主动行为 还是 对 涂鸦智能 App 查询dp数据指令的响应，如下所示定义：

```
typedef enum {
    DP_SEND_TYPE_ACTIVE = 0,        // The device actively sends dp data.
    DP_SEND_TYPE_PASSIVE,           // The device passively sends dp data. For
example, in order to answer the dp query command of the mobile app. Currently
only applicable to WIFI+BLE combo devices.
} tuya_ble_dp_data_send_type_t;
```

至于发送dp数据时是带 `DP_SEND_TYPE_ACTIVE` 还是 `DP_SEND_TYPE_PASSIVE` 有具体业务功能定义，响应手机App的查询指令的发送不一定要带 `DP_SEND_TYPE_PASSIVE` 参数。

6、mode的定义如下所示：

```
typedef enum {
    DP_SEND_FOR_CLOUD_PANEL = 0,    // The mobile app uploads the received dp
data to the cloud and also sends it to the panel for display.
    DP_SEND_FOR_CLOUD,              // The mobile app will only upload the
received dp data to the cloud.
    DP_SEND_FOR_PANEL,              // The mobile app will only send the received
dp data to the panel display.
    DP_SEND_FOR_NONE,               // Neither uploaded to the cloud nor sent to
the panel display.
} tuya_ble_dp_data_send_mode_t;
```

涂鸦智能 App 的面板可以认为是某一个产品的 UI 界面，同时还负责具体产品定义的业务功能逻辑，有些产品定义的某一个 dp 数据只是用来表示临时数据，只需要发送到手机 App 面板显示，不需要发送到云端存储，所以只需要带 `DP_SEND_FOR_PANEL` 参数发送，例如智能体脂秤称重过程中的动态数据。

7、ack表示是否需要涂鸦智能 App 的响应，定义如下所示：

```
typedef enum {
    DP_SEND_WITH_RESPONSE = 0,    //  Need a mobile app to answer.
    DP_SEND_WITHOUT_RESPONSE,     //  No need for mobile app to answer.
} tuya_ble_dp_data_send_ack_t;
```

是否需要响应也是由具体的产品业务功能逻辑决定的，响应主要是为了反馈给设备应用程序 手机 App 收到了发送的 dp 数据。

tuya_ble_dp_data_with_time_send

函数名	tuya_ble_dp_data_with_time_send
函数原型	tuya_ble_status_t tuya_ble_dp_data_with_time_send(uint32_t sn,tuya_ble_dp_data_send_mode_t mode,tuya_ble_dp_data_send_time_type_t time_type,uint8_t *p_time_data,uint8_t *p_dp_data,uint32_t dp_data_len)
功能概述	发送带时间戳的 dp 数据。
参数	sn[in] : 发送序号； mode[in] : 发送模式； time_type[in] : 时间类型； p_time_data[in] : 时间数据； p_dp_data [in] : dp数据； dp_data_len[in] : 数据长度，最大不能超过 TUYA_BLE_SEND_MAX_DATA_LEN-21 ,其中 TUYA_BLE_SEND_MAX_DATA_LEN 可配置。
返回值	TUYA_BLE_SUCCESS ： 发送成功； TUYA_BLE_ERR_INVALID_PARAM：参数无效； TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开； TUYA_BLE_ERR_NO_MEM：内存申请失败； TUYA_BLE_ERR_INVALID_LENGTH：数据长度错误； TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	

1、sn、mode、p_dp_data 和 dp_data_len 上面已有介绍，参照 `tuya_ble_dp_data_send()` 的说明，其中该函数的 dp_data_len 最大不能超过 `TUYA_BLE_SEND_MAX_DATA_LEN-21`，`TUYA_BLE_SEND_MAX_DATA_LEN` 可配置。

2、该函数没有 ack 参数，因为带时间戳的dp数据发送必须要有手机 App 的响应。

3、time_type 和 p_time_data : 时间类型和时间数据指针，时间类型定义如下所示：

```
typedef enum {
    DP_TIME_TYPE_MS_STRING = 0,
    DP_TIME_TYPE_UNIX_TIMESTAMP,
} tuya_ble_dp_data_send_time_type_t;
```

`DP_TIME_TYPE_MS_STRING` 表示参数 p_time_data 指向的时间数据是毫秒级字符串数据，必须是13个字符，例如 “1600777955000” 单位是毫秒；

`DP_TIME_TYPE_UNIX_TIMESTAMP` 表示参数 p_time_data 指向的时间数据是四字节的 unix 时间戳，大端格式，例如 1600777955 = 0x5F69EEE3, 那么 p_time_data 指向数组是 {0x5F,0x69,0xEE,0xE3} ;

无论是 `DP_TIME_TYPE_MS_STRING` 类型的时间还是 `DP_TIME_TYPE_UNIX_TIMESTAMP` 类型的时间，都必须是格林威治时间。

tuya_ble_connected_handler

函数名	tuya_ble_connected_handler
函数原型	void tuya_ble_connected_handler(void)
功能概述	蓝牙连接回调函数。
参数	无。
返回值	无。
备注	应用程序需要在芯片平台 SDK 的蓝牙连接回调处调用此函数，Tuya BLE SDK是根据此函数的执行来管理内部蓝牙连接状态的。

tuya_ble_disconnected_handler

函数名	tuya_ble_disconnected_handler
函数原型	void tuya_ble_disconnected_handler(void)
功能概述	蓝牙断开连接回调函数。
参数	无。
返回值	无。
备注	应用代码需要在芯片平台 SDK 的蓝牙断开连接回调处调用此函数，Tuya BLE SDK 是根据此函数的执行来管理内部蓝牙连接状态的。

nrf52832平台调用`tuya_ble_connected_handler()`和`tuya_ble_disconnected_handler()`的示例:

```
/**@brief Function for handling BLE events.
 *
 * @param[in]   p_ble_evt   Bluetooth stack event.
 * @param[in]   p_context   Unused.
 */
static void ble_evt_handler(ble_evt_t const * p_ble_evt, void * p_context)
{
    uint32_t err_code;

    switch (p_ble_evt->header.evt_id)
    {
    case BLE_GAP_EVT_CONNECTED:

        NRF_LOG_INFO("Connected");

        tuya_ble_connected_handler();

        err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
        APP_ERROR_CHECK(err_code);
        m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
        err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle);
```

```
        APP_ERROR_CHECK(err_code);

        break;

    case BLE_GAP_EVT_DISCONNECTED:

        NRF_LOG_INFO("Disconnected");

        tuya_ble_disconnected_handler();

        tuya_ota_init_disconnect();
        // LED indication will be changed when advertising starts.
        m_conn_handle = BLE_CONN_HANDLE_INVALID;
        break;

    case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
    {
        NRF_LOG_DEBUG("PHY update request.");
        ble_gap_phys_t const phys =
        {
            .rx_phys = BLE_GAP_PHY_AUTO,
            .tx_phys = BLE_GAP_PHY_AUTO,
        };
        err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.conn_handle,
&phys);
        APP_ERROR_CHECK(err_code);
    }
    break;

    case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
        // Pairing not supported
        err_code = sd_ble_gap_sec_params_reply(m_conn_handle,
BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
        APP_ERROR_CHECK(err_code);
        break;

    case BLE_GATTS_EVT_SYS_ATTR_MISSING:
        // No system attributes have been stored.
        err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0, 0);
        APP_ERROR_CHECK(err_code);
        break;

    case BLE_GATTC_EVT_TIMEOUT:
        // Disconnect on GATT Client timeout event.
        err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.conn_handle,

BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
        APP_ERROR_CHECK(err_code);
        break;

    case BLE_GATTS_EVT_TIMEOUT:
        // Disconnect on GATT Server timeout event.
        err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.conn_handle,

BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION);
        APP_ERROR_CHECK(err_code);
        break;
```

```
    default:
        // No implementation needed.
        break;
    }
}
```

tuya_ble_link_encrypted_handler

函数名	tuya_ble_link_encrypted_handler
函数原型	void tuya_ble_link_encrypted_handler(void)
功能概述	蓝牙Link层加密回调函数。
参数	无。
返回值	无。
备注	1、应用程序需要在芯片平台 SDK 的蓝牙Link层加密成功回调处调用此函数，Tuya BLE SDK是根据此函数的执行来管理内部蓝牙连接状态。 2、主要用于开启蓝牙Link层加密机制的场景，否则不需要调用该函数。

nordic 平台示例如下：

```
/**@brief Function for handling Peer Manager events.
 *
 * @param[in] p_evt  Peer Manager event.
 */
static void pm_evt_handler(pm_evt_t const * p_evt)
{
    pm_handler_on_pm_evt(p_evt);
    pm_handler_flash_clean(p_evt);

    switch (p_evt->evt_id)
    {
        case PM_EVT_BONDED_PEER_CONNECTED:

            TUYA_APP_LOG_DEBUG("Bonded peer CONNECTED!");
            break;
        case PM_EVT_CONN_SEC_SUCCEEDED:
            tuya_ble_link_encrypted_handler();
            TUYA_APP_LOG_DEBUG("The link has been encrypted!");
            break;
        case PM_EVT_PEERS_DELETE_SUCCEEDED:
            //advertising_start();
            TUYA_APP_LOG_DEBUG("Erase bonds SUCCEED!");
            break;

        default:
            break;
    }
}
```

####

tuya_ble_adv_data_connecting_request_set

函数名	tuya_ble_adv_data_connecting_request_set
函数原型	tuya_ble_status_t tuya_ble_adv_data_connecting_request_set(uint8_t on_off)
功能概述	请求连接标志置位函数
参数	on_off[in] : 0 - 清除请求连接标位；1 - 置位请求连接标位；
返回值	无。
备注	具体应用方法请咨询涂鸦技术支持人员，暂未开放。

tuya_ble_data_passthrough

函数名	tuya_ble_data_passthrough
函数原型	tuya_ble_status_t tuya_ble_data_passthrough(uint8_t *p_data,uint32_t len)
功能概述	透传数据。
参数	p_data [in] : 需要透传的数据 Len[in] : 数据长度，最大不能超过 TUYA_BLE_SEND_MAX_DATA_LEN。
返回值	TUYA_BLE_SUCCESS ：发送成功； TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开；TUYA_BLE_ERR_INVALID_LENGTH：数据长度错误； TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	1、 应用代码通过调用该函数透传数据到手机 App。2、 透传的数据格式是由设备和手机App端协商定义，Tuya BLE SDK不做解析。

tuya_ble_production_test_asynchronous_response

函数名	tuya_ble_production_test_asynchronous_response
函数原型	tuya_ble_status_t tuya_ble_production_test_asynchronous_response(uint8_t channel,uint8_t *p_data,uint32_t len)
功能概述	产测指令异步响应函数。
参数	Channel[in] : 传输通道，0-uart;1-ble； p_data [in] : 需要响应的完整指令数据； Len[in] : 数据长度。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开； TUYA_BLE_ERR_INVALID_LENGTH：数据长度错误； TUYA_BLE_ERR_NO_MEM：申请内存失败； TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	在进行产测时（产测授权是通过 uart ,整机测试通过 BLE）有些测试项是不能立即响应结果给上位机产测工具的，或者有些测试项需要应用程序来处理，这个时候就需要调用该函数将测试结果发送给上位机产测工具。

tuya_ble_net_config_response

函数名	tuya_ble_net_config_response
函数原型	tuya_ble_status_t tuya_ble_net_config_response(int16_t result_code)
功能概述	Wi-Fi配网响应函数。
参数	Result_code[in] : 配网状态；
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开；TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	适用于 Wi-Fi / BLE 多协议设备。

tuya_ble_ubound_response

函数名	tuya_ble_ubound_response
函数原型	tuya_ble_status_t tuya_ble_ubound_response(uint8_t result_code)
功能概述	多协议设备解绑响应函数。
参数	Result_code[in] : 状态;
返回值	TUYA_BLE_SUCCESS： 发送成功; TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开；TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	适用于 Wi-Fi / BLE 多协议设备。

tuya_ble_anomaly_ubound_response

函数名	tuya_ble_anomaly_ubound_response
函数原型	tuya_ble_status_t tuya_ble_anomaly_ubound_response (uint8_t result_code)
功能概述	多协议设备异常解绑响应函数。
参数	Result_code[in] : 状态;
返回值	TUYA_BLE_SUCCESS： 发送成功; TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开; TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	适用于 Wi-Fi / BLE 多协议设备。

tuya_ble_device_reset_response

函数名	tuya_ble_device_reset_response
函数原型	tuya_ble_status_t tuya_ble_device_reset_response (uint8_t result_code)
功能概述	多协议设备恢复出厂设置响应函数。
参数	Result_code[in] : 状态;
返回值	TUYA_BLE_SUCCESS： 发送成功; TUYA_BLE_ERR_INVALID_STATE：当前状态不支持发送，例如蓝牙断开；TUYA_BLE_ERR_NO_EVENT：其他错误。
备注	适用于 Wi-Fi / BLE 多协议设备。

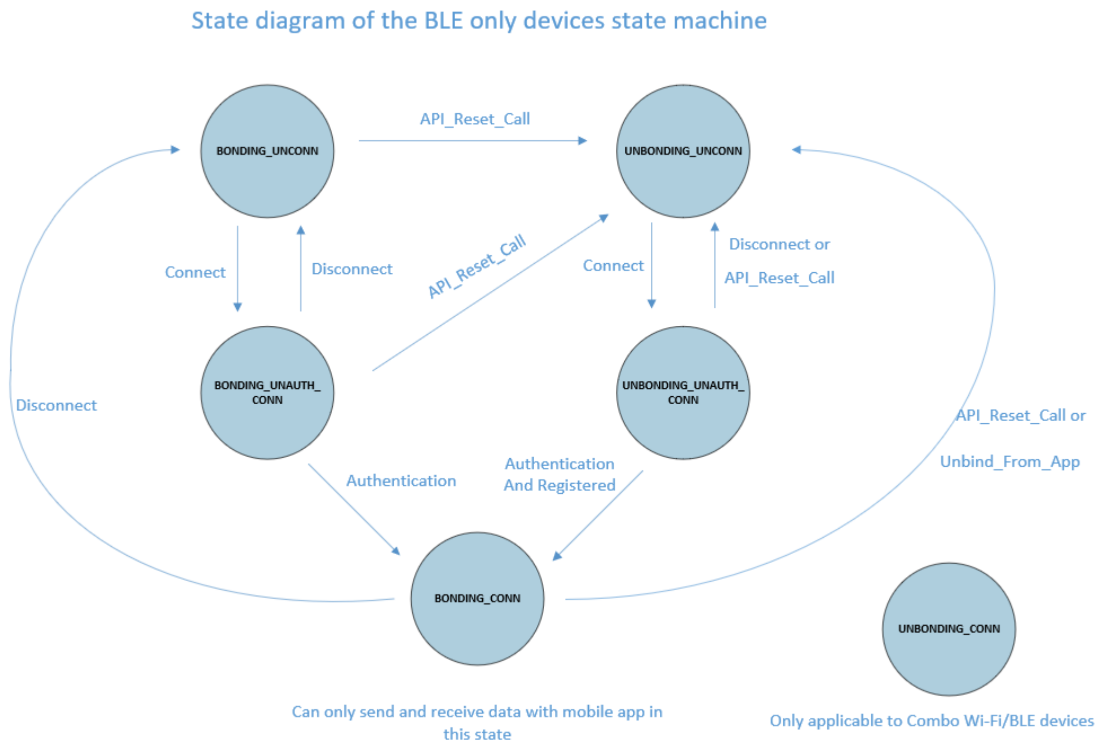
tuya_ble_connect_status_get

函数名	tuya_ble_connect_status_get
函数原型	tuya_ble_connect_status_t tuya_ble_connect_status_get(void)
功能概述	获取 BLE 连接状态。
参数	无。
返回值	当前连接状态
备注	

连接状态定义如下所示：

```
typedef enum
{
    UNBONDING_UNCONN = 0, //未绑定未连接
    UNBONDING_CONN,        //未绑定已连接已认证
    BONDING_UNCONN,        //已绑定未连接
    BONDING_CONN,          //已绑定已连接已认证
    BONDING_UNAUTH_CONN,   //已绑定已连接未认证
    UNBONDING_UNAUTH_CONN, //未绑定已连接未认证
    UNKNOW_STATUS
}tuya_ble_connect_status_t;
```

各状态流程如下所示：



tuya_ble_device_unbind

函数名	tuya_ble_device_unbind
函数原型	tuya_ble_status_t tuya_ble_device_unbind(void)
功能概述	设备主动解绑。
参数	无。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INTERNAL：其他错误。
备注	1、对于定义外部按键重置解绑功能的设备，按键触发重置解绑后应用程序需要调用该函数通知Tuya BLE SDK 清除相关信息，例如清除绑定信息。 2、该函数不会清除 Tuya BLE SDK 绑定信息中的设备虚拟ID （虚拟ID 管理着云端历史数据）。

tuya_ble_device_factory_reset

函数名	tuya_ble_device_factory_reset
函数原型	tuya_ble_status_t tuya_ble_device_factory_reset(void)
功能概述	设备重置。
参数	无。
返回值	TUYA_BLE_SUCCESS： 发送成功； TUYA_BLE_ERR_INTERNAL：其他错误。
备注	1、对于定义外部按键重置解绑功能的设备，按键触发重置解绑后应用程序需要调用该函数通知Tuya BLE SDK 清除相关信息，例如清除绑定信息。 2、该函数会清除 Tuya BLE SDK 绑定信息中的设备虚拟ID （虚拟ID 管理着云端历史数据）。

tuya_ble_time_req

函数名	tuya_ble_time_req
函数原型	tuya_ble_status_t tuya_ble_time_req(uint8_t time_type)
功能概述	请求云端时间。
参数	time_type[in] : 0 - 请求云端13 字节 ms 级字符串格式的时间; 1 - 请求云端年月日时分秒星期格式的时间。 2 - 请求手机本地的年月日时分秒星期格式的时间。
返回值	TUYA_BLE_SUCCESS ： 发送成功; TUYA_BLE_ERR_INVALID_PARAM： 参数错误; TUYA_BLE_ERR_INTERNAL：其他错误。
备注	13字节字符串时间格式："0000000123456" 表示123456ms时间戳; 常规时间格式：0x13,0x04,0x1C,0x0C,0x17,0x19,0x02 表示 2019 年 4 月 28 日 12:23:25 星期二; BLE SDK收到请求后会发送相应指令给 App，BLE SDK 收到 App 返回的时间后将会以消息的方式发送给设备应用程序。

tuya_ble_ota_response

函数名	tuya_ble_ota_response
函数原型	tuya_ble_status_t tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)
功能概述	OTA 响应指令。
参数	p_data[in] : OTA 响应数据。
返回值	TUYA_BLE_SUCCESS ： 发送成功; TUYA_BLE_ERR_INVALID_STATE：状态错误; TUYA_BLE_ERR_INVALID_LENGTH： 数据长度错误; TUYA_BLE_ERR_NO_MEM： 内存申请失败; TUYA_BLE_ERR_INTERNAL：其他错误。
备注	具体格式见 OTA 相关章节。

tuya_ble_custom_event_send

函数名	tuya_ble_custom_event_send
函数原型	uint8_t tuya_ble_custom_event_send(tuya_ble_custom_evt_t evt)
功能概述	发送应用自定义消息及回调给 BLE SDK。
参数	evt[in] : 自定义事件，定义如下所示。
返回值	0： 发送成功; 1： 失败。
备注	

```
typedef struct {
    int32_t evt_id;
    void *data;
    void (*custom_event_handler)(int32_t evt_id,void*data);
} tuya_ble_custom_evt_t;
```

说明：该函数主要用于客户应用程序想使用 Tuya BLE SDK 内部的消息调度器来处理应用程序的消息事件的场景。

tuya_ble_custom_event_send 应用示例：

```
#define APP_CUSTOM_EVENT_1  1
#define APP_CUSTOM_EVENT_2  2
#define APP_CUSTOM_EVENT_3  3
#define APP_CUSTOM_EVENT_4  4
#define APP_CUSTOM_EVENT_5  5

typedef struct {
    uint8_t data[50];
} custom_data_type_t;

void custom_data_process(int32_t evt_id,void *data)
{
    custom_data_type_t *event_1_data;
    TUYA_BLE_LOG_DEBUG("custom event id = %d",evt_id);
    switch (evt_id)
    {
        case APP_CUSTOM_EVENT_1:
            event_1_data = (custom_data_type_t *)data;
            TUYA_BLE_LOG_HEXDUMP_DEBUG("received APP_CUSTOM_EVENT_1 
data:",event_1_data->data,50);
            break;
        case APP_CUSTOM_EVENT_2:
            break;
        case APP_CUSTOM_EVENT_3:
            break;
        case APP_CUSTOM_EVENT_4:
            break;
        case APP_CUSTOM_EVENT_5:
            break;
        default:
```

```
              break;

        }
}

custom_data_type_t custom_data;

void custom_evt_1_send_test(uint8_t data)
{
    tuya_ble_custom_evt_t event;

    for(uint8_t i=0; i<50; i++)
    {
        custom_data.data[i] = data;
    }
    event.evt_id = APP_CUSTOM_EVENT_1;
    event.custom_event_handler = (void *)custom_data_process;
    event.data = &custom_data;
    tuya_ble_custom_event_send(event);
}
```

tuya_ble_callback_queue_register

函数名	tuya_ble_callback_queue_register
函数原型	原型1：tuya_ble_status_t tuya_ble_callback_queue_register(void *cb_queue); 原型2：tuya_ble_status_t tuya_ble_callback_queue_register(tuya_ble_callback_t cb);
功能概述	RTOS 架构下注册用于接收 BLE SDK 消息的消息队列 使用原型1； 无RTOS架构下注册用于BLE SDK 消息的回调函数 使用原型2。
参数	cb_queue [in] : 消息队列。 cb[in]： callback函数地址。
返回值	TUYA_BLE_ERR_RESOURCES： 注册失败； TUYA_BLE_SUCCESS： 注册成功。
备注	

RTOS 下应用示例：

```
void *tuya_custom_queue_handle;

os_msg_queue_create(&tuya_custom_queue_handle,
MAX_NUMBER_OF_TUYA_CUSTOM_MESSAGE, sizeof(tuya_ble_cb_evt_param_t));

tuya_ble_callback_queue_register(tuya_custom_queue_handle);
```

无RTOS 下应用示例：

```
void tuya_cb_handler(tuya_ble_cb_evt_param_t* event);

tuya_ble_callback_queue_register(tuya_cb_handler);
```

tuya_ble_event_response

函数名	tuya_ble_event_response
函数原型	tuya_ble_status_t tuya_ble_event_response(tuya_ble_cb_evt_param_t *param)
功能概述	RTOS 架构下用于响应 BLE SDK 的消息。
参数	param [in] : 消息指针。
返回值	TUYA_BLE_SUCCESS： 成功; 其他： 失败。
备注	RTOS架构下，应用代码处理完BLE SDK发送来的消息后，必须调用此函数给与BLE SDK反馈。

应用示例。

```
 /*处理ble sdk消息的应用task*/
void app_custom_task(void *p_param)
{
    tuya_ble_cb_evt_param_t event;

    while (true)
    {
        if (os_msg_recv(tuya_custom_queue_handle, &event, 0xFFFFFFFF) == true)
        {
            switch (event.evt)
            {
            case TUYA_BLE_CB_EVT_CONNECTE_STATUS:
                break;
            case TUYA_BLE_CB_EVT_DP_WRITE:
                break;
            case TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE:
                break;
            case TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE:
                break;
            case TUYA_BLE_CB_EVT_UNBOUND:
                break;
            case TUYA_BLE_CB_EVT_ANOMALY_UNBOUND:
                break;
            case TUYA_BLE_CB_EVT_DEVICE_RESET:
                break;
            case TUYA_BLE_CB_EVT_DP_QUERY:
                break;
            case TUYA_BLE_CB_EVT_OTA_DATA:
                break;
            case TUYA_BLE_CB_EVT_NETWORK_INFO:
                break;
            case TUYA_BLE_CB_EVT_WIFI_SSID:
                break;
```

```
            case TUYA_BLE_CB_EVT_TIME_STAMP:
                break;
            case TUYA_BLE_CB_EVT_TIME_NORMAL:
                break;
            case TUYA_BLE_CB_EVT_DATA_PASSTHROUGH:
                break;
            default:
                break;
            }

            tuya_ble_event_response(&event);
        }
    }

}
```

tuya_ble_scheduler_queue_size_get

函数名	tuya_ble_scheduler_queue_size_get
函数原型	uint16_t tuya_ble_scheduler_queue_size_get(void)
功能概述	无OS架构下获取事件消息队列的大小
参数	无
返回值	消息队列的大小
备注	

tuya_ble_scheduler_queue_space_get

函数名	tuya_ble_scheduler_queue_space_get
函数原型	uint16_t tuya_ble_scheduler_queue_space_get(void)
功能概述	无OS架构下获取事件消息队列可用空间大小
参数	无
返回值	消息队列的可用空间大小
备注	

tuya_ble_scheduler_queue_events_get

函数名	tuya_ble_scheduler_queue_events_get
函数原型	uint16_t tuya_ble_scheduler_queue_events_get(void)
功能概述	无OS架构下获取事件消息队列的未处理事件个数
参数	无
返回值	
备注	

tuya_ble_sleep_allowed_check

函数名	tuya_ble_sleep_allowed_check
函数原型	bool tuya_ble_sleep_allowed_check(void)
功能概述	查询Tuya BLE SDK是否允许休眠
参数	无
返回值	true : 允许休眠；false : 不允许休眠
备注	应用程序在进入休眠前必须调用该函数查询sdk是否允许休眠，否则可能会引起错误。

CALL BACK EVENT介绍

TUYA BLE SDK通过message（RTOS架构下）或者设备App注册的call back函数（无RTOS架构下）向设备应用程序发送消息（状态、数据等），如下所示是RTOS架构下设备应用代码处理BLE SDK消息的典型架构，同理无RTOS架构下的芯片平台也可以使用所示的代码架构来处理消息，只是基于call back函数处理消息后不需要调用 `tuya_ble_event_response()` 响应BLE SDK，本章节主要介绍各种 EVENT 的含义。

注册 call back 函数示例：

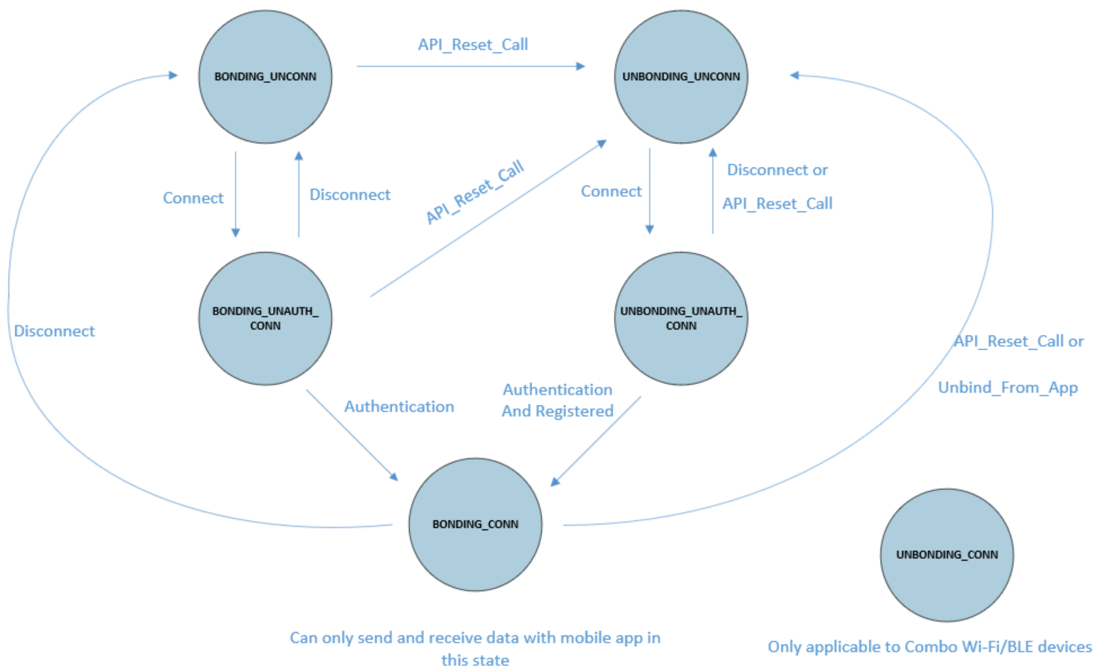
```
/*处理ble sdk消息的call back函数*/
static void tuya_cb_handler(tuya_ble_cb_evt_param_t* event)
{
        switch (event->evt)
        {
        case TUYA_BLE_CB_EVT_CONNECTE_STATUS:
            break;
        case TUYA_BLE_CB_EVT_DP_WRITE:
            break;
        case TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE:
            break;
        case TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE:
            break;
        case TUYA_BLE_CB_EVT_UNBOUND:
```

```
                break;
            case TUYA_BLE_CB_EVT_ANOMALY_UNBOUND:
                break;
            case TUYA_BLE_CB_EVT_DEVICE_RESET:
                break;
            case TUYA_BLE_CB_EVT_DP_QUERY:
                break;
            case TUYA_BLE_CB_EVT_OTA_DATA:
                break;
            case TUYA_BLE_CB_EVT_NETWORK_INFO:
                break;
            case TUYA_BLE_CB_EVT_WIFI_SSID:
                break;
            case TUYA_BLE_CB_EVT_TIME_STAMP:
                break;
            case TUYA_BLE_CB_EVT_TIME_NORMAL:
                break;
            case TUYA_BLE_CB_EVT_DATA_PASSTHROUGH:
                break;
            default:
                break;
            }

    }

void tuya_ble_app_init(void)
{
    device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
    device_param.product_id_len = 8;
    memcpy(device_param.product_id,APP_PRODUCT_ID,8);
    device_param.firmware_version = TY_APP_VER_NUM;
    device_param.hardware_version = TY_HARD_VER_NUM;
    device_param.adv_local_name_len = strlen(device_local_name);

memcpy(device_param.adv_local_name,device_local_name,device_param.adv_local_name
_len);

    device_param.use_ext_license_key = 1; //If use the license stored by the
SDK,initialized to 0, Otherwise 1.
    device_param.device_id_len = 16;

    if(device_param.use_ext_license_key==1)
    {
        memcpy(device_param.auth_key,(void *)auth_key_test,AUTH_KEY_LEN);
        memcpy(device_param.device_id,(void *)device_id_test,DEVICE_ID_LEN);
        memcpy(device_param.mac_addr_string,mac_test,12);
        device_param.mac_addr.addr_type = TUYA_BLE_ADDRESS_TYPE_RANDOM;
    }

    tuya_ble_sdk_init(&device_param);
    tuya_ble_callback_queue_register(tuya_cb_handler);

    tuya_ota_init();

}
```

TUYA_BLE_CB_EVT_CONNECTE_STATUS

Event	TUYA_BLE_CB_EVT_CONNECTE_STATUS
对应数据结构	如下所示。
描述	BLE SDK 每次状态的改变都会发送该 event 给设备应用程序。
备注	

```
typedef enum {
    UNBONDING_UNCONN = 0,
    UNBONDING_CONN,
    BONDING_UNCONN,
    BONDING_CONN,
    BONDING_UNAUTH_CONN,
    UNBONDING_UNAUTH_CONN,
    UNKNOW_STATUS
} tuya_ble_connect_status_t;
```



某些场景下同一个状态可能会发送两次，所以应用程序需要按照状态的改变来触发某个功能时，就需要比较上一次收到的状态是否一样，建议采用如下所示的处理方法：

```
static tuya_ble_connect_status_t current_connect_status = UNKNOW_STATUS;
static tuya_ble_connect_status_t last_connect_status = UNKNOW_STATUS;

static void tuya_cb_handler(tuya_ble_cb_evt_param_t* event)
{
    int16_t result = 0;
    tuya_ble_status_t err_code;
    switch (event->evt)
```

```
    {
    case TUYA_BLE_CB_EVT_CONNECTE_STATUS:

        TUYA_APP_LOG_INFO("received tuya ble conncet status update event,current
connect status = %d",event->connect_status);
        if((event->connect_status == BONDING_CONN)&&
(last_connect_status!=BONDING_CONN))
        {
            //Do something
        }
        last_connect_status = event->connect_status;

        break;
     default:
        break;
    }
}
```

TUYA_BLE_CB_EVT_DP_WRITE

Event	TUYA_BLE_CB_EVT_DP_WRITE
对应数据结构	
描述	BLE SDK 收到的手机 App 发送的 dp 点数据。
备注	deprecated 弃用

TUYA_BLE_CB_EVT_DP_QUERY

Event	TUYA_BLE_CB_EVT_DP_QUERY
对应数据结构	tuya_ble_dp_query_data_t
描述	BLE SDK 收到的手机 App 发送的要查询的 dp id数组。
备注	data_len=0 表示查询所有的 dp 点，否则 p_data 指向的每一个字节代表要查询的一个 dp 点，例如 data_len=3，p_data指向内容为 {0x01,0x02,0x03}，表示要查询 dp_id=1，dp_id=2,dp_id=3 的 3 个 dp 数据。

```
typedef struct {
    uint8_t *p_data;
    uint16_t data_len;
} tuya_ble_dp_query_data_t;
```

TUYA_BLE_CB_EVT_DP_DATA_RECEIVED

Event	TUYA_BLE_CB_EVT_DP_DATA_RECEIVED
对应数据结构	tuya_ble_dp_data_received_data_t;
描述	BLE SDK 收到的手机 App 发送的 dp 数据。
备注	

数据结构：

```
typedef struct {
    uint32_t sn;
    uint8_t *p_data;
    uint16_t data_len;
} tuya_ble_dp_data_received_data_t;
```

sn： 手机app定义的发送序号，逐次累加，设备应用程序按需使用。

p_data： dp数据指针，指向的数据内容格式如下所示：

Dp1的数据	~				Dpn的数据			
1	2	3 - 4	5~	~	n	n+1	n+2 - n+3	n+4~
Dp_id	Dp_type	Dp_len	Dp_data	~	Dp_id	Dp_type	Dp_len	Dp_data

详见 `tuya_ble_dp_data_send()` api介绍。

data_len： 上面介绍的dp数据长度。

TUYA_BLE_CB_EVT_OTA_DATA

Event	TUYA_BLE_CB_EVT_OTA_DATA
对应数据结构	tuya_ble_ota_data_t
描述	BLE SDK 收到的手机 App 发送的 OTA 固件数据。
备注	具体见 OTA 章节介绍。

数据结构：

```
typedef struct {
    tuya_ble_ota_data_type_t type;
    uint16_t data_len;
    uint8_t *p_data;
} tuya_ble_ota_data_t;
```

TUYA_BLE_CB_EVT_BULK_DATA

Event	TUYA_BLE_CB_EVT_BULK_DATA
对应数据结构	tuya_ble_bulk_data_request_t
描述	大数据传输通道
备注	具体见 大数据 BULK data 章节介绍。

数据结构：

```
typedef struct {
    tuya_ble_bulk_data_evt_type_t evt;
    uint8_t bulk_type;
    union
    {
        tuya_ble_bulk_data_evt_read_block_req_t block_data_req_data;
        tuya_ble_bulk_data_evt_send_data_req_t  send_data_req_data;
    } params;
} tuya_ble_bulk_data_request_t;
```

TUYA_BLE_CB_EVT_NETWORK_INFO

Event	TUYA_BLE_CB_EVT_NETWORK_INFO
对应数据结构	tuya_ble_network_data_t
描述	BLE SDK 收到的手机 App 发送的 Wi-Fi 配网信息，例如： 字符串 " {"wifi_ssid":"tuya","password":"12345678","token":"xxxxxxxxxx"} "
备注	只适用于 Wi-Fi / BLE 双协议设备。

数据结构：

```
typedef struct{
uint16_t  data_len; //include '\0'
uint8_t  *p_data;
}tuya_ble_network_data_t;
```

TUYA_BLE_CB_EVT_WIFI_SSID

Event	TUYA_BLE_CB_EVT_WIFI_SSID
对应数据结构	tuya_ble_wifi_ssid_data_t
描述	BLE SDK 收到的手机 App 发送的 Wi-Fi 配网信息，例如：字符串 "{"wifi_ssid":"tuya","password":"12345678"}"
备注	只适用于 Wi-Fi / BLE 双协议设备，和 TUYA_BLE_CB_EVT_NETWORK_INFO 相比缺少 token 字段，主要已配网设备的 Wi-Fi SSID 更新。

数据结构：

```
typedef struct{
uint16_t  data_len;//include '\0'
uint8_t  *p_data;
}tuya_ble_wifi_ssid_data_t;
```

TUYA_BLE_CB_EVT_TIME_STAMP

Event	TUYA_BLE_CB_EVT_TIME_STAMP
对应数据结构	tuya_ble_timestamp_data_t
描述	BLE SDK 收到的手机 App 发送的字符串格式的时间戳，例如 "0000000123456" 表示 123456 ms，ms 级 unix 时间戳。
备注	time_zone 时区为实际时区的 100 倍，例如 -8 区为 -800。

数据结构：

```
typedef struct{
uint8_t  timestamp_string[14];
int16_t time_zone;  //actual time zone Multiply by 100.
}tuya_ble_timestamp_data_t;
```

timestamp_string： 13位毫秒级字符串（加上结束符总共14字节），例如 "0000000123456" 表示 123456 ms。

time_zone： 实际时区的100倍，如果该值为-800，那么实际时区是 -8 区。

设备每次和涂鸦app 连接后，app都会同步一次时间，Tuya BLE SDK 收到 app 发送的时间后便会发送该 event 给设备应用程序，时间戳数据来自云端，时区数据是手机本地时区。

设备应用程序通过调用 `tuya_ble_time_req(0)` 请求云端时间后，收到该 event，其中的时间数据来自云端，时区数据来自手机本地时区。

设备应用程序通过调用 `tuya_ble_time_req(1)` 和 `tuya_ble_time_req(2)` 不会收到该 event，而是会收到下面介绍的 `TUYA_BLE_CB_EVT_TIME_NORMAL` 和 `TUYA_BLE_CB_EVT_APP_LOCAL_TIME_NORMAL` event.

TUYA_BLE_CB_EVT_TIME_NORMAL

Event	TUYA_BLE_CB_EVT_TIME_NORMAL
对应数据结构	tuya_ble_time_noraml_data_t
描述	BLE SDK 收到的手机 App 发送的常规格式的时间，如 正 8 区，2019 年 4 月 28 日 12:23:25 星期二， 对应数据：0x13 ,0x04 ,0x1C ,0x0C ,0x17 ,0x19 ,0x02 （星期），0x0320(时区time_zone)。
备注	

数据结构：

```
typedef struct {
    uint16_t nYear; // Actual year minus 2000
    uint8_t nMonth;
    uint8_t nDay;
    uint8_t nHour;
    uint8_t nMin;
    uint8_t nSec;
    uint8_t DayIndex; /* 0 = Sunday */
    int16_t time_zone;   //actual time zone Multiply by 100.
} tuya_ble_time_noraml_data_t;
```

设备应用程序通过调用 `tuya_ble_time_req(1)` 将会收到该 event，其中 `tuya_ble_time_req(1)` 收到的时间数据来自云端，时区数据来自手机本地时区。

TUYA_BLE_CB_EVT_APP_LOCAL_TIME_NORMAL

Event	TUYA_BLE_CB_EVT_APP_LOCAL_TIME_NORMAL
对应数据结构	tuya_ble_time_noraml_data_t
描述	BLE SDK 收到的手机 App 发送的常规格式的时间，如 正 8 区，2019 年 4 月 28 日 12:23:25 星期二， 对应数据：0x13 ,0x04 ,0x1C ,0x0C ,0x17 ,0x19 ,0x02 （星期），0x0320(时区time_zone)。
备注	

数据结构：

```
typedef struct {
    uint16_t nYear; // Actual year minus 2000
    uint8_t nMonth;
    uint8_t nDay;
    uint8_t nHour;
    uint8_t nMin;
    uint8_t nSec;
    uint8_t DayIndex; /* 0 = Sunday */
    int16_t time_zone;   //actual time zone Multiply by 100.
} tuya_ble_time_noraml_data_t;
```

设备应用程序通过调用 `tuya_ble_time_req(2)` 将会收到该 event， 其中收到的时间数据和时区数据都来自手机本地。

TUYA_BLE_CB_EVT_TIME_STAMP_WITH_DST

Event	TUYA_BLE_CB_EVT_TIME_STAMP_WITH_DST
对应数据结构	
描述	
备注	暂未使用

TUYA_BLE_CB_EVT_DATA_PASSTHROUGH

Event	TUYA_BLE_CB_EVT_DATA_PASSTHROUGH
对应数据结构	tuya_ble_passthrough_data_t
描述	BLE SDK 收到的手机 App 发送的透传数据。
备注	BLE SDK 不解析透传的数据，数据格式由设备应用程序和手机 App 协商定义。

数据结构：

```
typedef struct{
    uint16_t  data_len;
    uint8_t  *p_data;
}tuya_ble_passthrough_data_t;
```

透传通道主要用于收发 设备应用程序 和 手机app 面板共同定义的协议数据，Tuya BLE SDK不做任何解析，其中App发送的透传数据通过 `TUYA_BLE_CB_EVT_DATA_PASSTHROUGH` event 推送给设备应用，设备应用程序通过调用`tuya_ble_data_passthrough()` 发送数据给 手机 App 。

TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE
对应数据结构	
描述	
备注	deprecated 弃用

TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE
对应数据结构	
描述	
备注	deprecated 弃用

TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_REPORT_RESPONSE
对应数据结构	
描述	
备注	deprecated 弃用

TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_AND_TIME_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_AND_TIME_REPORT_RESPONSE
对应数据结构	
描述	
备注	deprecated 弃用

TUYA_BLE_CB_EVT_DP_DATA_SEND_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_SEND_RESPONSE
对应数据结构	tuya_ble_dp_data_send_response_data_t
描述	tuya_ble_dp_data_send() 的响应 event
备注	

数据结构：

```
typedef struct {
    uint32_t sn;
    tuya_ble_dp_data_send_type_t type;
    tuya_ble_dp_data_send_mode_t mode;
    tuya_ble_dp_data_send_ack_t ack;
    uint8_t status;  // 0 - succeed, 1- failed.
} tuya_ble_dp_data_send_response_data_t;
```

设备应用程序通过调用 `tuya_ble_dp_data_send()` 发送dp数据给手机App后，如果携带的 ack 参数是 `DP_SEND_WITH_RESPONSE` ，那么设备手机App收到dp数据后，就会发送响应给Tuya BLE SDK, BLE SDK收到手机App的响应后，就会发送该 event 给设备应用程序。

其中 sn 、type 、mode 、ack 和 `tuya_ble_dp_data_send()` 发送dp 数据是携带参数一致。

status 为0 表示App成功收到了dp数据。

如果设备应用程序连续调用多次 `tuya_ble_dp_data_send()` 函数发送多次dp数据，那么设备应用程序可以根据该 event 数据中的 sn 和 status 来判断哪一次是发送成功的，哪一次是发送失败的。

TUYA_BLE_CB_EVT_DP_DATA_WITH_TIME_SEND_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_WITH_TIME_SEND_RESPONSE
对应数据结构	tuya_ble_dp_data_with_time_send_response_data_t
描述	tuya_ble_dp_data_with_time_send() 的响应 event
备注	

数据结构：

```
typedef struct {
    uint32_t sn;
    tuya_ble_dp_data_send_type_t type;
    tuya_ble_dp_data_send_mode_t mode;
    tuya_ble_dp_data_send_ack_t ack;
    uint8_t status; // 0 - succeed, 1- failed.
} tuya_ble_dp_data_with_time_send_response_data_t;
```

设备应用程序通过调用 `tuya_ble_dp_data_with_time_send()` 发送dp数据给手机App后，那么设备手机App收到dp数据后，就会发送响应给Tuya BLE SDK, BLE SDK收到手机App的响应后，就会发送该 event 给设备应用程序。

其中 sn 、type 、mode 、ack 和 `tuya_ble_dp_data_with_time_send()` 发送dp 数据是携带参数一致。

status 为0 表示App成功收到了dp数据。

如果设备应用程序连续调用多次 `tuya_ble_dp_data_with_time_send()` 函数发送多次dp数据，那么设备应用程序可以根据该 event 数据中的 sn 和 status 来判断哪一次是发送成功的，哪一次是发送失败的。

TUYA_BLE_CB_EVT_UNBOUND

Event	TUYA_BLE_CB_EVT_UNBOUND
对应数据结构	tuya_ble_unbound_data_t
描述	收到该 EVENT 表示手机 App 发送了解绑指令，其中 data 字段是保留字段，暂不使用。
备注	

数据结构：

```
typedef struct {
    uint8_t data;
} tuya_ble_unbound_data_t;
```

当手机App执行 “解除绑定” 操作后，BLE SDK 便会发送该 event ， 其中 data 为保留字段。

TUYA_BLE_CB_EVT_ANOMALY_UNBOUND

Event	TUYA_BLE_CB_EVT_ANOMALY_UNBOUND
对应数据结构	tuya_ble_anomaly_unbound_data_t
描述	收到该 EVENT 表示手机 App 发送了异常解绑指令，其中 data 字段是保留字段，暂不使用。
备注	

数据结构：

```
typedef struct {
    uint8_t data;
} tuya_ble_anomaly_unbound_data_t;
```

当手机App执行 “离线解绑” 操作后，BLE SDK 便会发送该 event ， 其中 data 为保留字段。

离线解绑 ：

当蓝牙设备和手机 App 没有保持蓝牙连接的场景下，手机 App 端执行了移除设备操作，由于是离线时移除，那么此时并不能通知到蓝牙设备，手机 App 离线移除完设备后，云端也就没有该设备的绑定信息。之后任何一个手机上的涂鸦智能 App 在发现了该设备后，便会发送离线解绑指令给设备，设备上的 Tuya BLE SDK 也就会发送该 event 。

TUYA_BLE_CB_EVT_DEVICE_RESET

Event	TUYA_BLE_CB_EVT_DEVICE_RESET
对应数据结构	tuya_ble_device_reset_data_t
描述	收到该 EVENT 表示手机 App 发送了重置指令，其中 data 字段是保留字段，暂不使用。
备注	设备应用程序收到重置 EVENT 后，需要执行重置功能定义的一些操作。

数据结构：

```
typedef struct {
    uint8_t data;
} tuya_ble_device_reset_data_t;
```

当手机App执行 “解绑并清除数据” 操作后，BLE SDK 便会发送该 event ， 其中 data 为保留字段。

TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID

Event	TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID
对应数据结构	tuya_ble_login_key_vid_data_t
描述	设备注册绑定成功后，手机 App 会发送设备的 login key 和虚拟 ID 给设备。
备注	主要用于多协议设备，单 BLE 设备无需处理该信息。

数据结构：

```
typedef struct {
    uint8_t login_key_len;
    uint8_t vid_len;
    uint8_t beacon_key_len;
    uint8_t login_key[LOGIN_KEY_LEN];
    uint8_t vid[DEVICE_VIRTUAL_ID_LEN];
    uint8_t beacon_key[BEACON_KEY_LEN];
} tuya_ble_login_key_vid_data_t;
```

TUYA_BLE_CB_EVT_UNBIND_RESET_RESPONSE

Event	TUYA_BLE_CB_EVT_UNBIND_RESET_RESPONSE
对应数据结构	tuya_ble_unbind_reset_response_data_t
描述	设备本地解绑和重置响应 event
备注	

数据结构：

```
typedef enum {
    RESET_TYPE_UNBIND,
    RESET_TYPE_FACTORY_RESET,
} tuya_ble_reset_type_t;

typedef struct {
    tuya_ble_reset_type_t type;
    uint8_t status;     //0-succeed,1-failed.
} tuya_ble_unbind_reset_response_data_t;
```

设备应用程序不仅可以通过手机 App 来解绑设备，也可以通过调用 Tuya BLE SDK提供的 `tuya_ble_device_unbind()` 和 `tuya_ble_device_factory_reset()` API 来执行本地解绑和重置操作，其中解绑并不会清除BLE SDK存储的设备id信息，而重置则会清除 BLE SDK存储的设备id信息，设备id 主要用于查找云端历史数据，也就是说一旦执行了重置操作，即使重新绑定后也不会回复历史数据，而解绑后再被绑定时可以恢复历史数据的。

由于 `tuya_ble_device_unbind()` 和 `tuya_ble_device_factory_reset()` 是异步API，调用后并不会立即执行对应操作，而是发送了对应的消息给 Tuya BLE SDK, BLE SDK执行完后便会发送该 event 给设备应用程序告知执行结果，所以设备应用程序调用完这两个 API 后 不能阻塞时延时，也不能立即重启，否则可能会执行失败。

TUYA_BLE_CB_EVT_WEATHER_DATA_REQ_RESPONSE

Event	TUYA_BLE_CB_EVT_WEATHER_DATA_REQ_RESPONSE
对应数据结构	tuya_ble_weather_data_req_response_t
描述	天气服务
备注	详见天气服务章节

数据结构：

```
typedef struct{
    uint8_t status;
}tuya_ble_weather_data_req_response_t;
```

TUYA_BLE_CB_EVT_WEATHER_DATA_RECEIVED

Event	TUYA_BLE_CB_EVT_WEATHER_DATA_RECEIVED
对应数据结构	tuya_ble_weather_data_received_data_t
描述	天气服务
备注	详见天气服务章节

数据结构：

```
typedef struct{
    uint16_t object_count;   /**< weather data object counts. */
    uint8_t location;        /**< location. */
    uint8_t *p_data;         /**< weather data. */
    uint16_t data_len;       /**< weather data length. */
}tuya_ble_weather_data_received_data_t;
```

移植示例

本章节以 nrf52832 为例，介绍无 RTOS 架构下的移植步骤，其他平台类似， nrf52832完整示例 demo 及其他平台的示例 demo 请联系 Tuya 项目对接人获取。

nrf52832移植示例

1、 下载 nrf52832 芯片原厂 SDK（以 nRF5_SDK_15.2.0_9412b96 为例说明），并准备一个 nrf52832 开发板。

2、 解压下载好的原厂 SDK 至某个自定义目录，如图 7-1 所示。

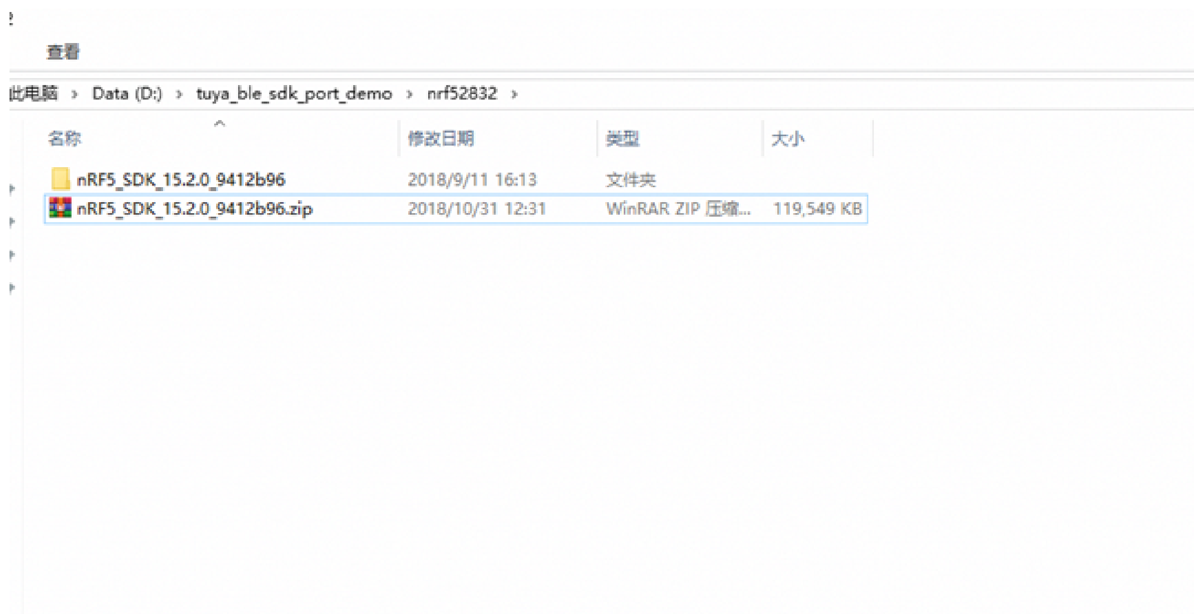


图7- 1 nrf52832 移植示例图 1

3、 依次进入 examples->ble_peripheral 目录，如图 7-2 所示。

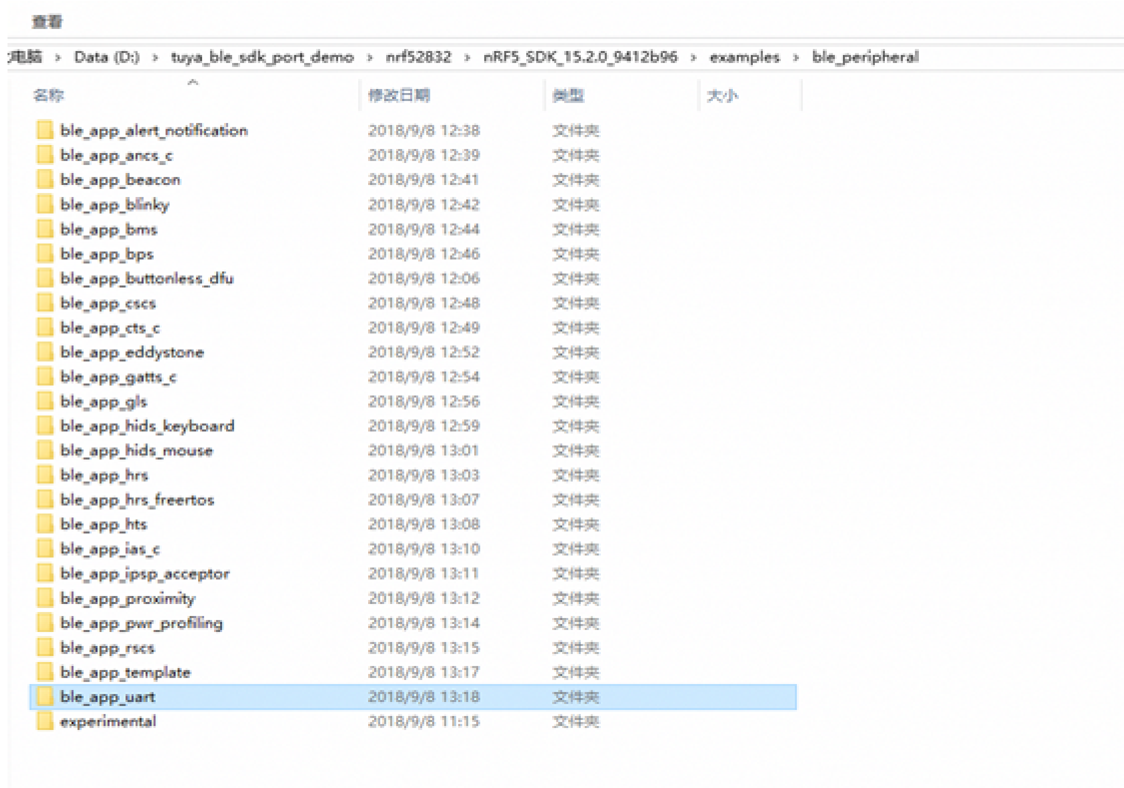


图7- 2 nrf52832 移植示例图 2

4、 该目录下为 ble peripheral 的各种 demo 例程，我们以 ble_app_uart 为模板新建一个项目，拷贝 ble_app_uart 目录并命名为 tuya_ble_standard_nordic（也可以命名为其他名字），如图7-3所示。

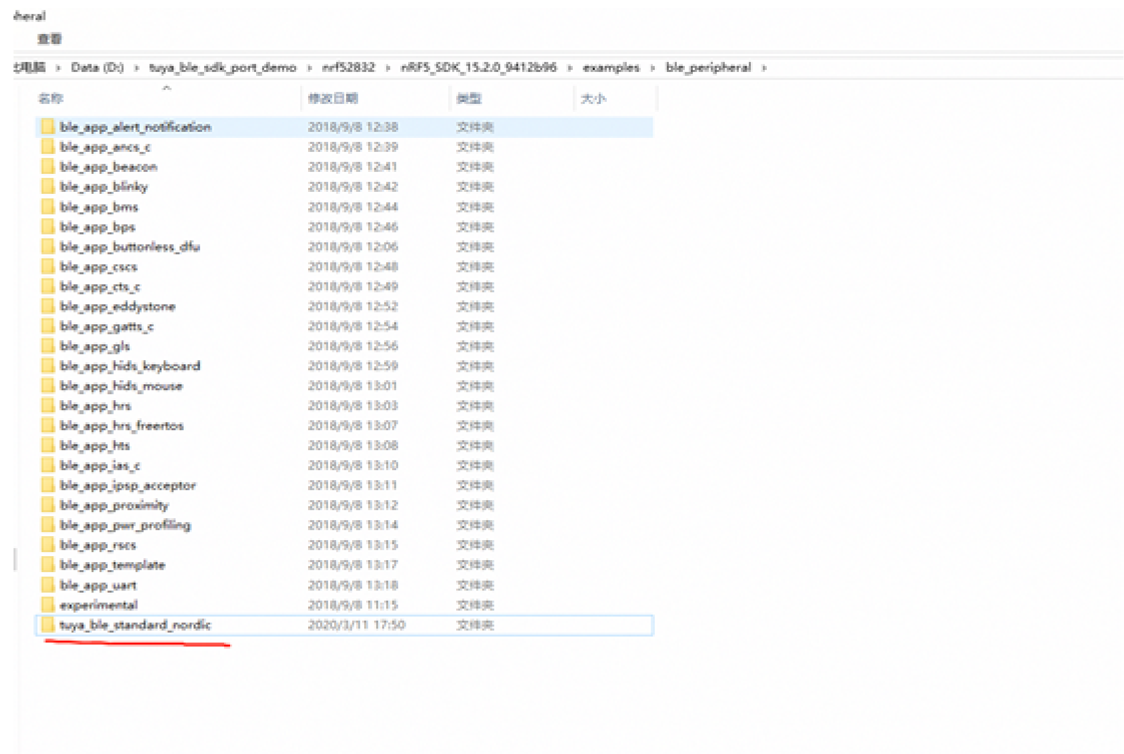


图7- 3 nrf52832 移植示例图 3

5、 打开工程（目录PCA10040-s132）并编译，先确认能编译通过并能在开发板上正确运行。

6、 以 nRF_BLE_Services->ble_nus.c 为模板新建 tuya_ble_service.c 文件并修改代码实现要求的 tuya ble service，修改 main.c 文件中广播相关的代码，按照之前介绍规定的广播内容广播。

7、 编译并下载到开发板运行，用手机 BLE 扫描 App（例如 iOS下的 lightBlue）扫描设备，扫描到后检查广播内容和 service 是否满足相关要求。

8、 下载 Tuya BLE SDK 并放到新建项目目录，如图 7-4 所示，并将相关源文件添加到工程中编译一次。

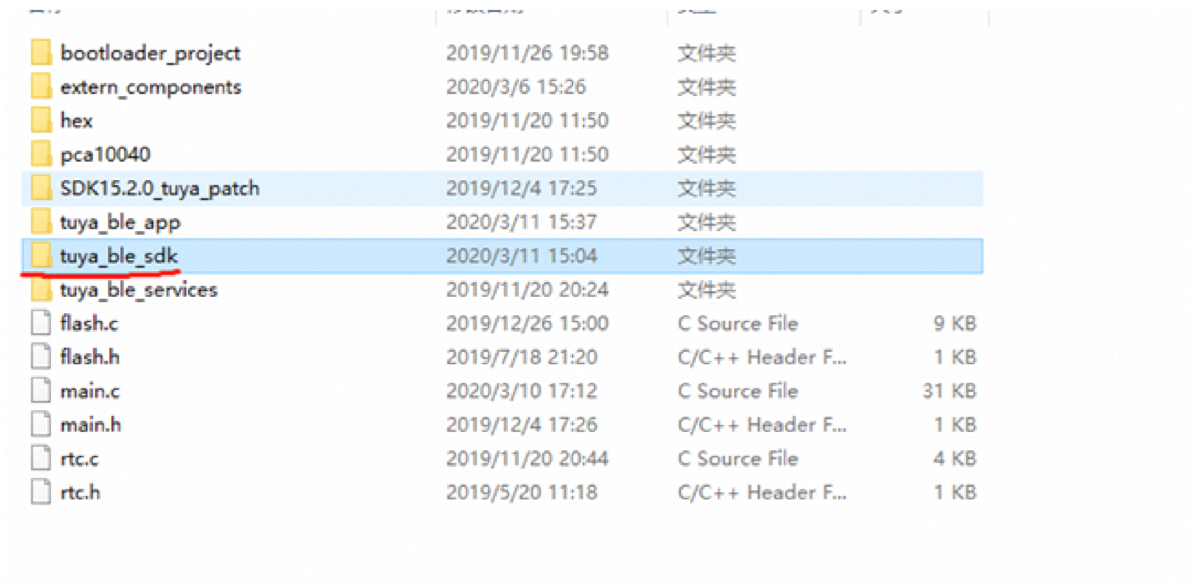


图7- 4 nrf52832 移植示例图 4

9、 添加好的工程目录如图 7-5 所示，注意选择正确的库文件。

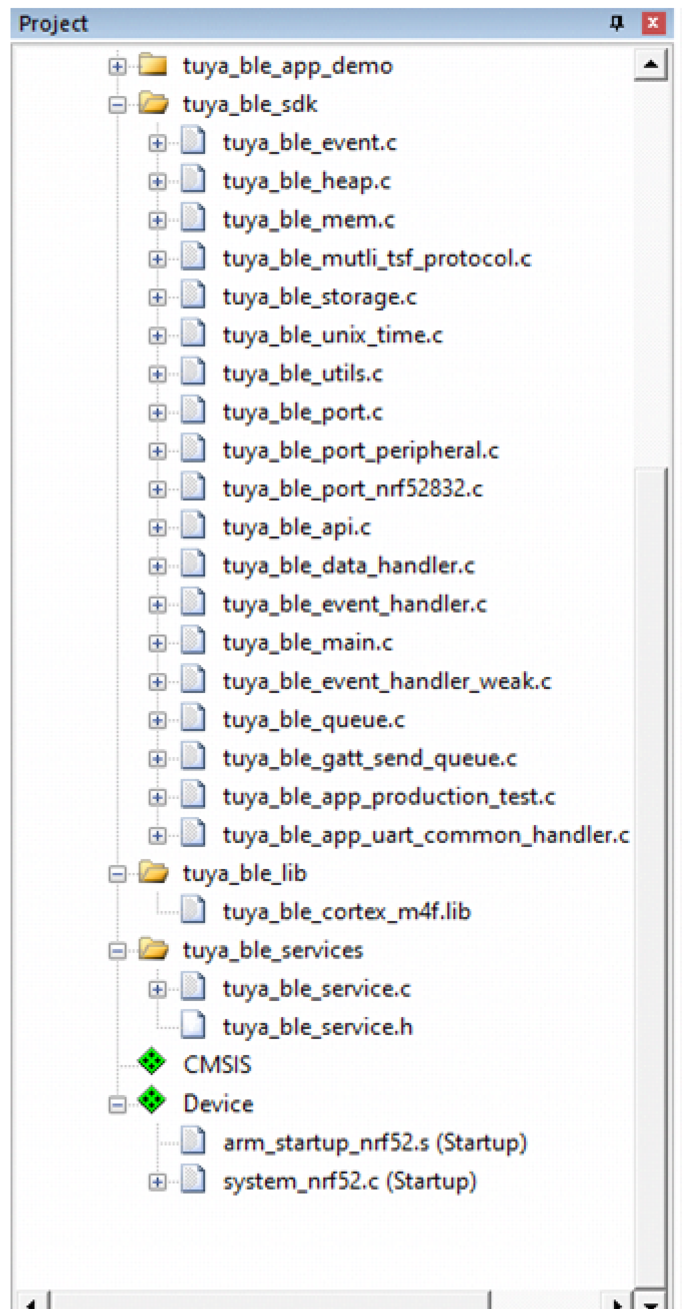


图7- 5 nrf52832 移植示例图 5

10、新建一个 custom_tuya_ble_config.h 文件，并放到工程目录中，本示例放在了 tuya_ble_app 目录下，custom_tuya_ble_config.h 配置项根据实际需求和环境配置，tuya ble sdk提供了一些芯片平台的参考，参考配置文件放在 tuya ble sdk 文件夹里 port 目录下的各平台目录下。

11、将 custom_tuya_ble_config.h 文件名字赋值给 CUSTOMIZED_TUYA_BLE_CONFIG_FILE，并添加到工程的宏定义中，如图 7-6 所示。

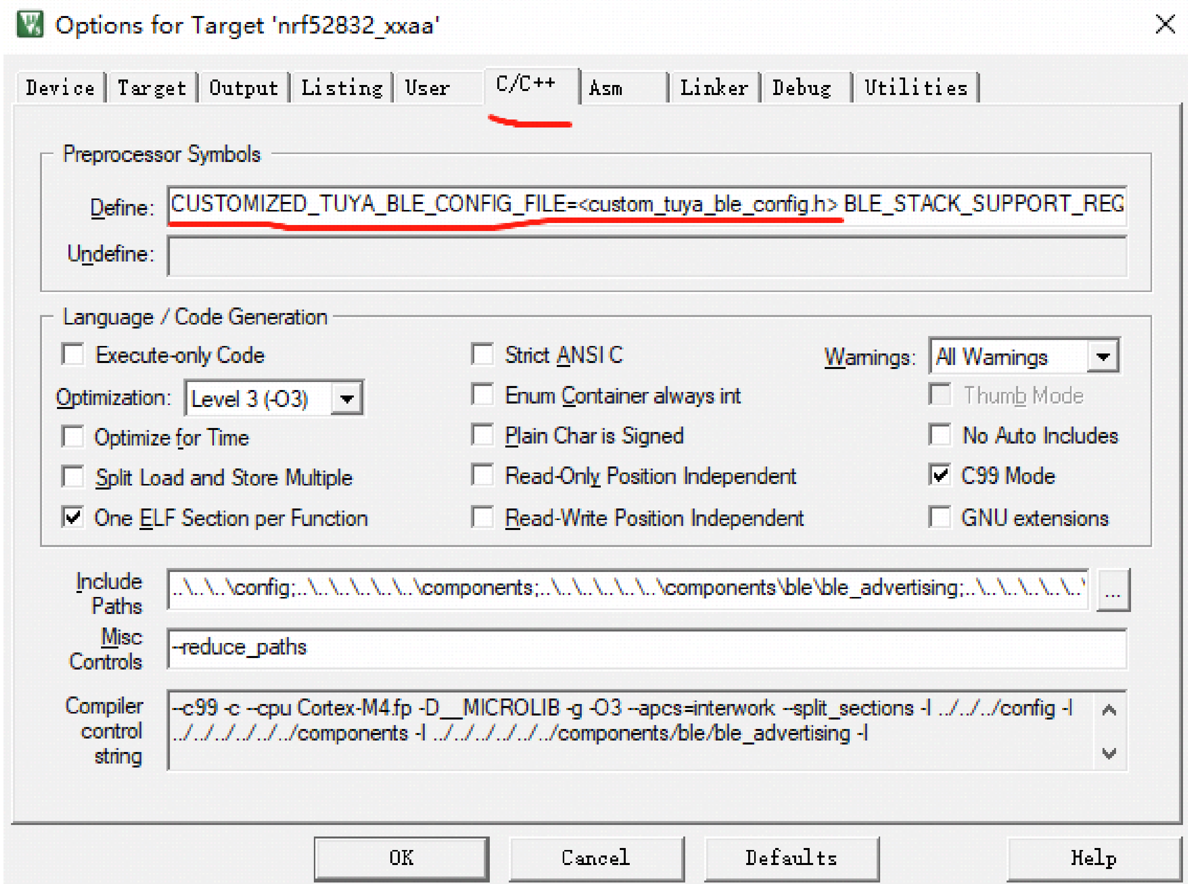


图7- 6 nrf52832 移植示例图 6

12、新建平台 port 文件，例如命名为 tuya_ble_port_nrf52832.h 和 tuya_ble_port_nrf52832.c，在新建的 port 文件中按照配置实现 tuya_ble_port.h 中所列的接口（并不需要实现全部的接口，例如本示例 demo 没有使用 RTOS，所以不需要实现 os 相关接口，本示例配置为使用 Tuya BLE SDK 的内部内存管理模块，所以也不需要实现内存分配和释放接口），tuya ble sdk 里 port 文件下的各平台名字命名的文件夹里有对应平台的参考移植实现文件。

13、编写完 port 文件后，在 custom_tuya_ble_config.h 配置文件中添加相应的 port 文件定义，如图 7-7 所示。

```
24
25
26 #ifndef CUSTOM_TUYA_BLE_CONFIG_H__
27 #define CUSTOM_TUYA_BLE_CONFIG_H__
28
29 #include "tuya_ble_type.h"
30
31
32
33
34 #define TUYA_BLE_PORT_PLATFORM_HEADER_FILE   "tuya_ble_port_nrf52832.h"
35
36
37 #define CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE "custom_app_product_test.h"
38
39
40 #define CUSTOMIZED_TUYA_BLE_APP_UART_COMMON_HEADER_FILE "custom_app_uart_common_handler.h"
41
42
43 #define TUYA_BLE_USE_OS 0
44
45
```

图7- 7 nrf52832 移植示例图 7

14、编译一次，如果编译不过，先检查代码优化错误。

15、接下来是 BLE SDK 的初始化，本示例专门新建了一个文件用于处理 SDK 的初始化、注册回调函数以及处理 SDK 的回调消息等，如图 7-8 所示。

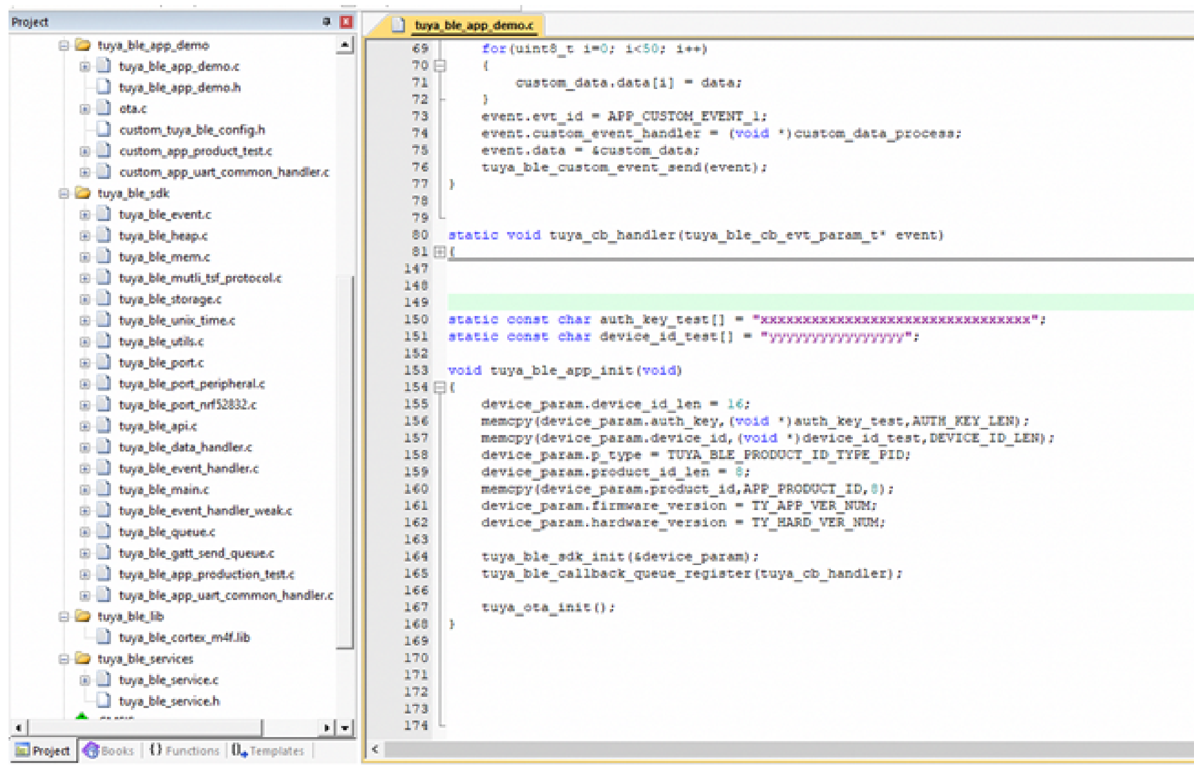


图7- 8 nrf52832 移植示例图 8

16、在 Tuya IoT 开发者平台注册产品并将生成的 product id 拷贝至工程代码中，如图 7-9 所示，其中 APP_PRODUCT_ID、APP_BUILD_FIRMNAME、TY_APP_VER_NUM、TY_APP_VER_STR、TY_HARD_VER_NUM、TY_HARD_VER_STR 宏定义名字不可改变，“xxxxxxxx” 替换为 IoT 平台注册的产品 id，"tuya_ble_sdk_app_demo_nrf52832" 替换为在涂鸦后台创建的项目名字（如果需要使用 Tuya 的产测工具授权，必须联系 Tuya 负责对接的同事在 Tuya 后台创建项目）。

tuya_ble_app_demo.h | tuya_ble_app_demo.c

```c
1  #ifndef TUYA_BLE_APP_DEMO_H_
2  #define TUYA_BLE_APP_DEMO_H_
3
4
5  #ifdef __cplusplus
6  extern "C" {
7  #endif
8
9
10
11 #define APP_PRODUCT_ID          "xxxxxxxx"
12
13 #define APP_BUILD_FIRMNAME      "tuya_ble_sdk_app_demo_nrf52832"
14
15 //固件版本
16 #define TY_APP_VER_NUM       0x0100
17 #define TY_APP_VER_STR       "1.0"
18
19 //硬件版本
20 #define TY_HARD_VER_NUM       0x0100
21 #define TY_HARD_VER_STR      "1.0"
22
23
24
25 void tuya_ble_app_init(void);
26
27
28 #ifdef __cplusplus
29 }
30 #endif
31
32 #endif //
33
```

图7- 9 nrf52832 移植示例图 9

17、在相应的位置分别调用 tuya_ble_app_init()、tuya_ble_main_tasks_exec()、tuya_ble_gatt_receive_data()、tuya_ble_common_uart_receive_data()、tuya_ble_disconnected_handler()、tuya_ble_connected_handler()，如图 5-1、5-2、5-4、5-5以及图 7-10 所示。

```c
/**@brief   Function for handling app_uart events.
 *
 * @details This function will receive a single character from the app_uart module and append it to
 *          a string. The string will be be sent over BLE when the last character received was a
 *          'new line' '\n' (hex 0x0A) or if the string has reached the maximum data length.
 */
/**@snippet [Handling the data received over UART] */
void uart_event_handle(app_uart_evt_t * p_event)
{
    //static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
    //static uint8_t index = 0;
    uint8_t rx_char=0;
    uint32_t       err_code;

    switch (p_event->evt_type)
    {
    case APP_UART_DATA_READY:
        UNUSED_VARIABLE(app_uart_get(&rx_char));
        //UNUSED_VARIABLE(app_uart_get(&data_array[index]));
        // index++;
        tuya_ble_common_uart_receive_data(&rx_char,1);
        break;

    case APP_UART_COMMUNICATION_ERROR:
        // APP_ERROR_HANDLER(p_event->data.error_communication);
        break;

    case APP_UART_FIFO_ERROR:
        //APP_ERROR_HANDLER(p_event->data.error_code);
        break;

    default:
        break;
    }
}
/**@snippet [Handling the data received over UART] */
```

图7- 10 nrf52832 移植示例图 10

18、开发调试阶段，图7-8所示的 auth_key_test 和 device_id_test 联系 tuya 负责对接的产品经理获取。

19、最后编译代码，下载进开发板执行，下载涂鸦智能 App，扫描添加设备即可联调。

其他平台移植示例

请参考各平台完整示例 demo.

OTA协议及接口介绍

固件升级和芯片平台架构关联性比较大，所以 tuya ble sdk 只提供固件升级接口，Application 只需通过 SDK 提供的 OTA 通信接口按照如下所述的 OTA 协议实现即可。

Application通过注册的回调函数（无 RTOS 环境下）或者注册的接收队列（RTOS 环境下）接收OTA数据，EVENT ID为`TUYA_BLE_CB_EVT_OTA_DATA`，数据格式见升级协议章节，OTA 响应数据通过 `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` 函数发送。

OTA升级流程

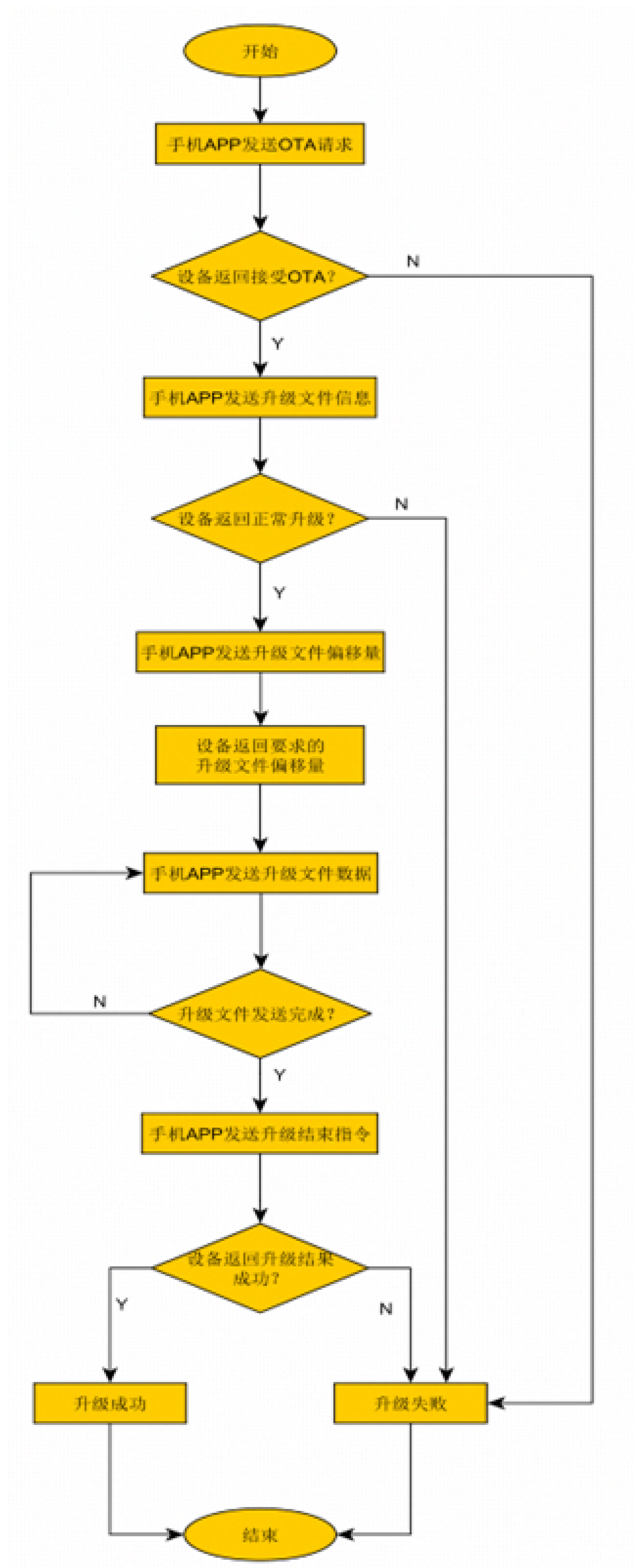


图8- 1 OTA 升级流程

OTA升级协议

OTA相关数据结构

```
typedef enum

{

•    TUYA_BLE_OTA_REQ,        //OTA升级请求指令
```

```
•    TUYA_BLE_OTA_FILE_INFO,     //OTA升级文件信息指令
•    TUYA_BLE_OTA_FILE_OFFSET_REQ, //OTA升级文件偏移量指令
•    TUYA_BLE_OTA_DATA,       //OTA升级数据指令
•    TUYA_BLE_OTA_END,       //OTA升级结束指令
•    TUYA_BLE_OTA_UNKONWN,
}tuya_ble_ota_data_type_t;
typedef struct{
•    tuya_ble_ota_data_type_t type;
•    uint16_t data_len;
•    uint8_t *p_data;
}tuya_ble_ota_data_t;   //手机App发送OTA升级EVENT（TUYA_BLE_CB_EVT_OTA_DATA）对应的数据结构。
typedef struct{
•    tuya_ble_ota_data_type_t type;
•    uint16_t data_len;
•    uint8_t *p_data;
}tuya_ble_ota_response_t;  // OTA响应数据发送函数 tuya_ble_ota_response(tuya_ble_ota_response_t *p_data) 对应的数据结构
```

OTA升级请求（`TUYA_BLE_OTA_REQ`）

App>设备:

	data_len=1
长度:	1 字节
Data:	固定 0

设备->App:

	data_len=9				
长度:	1 字节	1 字节	1 字节	4 字节	2 字节
data:	Flag	OTA_Version	0	Version 四字节	最大包长度

Flag：0x00 - 允许升级，0x01 - 拒绝升级。

OTA_Version：OTA 协议大版本，例如 0x03 代表 3.X 的协议版本。

Version：当前固件版本号,大端格式，例如 0x00 01 00 02 代表版本为 V1.0.2。

最大包长度：设备允许的单包最大长度，单位字节，当前版本不要超过 256 字节。

OTA升级文件信息（`TUYA_BLE_OTA_FILE_INFO`）

App->设备:

	data_len=37					
长度:	1 字节	8 字节	4 字节	16 字节	4 字节	4 字节
Data:	0	产品 PID	文件版本	文件 MD5	文件长度	CRC32

文件版本：例如，0x00010002代表版本为V1.0.2。

设备->App:

	data_len=26				
长度:	1 字节	1 字节	4 字节	4 字节	16 字节
data:	0	STATE	已储存文件长度	已储存文件 CRC32	已储存文件 MD5（目前不使用）

STATE：

0x00：正常升级

0x01：产品 PID 不一致

0x02: 文件版本低于或者等于当前版本

0x03：文件大小超过范围。

其他：保留。

已储存文件信息：

说明：为了支持断点续传，这里会返回设备端已经储存的文件信息，App 在收到后，首先根据设备返回的已储存文件长度计算新文件对应长度的 CRC32，然后和设备返回的 CRC32 对比，如果两者都吻合，那么在下面的文件起始传输请求中将起始传输偏移量改为该长度值，否则文件起始传输偏移量改为 0，表示从头开始传输。

OTA升级文件偏移（`TUYA_BLE_OTA_FILE_OFFSET_REQ`）

App->设备:

	data_len=5	
长度:	1 字节	4 字节
Data:	0	Offset

offset：升级文件偏移量。

设备->App:

	data_len=5	
长度:	1 字节	4 字节
Data:	0	Offset

offset：设备要求的起始传输文件偏移量。

说明：实际文件传输的偏移地址应该以设备端要求的为准，且设备端要求的地址会小于等于 App端给出的偏移。

OTA升级数据（`TUYA_BLE_OTA_DATA`）

App->设备:

	data_len=7+n				
长度:	1 字节	2 字节	2 字节	2 字节	n 字节
Data:	0	包号	当前包数据长度 n	当前包数据 CRC16	当前包数据

包号从 0 开始，高字节在前。

设备->App:

	data_len=2	
长度:	1 字节	1 字节
Data:	0	STATE

STATE：

0x00：成功

0x01：包号异常

0x02：长度不一致。

0x03：crc检验失败

0x04：其它

OTA升级结束（`TUYA_BLE_OTA_END`）

App->设备：

	data_len=1
长度：	1 字节
Data：	0

设备->App:

	data_len=2	
长度：	1 字节	1 字节
Data：	0	STATE

STATE：

0x00：成功

0x01：数据总长度错误

0x02: 数据总 CRC 检验失败

0x03: 其它

设备 OTA 文件验证成功后如果需要重启，通过 API `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` 响应给 App 结果至少无阻塞延时 2 秒后再重启。

OTA升级接口

Application通过注册的回调函数（无 RTOS 环境下）或者注册的接收队列（RTOS 环境下）接收OTA 数据，EVENT ID 为 `TUYA_BLE_CB_EVT_OTA_DATA`，OTA 响应数据通过 `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` 函数发送。

如图8-2所示，Application 在此处调用自定义的 OTA 处理函数，参考处理函数原型如图 8-3 所示。

```
static void tuya_cb_handler(tuya_ble_cb_evt_param_t* event)
{
    int16_t result = 0;
    switch (event->evt)
    {
    case TUYA_BLE_CB_EVT_CONNECTE_STATUS:
        TUYA_BLE_LOG_INFO("received tuya ble conncet status update event,current connect status = %d",event->connect_status);
        break;
    case TUYA_BLE_CB_EVT_DP_WRITE:
        dp_data_len = event->dp_write_data.data_len;
        memset(dp_data_array,0,sizeof(dp_data_array));
        memcpy(dp_data_array,event->dp_write_data.p_data,dp_data_len);
        TUYA_BLE_LOG_HEXDUMP_DEBUG("received dp write data :",dp_data_array,dp_data_len);
        tuya_ble_dp_data_report(dp_data_array,dp_data_len);
        custom_evt_1_send_test(dp_data_len);
        break;
    case TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE:
        TUYA_BLE_LOG_INFO("received dp data report response result code =%d",event->dp_response_data.status);
        break;
    case TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE:
        TUYA_BLE_LOG_INFO("received dp data report response result code =%d",event->dp_response_data.status);
        break;
    case TUYA_BLE_CB_EVT_UNBOUND:

        TUYA_BLE_LOG_INFO("received unbound req");

        break;
    case TUYA_BLE_CB_EVT_ANOMALY_UNBOUND:

        TUYA_BLE_LOG_INFO("received anomaly unbound req");

        break;
    case TUYA_BLE_CB_EVT_DEVICE_RESET:

        TUYA_BLE_LOG_INFO("received device reset req");

        break;
    case TUYA_BLE_CB_EVT_DP_QUERY:
        TUYA_BLE_LOG_INFO("received TUYA_BLE_CB_EVT_DP_QUERY event");
        tuya_ble_dp_data_report(dp_data_array,dp_data_len);
        break;
    case TUYA_BLE_CB_EVT_OTA_DATA:
        tuya_ota_proc(event->ota_data.type,event->ota_data.p_data,event->ota_data.data_len);
        break;
```

图8- 2 OTA 数据接口

OTA 处理函数参考：

```
void tuya_ota_proc(uint16_t cmd,uint8_t*recv_data,uint32_t recv_len)
{
    TUYA_BLE_LOG_DEBUG("ota cmd : 0x%04x , recv_len : %d",cmd,recv_len);
    switch(cmd)
    {
    case TUYA_BLE_OTA_REQ:
        tuya_ota_start_req(recv_data,recv_len);
        break;
    case TUYA_BLE_OTA_FILE_INFO:
        tuya_ota_file_info_req(recv_data,recv_len);
        break;
    case TUYA_BLE_OTA_FILE_OFFSET_REQ:
        tuya_ota_offset_req(recv_data,recv_len);
        break;
    case TUYA_BLE_OTA_DATA:
        tuya_ota_data_req(recv_data,recv_len);
        break;
    case TUYA_BLE_OTA_END:
        tuya_ota_end_req(recv_data,recv_len);
        break;
    default:
        break;
    }

}
```

产测接口介绍

BLE 设备接入涂鸦 IoT 平台需要预先烧录授权信息（一机一密），一般在工厂生产时烧录，客户可以使用涂鸦的产测工具进行烧录授权以及测试，当然也可以批量购买 license 使用自定义的协议和接口管理，如果使用自定义的接口管理授权信息，需要配置 Tuya BLE SDK 不管理 license，即

`#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT  0`

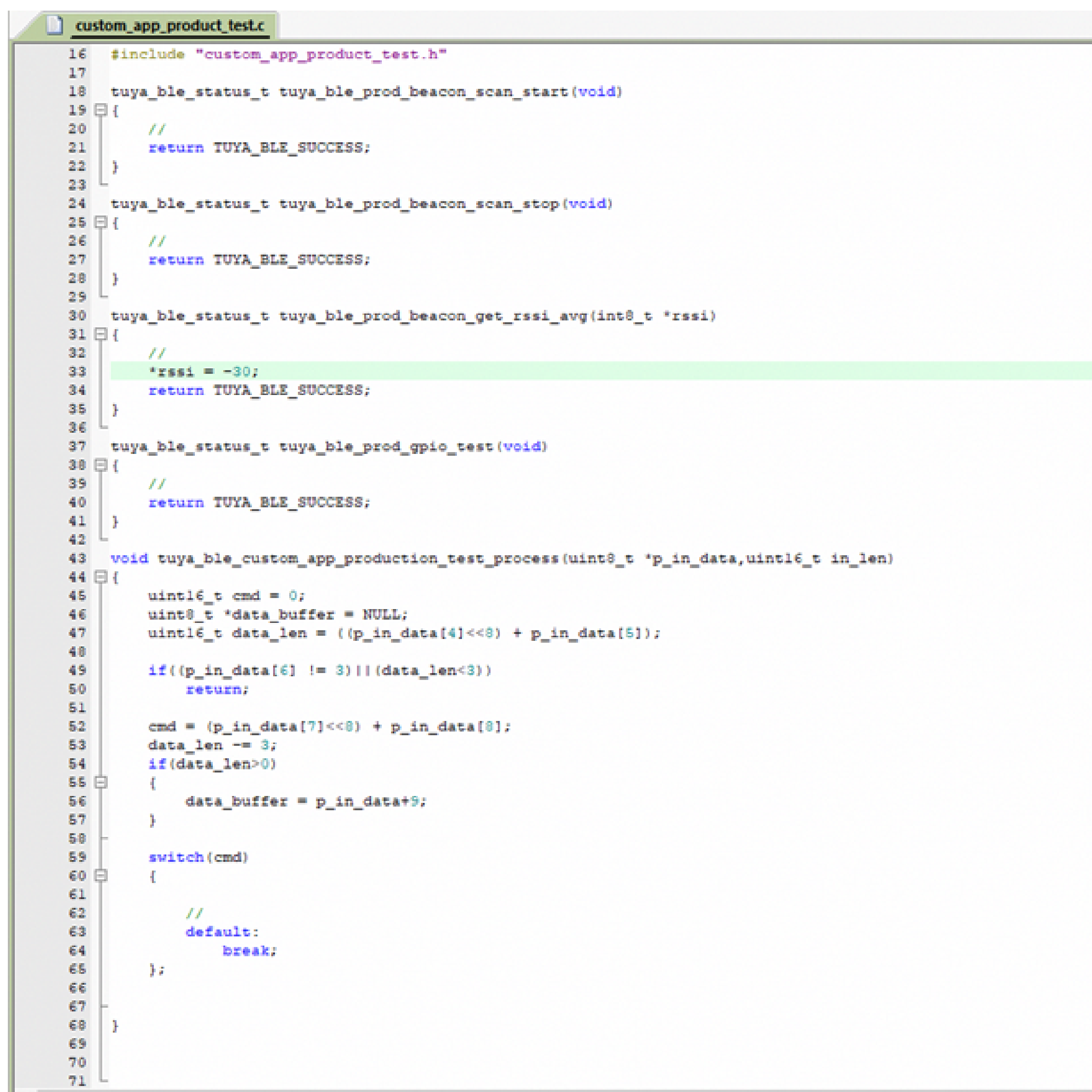
如果 TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 配置为 0，客户应用程序需要在初始化 BLE SDK 时代入各种 ID 信息，并在收到绑定时 SDK 发送的 login key、VID、bound flag 时安全存储到NV中。

如果使用涂鸦产测工具进行授权、测试并希望 Tuya BLE SDK 管理授权信息，请关注本章内容，并配置 TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 为 1。

产测分为 “通用产测授权“ 和 “通用整机产测”，“通用整机产测协议” 是 “通用产测授权协议” 的子集，“通用产测授权“ 主要包括烧录授权信息、GPIO测试以及RSSI测试；通用整机产测包含产品定制附加的一些测试，具体协议格式请参考《蓝牙通用产测授权协议》和《蓝牙通用整机产测协议》。

BLE SDK 已实现了“通用产测授权”的协议，但是像 RSSI 测试（被测设备扫描特定的 beacon 信标）和 GPIO 测试以及基于“通用整机产测协议”的产品附加项目测试需要根据产品定义实现，SDK 中 `tuya_ble_app_production_test.c` 源文件中已对这几项测试预留了对应函数接口，都是以 `__TUYA_BLE_WEAK` 定义的弱实现，客户应用程序只需要在其他的源文件中重新定义这几个函数即可，并在自定义配置文件中引用。

如图 9-1、9-2、9-3 所示。



```
custom_app_product_test.c
16  #include "custom_app_product_test.h"
17
18  tuya_ble_status_t tuya_ble_prod_beacon_scan_start(void)
19  {
20      //
21      return TUYA_BLE_SUCCESS;
22  }
23
24  tuya_ble_status_t tuya_ble_prod_beacon_scan_stop(void)
25  {
26      //
27      return TUYA_BLE_SUCCESS;
28  }
29
30  tuya_ble_status_t tuya_ble_prod_beacon_get_rssi_avg(int8_t *rssi)
31  {
32      //
33      *rssi = -30;
34      return TUYA_BLE_SUCCESS;
35  }
36
37  tuya_ble_status_t tuya_ble_prod_gpio_test(void)
38  {
39      //
40      return TUYA_BLE_SUCCESS;
41  }
42
43  void tuya_ble_custom_app_production_test_process(uint8_t *p_in_data,uint16_t in_len)
44  {
45      uint16_t cmd = 0;
46      uint8_t *data_buffer = NULL;
47      uint16_t data_len = ((p_in_data[4]<<8) + p_in_data[5]);
48
49      if((p_in_data[6] != 3)||(data_len<3))
50          return;
51
52      cmd = (p_in_data[7]<<8) + p_in_data[8];
53      data_len -= 3;
54      if(data_len>0)
55      {
56          data_buffer = p_in_data+9;
57      }
58
59      switch(cmd)
60      {
61
62          //
63          default:
64              break;
65      };
66
67
68  }
69
70
71
```

图 9- 1 应用实现的相关产测函数源文件示例

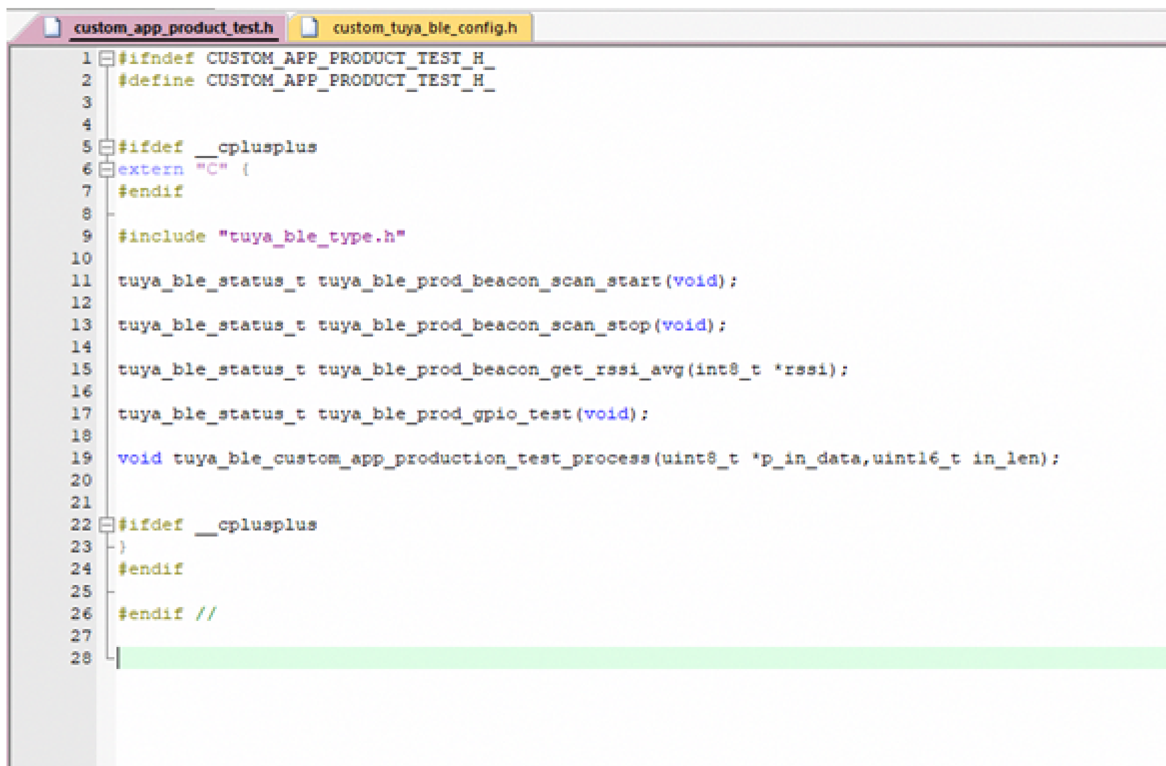


图 9- 2 应用实现的相关产测函数头文件示例

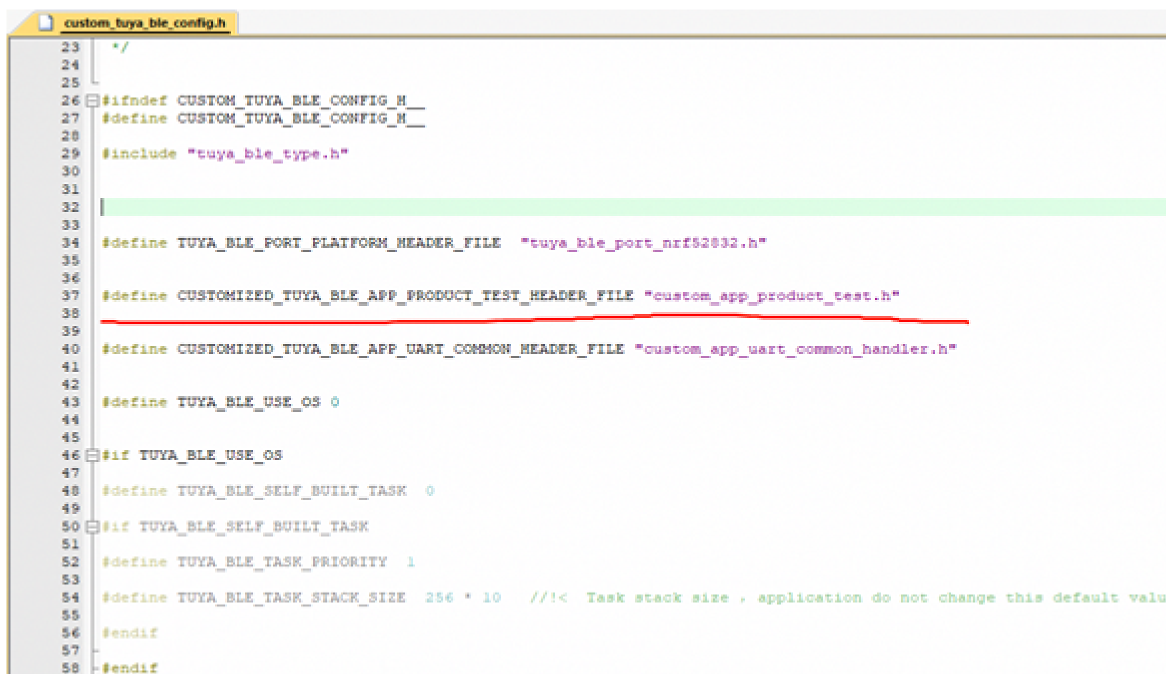


图 9- 3 应用配置文件引用自定义产测文件示例

附录