



TUYA BLE SDK User Guide

V1.1.0

2019/12/16

修订历史（Revision History）

版本	日期	作者	变更描述
1.0.0	2019-06-14	高永会	创建文档
1.0.1	2019-08-26	苏钉	新增
1.1.0	2019-12-16	高永会	新增

目录

目录

修订历史 (Revision History)	2
目录	3
图目录	9
表目录	11
1 概述	12
2 SDK 架构	12
2.1 系统架构	12
2.2 OS 支持	13
2.3 事件队列	13
2.4 SDK 目录	14
3 TUYA BLE SERVICE	15
3.1 概述	15
3.2 SERVICE	15
3.3 MTU	15
3.4 广播数据格式	16
3.4.1 ble 广播包数据结构	16
3.4.2 Tuya ble adv data	16
3.4.3 Tuya ble scan response data	17
4 PORT 和 CONFIG 介绍	18
4.1 port 说明	18
4.2 port 接口介绍	18
4.2.1 TUYA_BLE_LOG	18
4.2.2 TUYA_BLE_HEXDUMP	19
4.2.3 tuya_ble_gap_advertising_adv_data_update	19
4.2.4 tuya_ble_gap_advertising_scan_rsp_data_update	19
4.2.5 tuya_ble_gap_disconnect	20

4.2.6	tuya_ble_gatt_send_data.....	20
4.2.7	tuya_ble_timer_create	20
4.2.8	tuya_ble_timer_delete	21
4.2.9	tuya_ble_timer_start.....	21
4.2.10	tuya_ble_timer_restart.....	22
4.2.11	tuya_ble_timer_stop	22
4.2.12	tuya_ble_device_delay_ms.....	23
4.2.13	tuya_ble_device_delay_us	23
4.2.14	tuya_ble_device_reset	23
4.2.15	tuya_ble_gap_addr_get.....	24
4.2.16	tuya_ble_gap_addr_set	24
4.2.17	tuya_ble_device_enter_critical	25
4.2.18	tuya_ble_device_exit_critical.....	25
4.2.19	tuya_ble_rand_generator	25
4.2.20	tuya_ble_rtc_get_timestamp	26
4.2.21	tuya_ble_rtc_set_timestamp	26
4.2.22	tuya_ble_nv_init.....	26
4.2.23	tuya_ble_nv_erase.....	27
4.2.24	tuya_ble_nv_write	27
4.2.25	tuya_ble_nv_read	28
4.2.26	tuya_ble_common_uart_init.....	28
4.2.27	tuya_ble_common_uart_send_data	28
4.2.28	tuya_ble_os_task_create	29
4.2.29	tuya_ble_os_task_delete	29
4.2.30	tuya_ble_os_task_suspend.....	30
4.2.31	tuya_ble_os_task_resume.....	30
4.2.32	tuya_ble_os_msg_queue_create	31
4.2.33	tuya_ble_os_msg_queue_delete	31
4.2.34	tuya_ble_os_msg_queue_peek.....	31

4.2.35	tuya_ble_os_msg_queue_send	32
4.2.36	tuya_ble_os_msg_queue_rcv	32
4.2.37	tuya_ble_event_queue_send_port.....	33
4.2.38	tuya_ble_aes128_ecb_encrypt.....	34
4.2.39	tuya_ble_aes128_ecb_decrypt.....	34
4.2.40	tuya_ble_aes128_cbc_encrypt	35
4.2.41	tuya_ble_aes128_cbc_decrypt	35
4.2.42	tuya_ble_md5_crypt.....	36
4.2.43	tuya_ble_hmac_sha1_crypt.....	36
4.2.44	tuya_ble_hmac_sha256_crypt	36
4.2.45	tuya_ble_port_malloc	37
4.2.46	tuya_ble_port_free	37
4.3	config 说明.....	38
4.3.1	CUSTOMIZED_TUYA_BLE_CONFIG_FILE	38
4.3.2	TUYA_BLE_USE_OS.....	38
4.3.3	TUYA_BLE_SELF_BUILT_TASK.....	39
4.3.4	TUYA_BLE_TASK_PRIORITY.....	39
4.3.5	TUYA_BLE_TASK_STACK_SIZE.....	39
4.3.6	TUYA_BLE_DEVICE_COMMUNICATION_ABILITY	40
4.3.7	TUYA_BLE_WIFI_DEVICE_REGISTER_MODE	40
4.3.8	TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT.....	41
4.3.9	TUYA_BLE_SECURE_CONNECTION_TYPE	41
4.3.10	TUYA_BLE_DEVICE_MAC_UPDATE_RESET	42
4.3.11	TUYA_BLE_USE_PLATFORM_MEMORY_HEAP.....	42
4.3.12	TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE	42
4.3.13	TUYA_BLE_DATA_MTU_MAX.....	43
4.3.14	TUYA_BLE_LOG_ENABLED	43
4.3.15	TUYA_BLE_LOG_COLORS_ENABLE.....	43
4.3.16	TUYA_BLE_LOG_LEVEL.....	44

4.3.17	TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE.....	44
4.3.18	TUYA_NV_ERASE_MIN_SIZE.....	44
4.3.19	TUYA_NV_WRITE_GRAN	45
4.3.20	TUYA_NV_START_ADDR.....	45
4.3.21	TUYA_NV_AREA_SIZE.....	45
5	API 介绍.....	45
5.1	tuya_ble_main_tasks_exec.....	46
5.2	tuya_ble_gatt_receive_data.....	46
5.3	tuya_ble_common_uart_receive_data	47
5.4	tuya_ble_common_uart_send_full_instruction_received	48
5.5	tuya_ble_device_update_product_id	48
5.6	tuya_ble_device_update_login_key	49
5.7	tuya_ble_device_update_bound_state	49
5.8	tuya_ble_device_update_mcu_version.....	50
5.9	tuya_ble_sdk_init	50
5.10	tuya_ble_dp_data_report.....	53
5.11	tuya_ble_dp_data_with_time_report.....	54
5.12	tuya_ble_dp_data_with_time_ms_string_report	55
5.13	tuya_ble_connected_handler	55
5.14	tuya_ble_disconnected_handler.....	56
5.15	tuya_ble_data_passthrough.....	56
5.16	tuya_ble_production_test_asynchronous_response	57
5.17	tuya_ble_net_config_response.....	58
5.18	tuya_ble_ubound_response.....	58
5.19	tuya_ble_anomaly_ubound_response.....	59
5.20	tuya_ble_device_reset_response	59
5.21	tuya_ble_connect_status_get	60
5.22	tuya_ble_device_factory_reset	60
5.23	tuya_ble_time_req.....	61

5.24	tuya_ble_ota_response.....	61
5.25	tuya_ble_custom_event_send	62
5.26	tuya_ble_callback_queue_register	62
5.27	tuya_ble_event_response	64
6	CALL BACK EVENT 介绍.....	66
6.1	TUYA_BLE_CB_EVT_CONNECTE_STATUS	66
6.2	TUYA_BLE_CB_EVT_DP_WRITE	68
6.3	TUYA_BLE_CB_EVT_DP_QUERY	68
6.4	TUYA_BLE_CB_EVT_OTA_DATA.....	68
6.5	TUYA_BLE_CB_EVT_NETWORK_INFO.....	69
6.6	TUYA_BLE_CB_EVT_WIFI_SSID	69
6.7	TUYA_BLE_CB_EVT_TIME_STAMP	70
6.8	TUYA_BLE_CB_EVT_TIME_NORMAL	70
6.9	TUYA_BLE_CB_EVT_DATA_PASSTHROUGH	71
6.10	TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE	71
6.11	TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE.....	71
6.12	TUYA_BLE_CB_EVT_UNBOUND.....	72
6.13	TUYA_BLE_CB_EVT_ANOMALY_UNBOUND	72
6.14	TUYA_BLE_CB_EVT_DEVICE_RESET	72
6.15	TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID.....	73
7	移植示例	73
7.1	nrf52832 移植示例	73
7.2	其他平台移植示例	81
8	OTA 协议及接口介绍	81
8.1	OTA 升级流程.....	82
8.2	OTA 升级协议.....	83
8.2.1	OTA 相关数据结构	83
8.2.2	OTA 升级请求 (TUYA_BLE_OTA_REQ)	83
8.2.3	OTA 升级文件信息 (TUYA_BLE_OTA_FILE_INFO)	84

8.2.4	OTA 升级文件偏移 (TUYA_BLE_OTA_FILE_OFFSET_REQ)	85
8.2.5	OTA 升级数据 (TUYA_BLE_OTA_DATA)	86
8.2.6	OTA 升级结束 (TUYA_BLE_OTA_END)	86
8.3	OTA 升级接口	87
9	产测接口介绍	89
附录	92

图目录

图 2- 1 TUYA BLE SDK 系统架构	13
图 2- 2 SDK 目录	14
图 3- 1 ble 广播包数据结构	16
图 4- 1 port 文件夹目录结构	18
图 5- 1 tuyu_ble_main_tasks_exec 调用示例	46
图 5- 2 tuyu_ble_gatt_receive_data 示例	47
图 5- 3 tuyu_ble_device_param_t 结构体	51
图 5- 4 tuyu ble sdk 初始化示例	52
图 5- 5 tuyu_ble_connected_handler 示例	57
图 5- 6 tuyu_ble_custom_event_send 示例	63
图 5- 7 tuyu_ble_event_response 示例	65
图 6- 1 call back 函数示例	67
图 7- 1 nrf52832 移植示例图 1	74
图 7- 2 nrf52832 移植示例图 2	75
图 7- 3 nrf52832 移植示例图 3	75
图 7- 4 nrf52832 移植示例图 4	76
图 7- 5 nrf52832 移植示例图 5	77
图 7- 6 nrf52832 移植示例图 6	78
图 7- 7 nrf52832 移植示例图 7	78
图 7- 8 nrf52832 移植示例图 8	79
图 7- 9 nrf52832 移植示例图 9	80
图 7- 10 nrf52832 移植示例图 10	81
图 8- 1 OTA 升级流程	82
图 8- 2 OTA 数据接口	88
图 8- 3 OTA 处理函数参考	88
图 9- 1 应用实现的相关产测函数源文件示例	90
图 9- 2 应用实现的相关产测函数头文件示例	91

图 9- 3 应用配置文件引用自定义产测文件示例.....	91
-------------------------------	----

表目录

表 3- 1 tuyu Service characteristics.....	15
表 3- 2 adv data.....	16
表 3- 3 scan response data	17
表 5- 1 DP 点数据组装格式	54

1 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 APP 互联互通。

2 SDK 架构

2.1 系统架构

如图 2-1 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）。

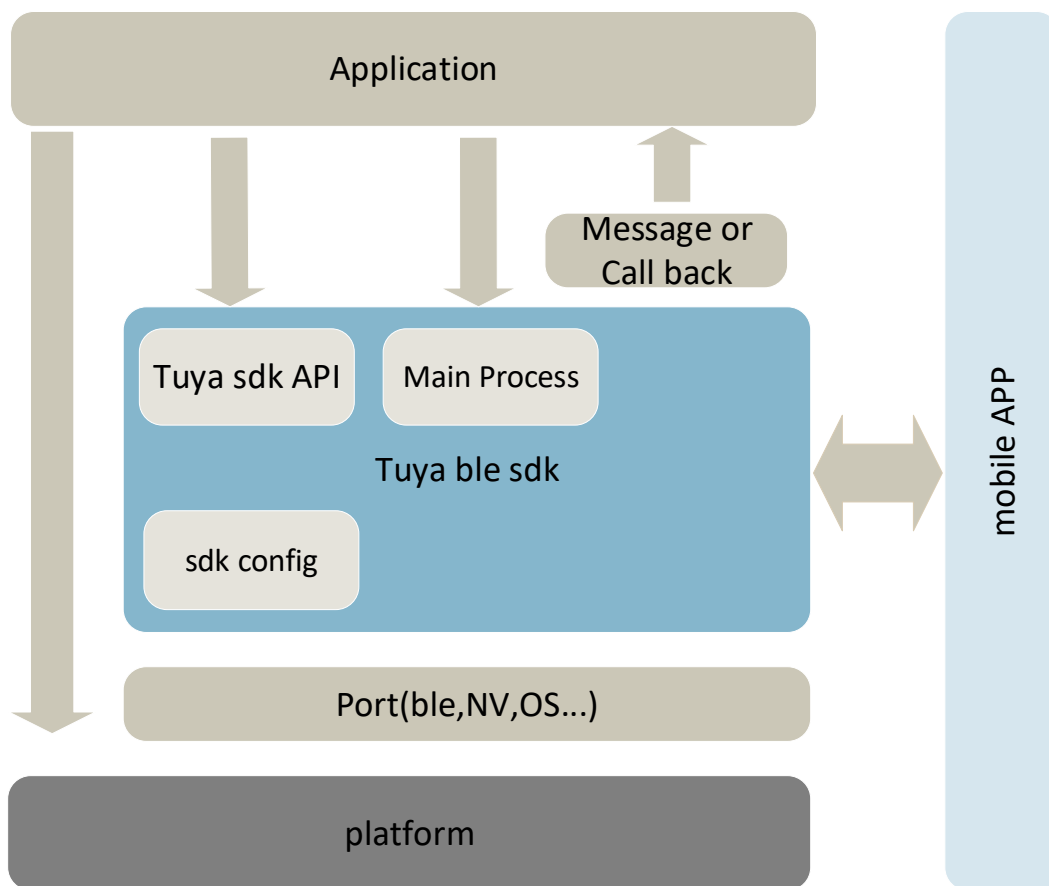


图 2- 1 TUYA BLE SDK 系统架构

2.2 OS 支持

TUYA BLE SDK 可运行在基于 RTOS 的芯片平台下（linux 暂不支持）。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。

2.3 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。

2.4 SDK 目录

Tuya ble sdk 目录如图 2-2 所示:

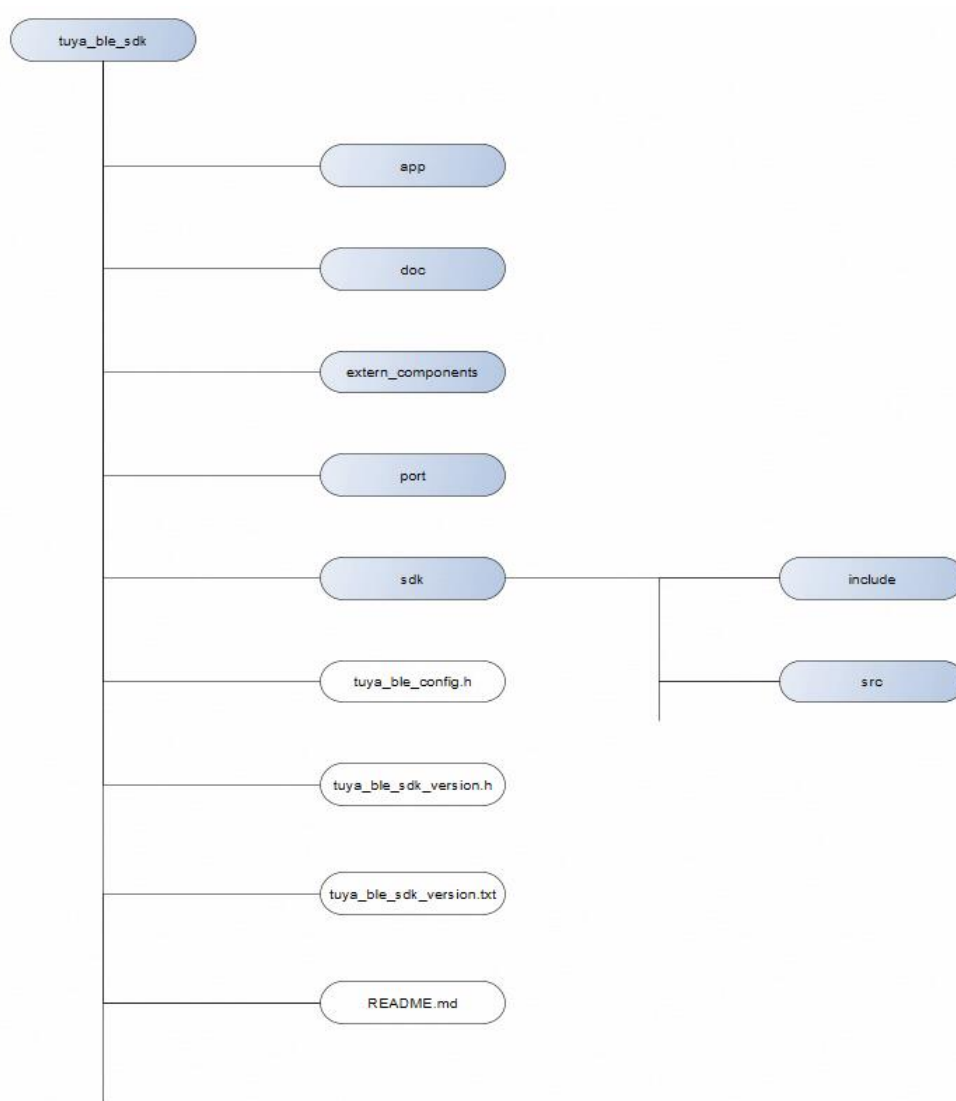


图 2-2 SDK 目录

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

3 TUYA BLE SERVICE

3.1 概述

Tuya ble sdk 不提供初始化 service 相关接口，Application 需要在初始化 sdk 前实现表 3-1 所定义的 service characteristics，当然，Application 除了定义 tuya ble sdk 所需的 service 外，也可以定义其他 service。广播数据的初始格式见表 3-2 和表 3-3，否则 sdk 将不能正常工作。

3.2 SERVICE

表 3-1 tuya Service characteristics

Service UUID	Characteristic UUID	Properties	Security Permissions
1910	2b10	Notify	None.
	2b11	Write, write without response	None.

3.3 MTU

基于更好的兼容性考虑，目前 tuya ble 使用的 ATT MTU 为 23，GATT MTU（ATT DATA MAX）为 20。

3.4 广播数据格式

3.4.1 ble 广播包数据结构

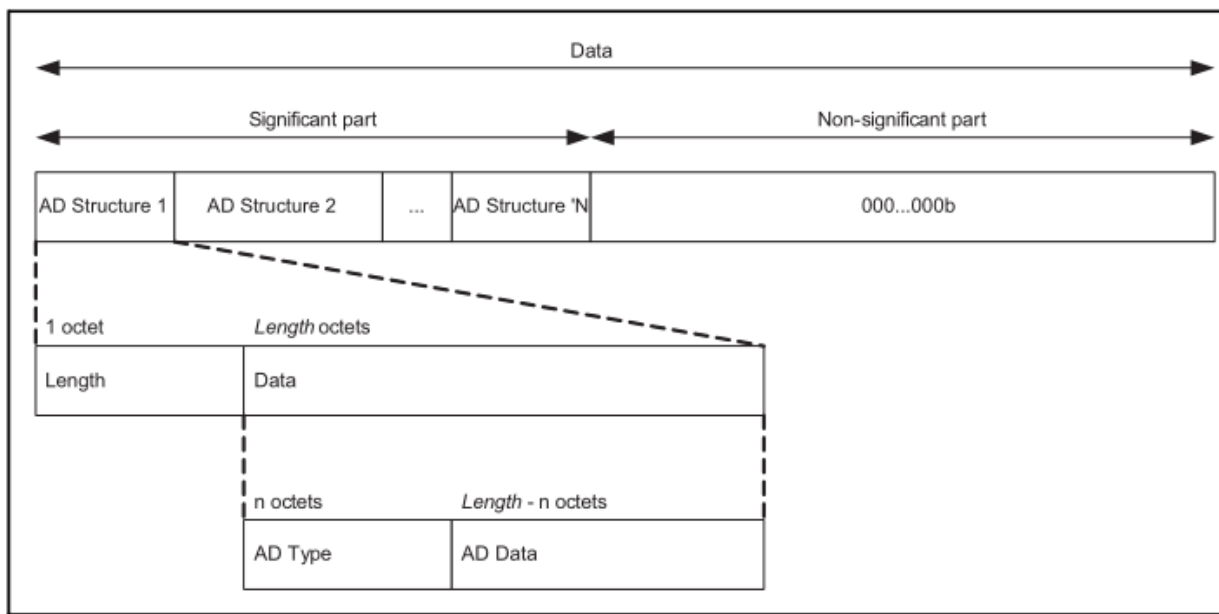


图 3- 1 ble 广播包数据结构

3.4.2 Tuya ble adv data

表 3- 2 adv data

广播数据段描述	类型	说明
设备 LE 物理连接标识	0x01	长度： 0x02 类型： 0x01 数据： 0x06
Service UUID	0x02	长度： 0x03 类型： 0x02 数据： 0xA201

4 PORT 和 CONFIG 介绍

4.1 port 说明

如图 4-1 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。以平台名字命名的文件里是 sdk 已经适配移植好的平台实现，用户可以直接使用。

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

图 4- 1 port 文件夹目录结构

4.2 port 接口介绍

4.2.1 TUYA_BLE_LOG

函数名	TUYA_BLE_LOG
函数原型	void TUYA_BLE_LOG(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	
----	--

4.2.2 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	

4.2.3 tuyu_ble_gap_advertising_adv_data_update

函数名	tuya_ble_gap_advertising_adv_data_update
函数原型	tuya_ble_status_t tuyu_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	

4.2.4 tuyu_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 tuyu_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	

4.2.5 tuy_ble_gap_disconnect

函数名	tuya_ble_gap_disconnect
函数原型	tuya_ble_status_t tuy_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	

4.2.6 tuy_ble_gatt_send_data

函数名	tuya_ble_gatt_send_data
函数原型	tuya_ble_status_t tuy_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	必须是 notify 方式发送。

4.2.7 tuy_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	

4.2.8 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	

4.2.9 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。

4.2.10 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	

4.2.11 tuya_ble_timer_stop

函数名	tuya_ble_timer_stop
函数原型	tuya_ble_status_t tuya_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	

4.2.12 tuy_ble_device_delay_ms

函数名	tuya_ble_device_delay_ms
函数原型	void tuy_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。

4.2.13 tuy_ble_device_delay_us

函数名	tuya_ble_device_delay_us
函数原型	void tuy_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。

4.2.14 tuy_ble_device_reset

函数名	tuya_ble_device_reset
函数原型	tuya_ble_status_t tuy_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	

4.2.15 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;
```

4.2.16 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	

4.2.17 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	

4.2.18 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	

4.2.19 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	

4.2.20 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	数据来自应用本身维护的 rtc 实时时钟。

4.2.21 tuya_ble_rtc_set_timestamp

函数名	tuya_ble_rtc_set_timestamp
函数原型	tuya_ble_status_t tuya_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。

4.2.22 tuya_ble_nv_init

函数名	tuya_ble_nv_init
函数原型	tuya_ble_status_t tuya_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。

4.2.23 tuya_ble_nv_erase

函数名	tuya_ble_nv_erase
函数原型	tuya_ble_status_t tuya_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。

4.2.24 tuya_ble_nv_write

函数名	tuya_ble_nv_write
函数原型	tuya_ble_status_t tuya_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。

4.2.25 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。

4.2.26 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	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。

4.2.27 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	

4.2.28 tuy_a_ble_os_task_create

函数名	tuya_ble_os_task_create
函数原型	bool tuy_a_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] : A descriptive name for the task; p_routine [in] : Pointer to 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] : The priority at which the task should run. Higher priority task has higher priority value。
返回值	True : Task was created 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。

4.2.29 tuy_a_ble_os_task_delete

函数名	tuya_ble_os_task_delete
函数原型	bool tuy_a_ble_os_task_delete(void *p_handle)

功能概述	Remove a task from RTOS's task management. The task being deleted will be removed from RUNNING, READY or WAITING state
参数	pp_handle [in] : The handle of the task to be deleted.
返回值	True : Task was deleted 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。

4.2.30 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 microcontroller processing time.
参数	pp_handle [in] : The handle of the task to be suspend.
返回值	True : Task was suspend 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。

4.2.31 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 the suspended task.
参数	pp_handle [in] : The handle of the task to be resumed.
返回值	True : Task was resumed 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。

4.2.32 tuy_ble_os_msg_queue_create

函数名	tuya_ble_os_msg_queue_create
函数原型	bool tuy_ble_os_msg_queue_create(void **pp_handle, uint32_t msg_num, uint32_t msg_size)
功能概述	Creates a message queue instance. This allocates the storage required by the new queue and passes 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] : The maximum number of items that the queue can contain. msg_size [in] : The number of bytes each item in the queue will require.
返回值	True : Message queue 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。

4.2.33 tuy_ble_os_msg_queue_delete

函数名	tuya_ble_os_msg_queue_delete
函数原型	bool tuy_ble_os_msg_queue_delete(void *p_handle)
功能概述	Deletes a message queue instance. This deallocates the storage required by the new queue and passes back a handle for the queue.
参数	pp_handle [in] : The handle to the message queue being deleted.
返回值	True : Message queue 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。

4.2.34 tuy_ble_os_msg_queue_peek

函数名	tuya_ble_os_msg_queue_peek
函数原型	bool tuy_ble_os_msg_queue_peek(void *p_handle, uint32_t *p_msg_num)
功能概述	Peek the number of items sent and resided on the message queue.

参数	pp_handle [in] : The handle to the message queue being peeked. p_msg_num[out] : Used to pass back the number of items residing on the message queue.
返回值	True : Message queue 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。

4.2.35 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 the back of the specified message queue.
参数	pp_handle [in] : The handle to the message queue on which the item is to be sent. p_msg[in] : Pointer to the item that is to be sent on the queue. wait_ms[in] : The maximum amount of time in milliseconds that the task should block waiting for the item to sent on the queue. * 0 No blocking and return immediately. * 0xFFFFFFFF Block infinitely until the item sent. * others The timeout value in milliseconds.
返回值	True : Message item was 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。

4.2.36 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 item from the specified message queue.

参数	<p>pp_handle [in] : The handle to the message queue from which the item is to be received.</p> <p>p_msg[out] : Pointer to the buffer into which the received item will be copied.</p> <p>wait_ms[in] : The maximum amount of time in milliseconds that the task should block waiting for the item to received on the queue.</p> <p>* 0 No blocking and return immediately.</p> <p>* 0xFFFFFFFF Block infinitely until the item received.</p> <p>* others The timeout value in milliseconds.</p>
返回值	<p>True : Message item was received successfully.</p> <p>false: Message item was failed to receive.</p>
备注	该接口函数只需在基于 os 的平台下实现。

4.2.37 tuyu_ble_event_queue_send_port

函数名	tuya_ble_event_queue_send_port
函数原型	bool tuyu_ble_event_queue_send_port(tuya_ble_evt_param_t *evt, uint32_t wait_ms)
功能概述	If undefine TUYA_BLE_SELF_BUILT_TASK ,application should provide the task to sdk to process the event. SDK will use this port to send event to the task of provided by application.
参数	<p>evt [in] : the message data point to be send.</p> <p>wait_ms[in] : The maximum amount of time in milliseconds that the task should block waiting for the item to sent on the queue.</p> <p>* 0 No blocking and return immediately.</p> <p>* 0xFFFFFFFF Block infinitely until the item sent.</p> <p>* others The timeout value in milliseconds.</p>
返回值	<p>True : Message item was sent successfully.</p> <p>false: Message item was failed to send.</p>
备注	该接口函数只需在基于 os 的平台下实现。

4.2.38 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_t input_len,uint8_t *output)
功能概述	128 bit AES ECB encryption on speicified plaintext and keys
参数	key [in] : keys to encrypt the plaintext In_put[in] : specifed plain text to be encrypted. in_put_len[in] : byte length of the data to be encrypted, must be multiples of 16. Out_put[out] : output buffer to store encrypted data.
返回值	True : successful. false: fail.
备注	

4.2.39 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_t input_len,uint8_t *output)
功能概述	128 bit AES ECB decryption on speicified encrypted data and keys
参数	key [in] : keys to decrypt the plaintext In_put[in] : specifed encrypted data to be decrypted in_put_len[in] : byte length of the data to be descrypted, must be multiples of 16. Out_put[out] : output buffer to store decrypted data.
返回值	True : successful. false: fail.
备注	

4.2.40 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_t *output)
功能概述	128 bit AES CBC encryption on specified plaintext and keys.
参数	<p>key [in] : keys to encrypt the plaintext.</p> <p>iv[in] : initialization vector (IV) for CBC mode.</p> <p>In_put[in] : specified plain text to be encrypted.</p> <p>in_put_len[in] : byte length of the data to be encrypted, must be multiples of 16.</p> <p>Out_put[out] : output buffer to store encrypted data.</p>
返回值	<p>True : successful.</p> <p>false: fail.</p>
备注	

4.2.41 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 decryption on specified plaintext and keys.
参数	<p>key [in] : keys to decrypt the plaintext.</p> <p>lv[in] : initialization vector (IV) for CBC mode</p> <p>In_put[in] : specified encrypted data to be decrypted.</p> <p>in_put_len[in] : byte length of the data to be decrypted, must be multiples of 16.</p> <p>Out_put[out] : output buffer to store decrypted data.</p>
返回值	<p>True : successful.</p> <p>false: fail.</p>
备注	

4.2.42 tuya_ble_md5_crypt

函数名	tuya_ble_md5_crypt
函数原型	bool tuya_ble_md5_crypt(uint8_t *input, uint16_t input_len, uint8_t *output)
功能概述	MD5 checksum.
参数	In_put[in] : specied plain text to be encrypted. in_put_len[in] : yte length of the data to be encrypted. Out_put[out] : output buffer to store md5 result data, output data len is always 16.
返回值	True : successful. false: fail.
备注	

4.2.43 tuya_ble_hmac_sha1_crypt

函数名	tuya_ble_hmac_sha1_crypt
函数原型	bool tuya_ble_hmac_sha1_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 full generic HMAC on the input buffer with the provided key.
参数	key [in] : The HMAC secret key. key_len[in] : The length of the HMAC secret key in Bytes. In_put[in] : specied plain text to be encrypted. in_put_len[in] : byte length of the data to be encrypted. Out_put[out] : output buffer to store the result data.
返回值	True : successful. false: fail.
备注	

4.2.44 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 full generic HMAC on the input buffer with the provided key.
参数	key[in] : The HMAC secret key. key_len[in] : The length of the HMAC secret key in Bytes. In_put[in] : specied plain text to be encrypted. in_put_len[in] : byte length of the data to be encrypted. Out_put[out] : output buffer to store the result data.
返回值	True : successful. false: fail.
备注	

4.2.45 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 memory block with required size.
参数	Size[in] : Required memory size.
返回值	The address of the allocated 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。

4.2.46 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 block that had been allocated.
参数	pv[in] : The address of memory block 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。
----	--

4.3 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 是否自我管理授权信息等。

各配置项说明如下所示：

4.3.1 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	

4.3.2 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	

4.3.3 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 是否使用自建的 task 来处理事件，如果是： #define TUYA_BLE_SELF_BUILT_TASK 1 否则： #define TUYA_BLE_SELF_BUILT_TASK 0
备注	

4.3.4 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。

4.3.5 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	

4.3.6 TUYA_BLE_DEVICE_COMMUNICATION_ABILITY

宏定义	TUYA_BLE_DEVICE_COMMUNICATION_ABILITY
依赖项	无。
描述	<p>设备能力定义，例如：</p> <pre>#define TUYA_BLE_DEVICE_COMMUNICATION_ABILITY (TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_BLE TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_REGISTER_FROM_BLE)</pre> <p>参数说明：</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_BLE --- 是否支持 ble；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_REGISTER_FROM_BLE --- 是否通过 ble 注册设备；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_MESH --- 是否支持 MESH；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_24G --- 是否支持 2.4G wifi；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_5G --- 是否支持 5G wifi；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_ZIGBEE --- 是否支持 zigbee；</p> <p>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_NB --- 是否支持 NB。</p>
备注	

4.3.7 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	<p>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：</p> <pre>#define TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 1</pre> <p>否则：</p>

	#define TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 0
备注	目前暂不使用。

4.3.8 TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT

宏定义	TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT
依赖项	无。
描述	<p>Tuya ble sdk 是否需要自己管理授权信息，如果是：</p> <pre>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 1</pre> <p>否则：</p> <pre>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 0</pre>
备注	对于不具备 wifi 能力的单 BLE 设备，该值必须定义为 1。

4.3.9 TUYA_BLE_SECURE_CONNECTION_TYPE

宏定义	TUYA_BLE_SECURE_CONNECTION_TYPE
依赖项	无。
描述	<p>BLE 通信安全加密方式，定义如下：</p> <pre>TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY</pre> <p>--- encrypt with auth key;</p> <pre>TUYA_BLE_SECURE_CONNECTION_WITH_ECC</pre> <p>--- encrypt with ECDH;</p> <pre>TUYA_BLE_SECURE_CONNECTION_WITH_PASSTHROUGH</pre> <p>--- no encrypt;</p> <p>例如：</p> <pre>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY</pre>
备注	目前仅支持 TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY。

4.3.10 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	

4.3.11 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 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 ble sdk 使用。

4.3.12 TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE

宏定义	TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE
依赖项	无。
描述	tuya ble sdk 内部使用的 gatt 发送队列的大小，默认 20： #define TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE 20
备注	如果对该定义不了解，建议保持为 20。

4.3.13 TUYA_BLE_DATA_MTU_MAX

宏定义	TUYA_BLE_DATA_MTU_MAX
依赖项	无。
描述	GATT MTU 大小： #define TUYA_BLE_DATA_MTU_MAX 20
备注	目前仅支持 20 字节的 GATT MTU，后续会扩展升级。

4.3.14 TUYA_BLE_LOG_ENABLED

宏定义	TUYA_BLE_LOG_ENABLED
依赖项	无。
描述	是否开启 log，如果开启： #define TUYA_BLE_LOG_ENABLED 1 否则： #define TUYA_BLE_LOG_ENABLED 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。

4.3.15 TUYA_BLE_LOG_COLORS_ENABLE

宏定义	TUYA_BLE_LOG_COLORS_ENABLE
依赖项	TUYA_BLE_LOG_ENABLED
描述	是否开启 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 显示工具都支持，例如使用 jlink 的 RTT 时不支持。

4.3.16 TUYA_BLE_LOG_LEVEL

宏定义	TUYA_BLE_LOG_LEVEL
依赖项	TUYA_BLE_LOG_ENABLED
描述	<p>定义 log 的显示等级，分为如下几个等级：</p> <pre>#define TUYA_BLE_LOG_LEVEL_ERROR 1U #define TUYA_BLE_LOG_LEVEL_WARNING 2U #define TUYA_BLE_LOG_LEVEL_INFO 3U #define TUYA_BLE_LOG_LEVEL_DEBUG 4U</pre> <p>如果只需要打印错误信息，则定义为：</p> <pre>#define TUYA_BLE_LOG_LEVEL TUYA_BLE_LOG_LEVEL_ERROR</pre>
备注	

4.3.17 TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE

宏定义	TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE
依赖项	无。
描述	<p>是否使用高级加密方式，如果是：</p> <pre>#define TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE 1</pre> <p>否则：</p> <pre>#define TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE 0</pre>
备注	暂不支持高级加密。

4.3.18 TUYA_NV_ERASE_MIN_SIZE

宏定义	TUYA_NV_ERASE_MIN_SIZE
依赖项	无。
描述	<p>给 ble sdk 分配的 NV (flash) 最小擦除单位，例如：</p> <pre>#define TUYA_NV_ERASE_MIN_SIZE 4096</pre>
备注	根据 port 层对 NV 接口的实现方式来定义。

4.3.19 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： <pre>#define TUYA_NV_WRITE_GRAN 4</pre> 只能以 4 字节方式写入。
备注	根据 port 层对 NV 接口的实现方式来定义。

4.3.20 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： <pre>#define TUYA_NV_START_ADDR 0x1000</pre>
备注	

4.3.21 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： <pre>#define TUYA_NV_AREA_SIZE (4*TUYA_NV_ERASE_MIN_SIZE)</pre> 必须是 TUYA_NV_ERASE_MIN_SIZE 的倍数。
备注	

5 API 介绍

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。

5.1 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	This function must be called from within the main loop. It will execute all events scheduled since the last time it was called.

示例：例如在 nrf52832 平台下的调用位置如下图所示：

```

/**@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)
{
    // app_sched_execute();
    tuya_ble_main_tasks_exec();

    if (NRF_LOG_PROCESS() == false)
    {
        //app_uart_close();
        nrf_pwr_mgmt_run();
        //uart_init();
    }

    // UNUSED_RETURN_VALUE(NRF_LOG_PROCESS());
    // nrf_pwr_mgmt_run();
}

```

图 5-1 tuya_ble_main_tasks_exec 调用示例

5.2 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 ble 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	This function must be called from where the ble data is received.

示例 (nrf52832 示例 demo):

```

/**@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)
    {
        tuyu_ble_gatt_receive_data((uint8_t*) (p_evt->params.rx_data.p_data), p_evt->params.rx_data.length);
        //TUYA_BLE_HEXDUMP("nus_commdata_handler :", 20, (uint8_t*) (p_evt->params.rx_data.p_data), p_evt->params.rx_data.length);
    }
}

```

图 5- 2 tuyu_ble_gatt_receive_data 示例

5.3 tuyu_ble_common_uart_receive_data

函数名	tuyu_ble_common_uart_receive_data
函数原型	tuya_ble_status_t tuyu_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 数据发送至 ble sdk。
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 tuyu_ble_common_uart_receive_data。

5.4 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 tuya 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。

5.5 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	<p>type [in] : id 类型;</p> <p>len[in] : id 长度。</p> <p>p_buf[in] : id 数据。</p>
返回值	<p>TUYA_BLE_SUCCESS : 发送成功;</p> <p>TUYA_BLE_ERR_INVALID_PARAM: 参数有误。</p> <p>TUYA_BLE_ERR_INTERNAL : 内部错误。</p>
备注	<p>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。</p>

5.6 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	<p>len[in] : 长度。</p> <p>p_buf[in] : login key。</p>
返回值	<p>TUYA_BLE_SUCCESS : 发送成功;</p> <p>TUYA_BLE_ERR_INVALID_PARAM: 参数有误。</p> <p>TUYA_BLE_ERR_INTERNAL : 内部错误。</p>
备注	<p>该 api 主要用于 wifi/ble 双协议设备中 (通过 ble 发送配网信息给 wifi, 设备通过 wifi 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状态)。</p>

5.7 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 wifi/ble 双协议设备中（设备通过 wifi 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）。

5.8 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。

5.9 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 不能工作。

参数说明:

tuya_ble_device_param_t 结构体如下图所示:

```
typedef struct{
    uint8_t device_id_len;    //if ==20,Compressed into 16
    uint8_t device_id[DEVICE_ID_LEN_MAX];
    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 device_vid[DEVICE_VIRTUAL_ID_LEN];
    uint8_t auth_key[AUTH_KEY_LEN];
    uint8_t login_key[LOGIN_KEY_LEN];
    uint8_t bound_flag;
    uint32_t firmware_version; //0x00010102 : v1.1.2
    uint32_t hardware_version;
    uint8_t reserve_1;
    uint8_t reserve_2;
}tuya_ble_device_param_t;
```

图 5- 3 tuya_ble_device_param_t 结构体

各成员含义：

1、device id 和 auth key：device_uuid 和 auth key 是 tuya iot 分配给设备的唯一 id，在产测授权时通过产测工具写入，uuid 和 auth key 一一对应，单 ble 设备经过产测授权后，ble sdk 会自动管理授权信息

(device uuid 和 auth key)，所以一般应用初始化 ble sdk 时将 device id 和 auth key 赋值 0 即可；但是应用在开发时并不会做产测授权，为了方便联调，需要在初始化 sdk 时代入 device id 和 auth key。当然，对于 wifi/ble 双协议设备来讲，device id 和 auth key 是由 wifi 端来管理的，所以在初始化 ble sdk 时也需要代入。

2、product id：产品 id 简称 pid，是在 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，需要在初始化 ble sdk 时代入。

3、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 wifi/ble 多协议设备需要代入。

4、login 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显示代入。

5、firmware_version 和 hardware_version：设备固件版本号和硬件（PCBA）版本号，需要在初始化 sdk 时代入，目前蓝牙 BLE 设备只支持两位版本号例如（0x0100 表示 V1.0）。

图 5-4 为 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[] = "yyyyyyyyyyyyyyyyyy";

#define APP_PRODUCT_ID "vvvvvvvv"

void tuyu_ble_app_init(void)
{
    tuyu_ble_device_param_t device_param = {0};
    device_param.device_id_len = 16;
    memcpy(device_param.auth_key, (void *)auth_key_test, AUTH_KEY_LEN);
    memcpy(device_param.device_id, (void *)device_id_test, DEVICE_ID_LEN);
    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;

    tuyu_ble_sdk_init(&device_param);
    tuyu_ble_callback_queue_register(tuya_cb_handler);

    tuyu_ota_init();
}

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

    // Initialize.
    uart_init();
    //log_init();
    app_log_init();
    timers_init();
    buttons_leds_init(&erase_bonds);
    power_management_init();
    ble_stack_init();
    // scheduler_init();
    gap_params_init();
    gatt_init();
    services_init();
    advertising_init();
    conn_params_init();
    // app_rtc_config();
    tuyu_ble_app_init();
    advertising_start();

    NRF_LOG_INFO("app version : %s\r\n", TY_APP_VER_STR);

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

```

图 5- 4 tuyu ble sdk 初始化示例

5.10 tuya_ble_dp_data_report

函数名	tuya_ble_dp_data_report
函数原型	tuya_ble_status_t tuya_ble_dp_data_report(uint8_t *p_data, uint32_t len);
功能概述	上报 dp 点数据。
参数	p_data [in] : dp 点数据。 Len[in] : 数据长度，最大不能超过 TUYA_BLE_REPORT_MAX_DP_DATA_LEN。
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	应用代码通过调用该函数上报 dp 点数据到 app。

参数说明:

1、tuya iot 平台是以 dp 点的方式管理数据，任何设备产生的数据都需要抽象为 dp 点的形式，一个完整的 dp 点数据由四部分组成（具体参考 TUYA 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: 1 个字节或者两个字节，目前蓝牙仅支持一个字节，即单个 dp 点数据最长 255 个字节。

Dp_data: 数据，dp_len 个字节。

2、该 dp 点上报函数的参数 p_data 指向的数据必须以下表格式组装上报：

表 5- 1 DP 点数据组装格式

Dp 点 1 的数据				~	Dp 点 n 的数据			
1	2	3	4~	~	n	n+1	n+2	n+3~
Dp_id	Dp_type	Dp_len	Dp_data	~	Dp_id	Dp_type	Dp_len	Dp_data

3、调用该函数时，参数 len 的最大长度为 TUYA_BLE_REPORT_MAX_DP_DATA_LEN（目前为 255+3）。

5.11 tuy_ble_dp_data_with_time_report

函数名	tuya_ble_dp_data_with_time_report
函数原型	tuya_ble_status_t tuy_ble_dp_data_with_time_report(uint32_t timestamp, uint8_t *p_data, uint32_t len);
功能概述	上报带时间戳的 dp 点数据。
参数	timestamp[in]：四字节 unix 时间戳。 p_data [in]：dp 点数据。 Len[in]：数据长度，最大不能超过 TUYA_BLE_REPORT_MAX_DP_DATA_LEN。
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、应用代码通过调用该函数带时间戳上报 dp 点数据到 app。 2、一般是离线缓存的数据才需要带时间戳上报。

5.12 tuya_ble_dp_data_with_time_ms_string_report

函数名	tuya_ble_dp_data_with_time_ms_string_report
函数原型	tuya_ble_status_t tuya_ble_dp_data_with_time_ms_string_report(uint8_t *time_string, uint8_t *p_data, uint32_t len)
功能概述	上报带字符串格式时间的 dp 点数据。
参数	time_string[in] : 13 字节 ms 级字符串格式时间。 p_data [in] : dp 点数据。 Len[in] : 数据长度，最大不能超过 TUYA_BLE_REPORT_MAX_DP_DATA_LEN。
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、应用代码通过调用该函数带字符串格式时间上报 dp 点数据到 app。 2、一般是离线缓存的数据才需要带时间上报。 3、13 字节 ms 级时间字符串，例如"0000000123456"，不够 13 字节前面补字符 0。

5.13 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	应用代码需要在芯片平台 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。

5.14 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。

图 5-5 为 nrf52832 平台调用 tuya_ble_connected_handler 和 tuya_ble_disconnected_handler 的示例。

5.15 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_TRANSMISSION_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 app 。 2、透传的数据格式是由设备和手机 app 端协商的， ble sdk 不做解析。


```
/**@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,
            };
        }
    }
};
```

图 5- 5 tuyable_connected_handler 示例

5.16 tuyable_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。

5.17 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	Wifi 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 wifi/ble 多协议设备。

5.18 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 wifi/ble 多协议设备。

5.19 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 wifi/ble 多协议设备。

5.20 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 wifi/ble 多协议设备。

5.21 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	tuya_ble_connect_status_t
备注	<pre>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;</pre>

5.22 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	对于定义外部按键重置功能的设备, 按键触发重置后应用程序需要调用该函数通知 ble sdk 重置相关信息, 例如清除绑定信息。

5.23 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 13 字节 ms 级字符串格式的时间; 1-请求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。

5.24 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。

5.25 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。
返回值	0：发送成功； 1：失败。
备注	typedef struct { void *data; void (*custom_event_handler)(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。

图 5-6 是 tuya_ble_custom_event_send 应用示例。

5.26 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 tuyab_handler(tuya_ble_cb_evt_param_t* event);
```

```
tuya_ble_callback_queue_register(tuyab_handler);
```

```
#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_data;  
    TUYA_BLE_LOG_DEBUG("custom event id = %d", evt_id);  
    switch (evt_id)  
    {  
        case APP_CUSTOM_EVENT_1:  
            event_data = (custom_data_type_t *)data;  
            TUYA_BLE_LOG_HEXDUMP_DEBUG("received APP_CUSTOM_EVENT_1 data:", event_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)  
{  
    tuyab_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;  
    tuyab_custom_event_send(event);  
}
```

图 5- 6 tuyab_custom_event_send 示例

5.27 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。

图 5-7 展示了如何使用该函数。


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

    while (true)
    {
        if (os_msg_rcv(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_WITH_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;
            }

            tuyu_ble_event_response(&event);
        }
    }
}
```

图 5- 7 tuyu_ble_event_response 示例

6 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) 向设备应用发送消息 (状态、数据等), 图 5-7 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图 5-7 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, 如图 6-1 所示, 本章节主要介绍各种 EVENT 的含义。

6.1 TUYA_BLE_CB_EVT_CONNECTE_STATUS

Event	TUYA_BLE_CB_EVT_CONNECTE_STATUS
对应数据结构	<pre>typedef enum{ UNBONDING_UNCONN = 0, UNBONDING_CONN, BONDING_UNCONN, BONDING_CONN, BONDING_UNAUTH_CONN, UNBONDING_UNAUTH_CONN, UNKNOW_STATUS }tuya_ble_connect_status_t;</pre>
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	

```
/*处理ble sdk消息的call back函数*/
static void tuyu_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 tuyu_ble_app_init(void)
{
    device_param.device_id_len = 16;
    memcpy(device_param.auth_key, (void *)auth_key_test, AUTH_KEY_LEN);
    memcpy(device_param.device_id, (void *)device_id_test, DEVICE_ID_LEN);
    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;

    tuyu_ble_sdk_init(&device_param);
    tuyu_ble_callback_queue_register(tuyu_cb_handler);

    tuyu_ota_init();
}
```

图 6- 1 call back 函数示例

6.2 TUYA_BLE_CB_EVT_DP_WRITE

Event	TUYA_BLE_CB_EVT_DP_WRITE
对应数据结构	<pre>typedef struct{ uint8_t *p_data; uint16_t data_len; }tuya_ble_dp_write_data_t;</pre>
描述	Ble sdk 收到的手机 app 发送的 dp 点数据。
备注	格式见 5.10 节介绍。

6.3 TUYA_BLE_CB_EVT_DP_QUERY

Event	TUYA_BLE_CB_EVT_DP_QUERY
对应数据结构	<pre>typedef struct{ uint8_t *p_data; uint16_t data_len; }tuya_ble_dp_query_data_t;</pre>
描述	Ble sdk 收到的手机 app 发送的要查询的 dp 点。
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{0x01,0x02,0x03}，表示要查询 dp_id=1，dp_id=2,dp_id=3 的 3 个 dp 点数据。

6.4 TUYA_BLE_CB_EVT_OTA_DATA

Event	TUYA_BLE_CB_EVT_OTA_DATA
对应数据结构	<pre>typedef struct{ tuya_ble_ota_data_type_t type; uint16_t data_len; uint8_t *p_data;</pre>

	}tuya_ble_ota_data_t;
描述	Ble sdk 收到的手机 app 发送的 ota 数据。
备注	具体见 OTA 章节介绍。

6.5 TUYA_BLE_CB_EVT_NETWORK_INFO

Event	TUYA_BLE_CB_EVT_NETWORK_INFO
对应数据结构	<pre>typedef struct{ uint16_t data_len; //include '\0' uint8_t *p_data; }tuya_ble_network_data_t;</pre>
描述	Ble sdk 收到的手机 app 发送的 wifi 配网信息，例如：{"wifi_ssid":"tuya","password":"12345678","token":"xxxxxxxxxx"} "
备注	只适用于 wifi/ble 双协议设备。

6.6 TUYA_BLE_CB_EVT_WIFI_SSID

Event	TUYA_BLE_CB_EVT_WIFI_SSID
对应数据结构	<pre>typedef struct{ uint16_t data_len;//include '\0' uint8_t *p_data; }tuya_ble_wifi_ssid_data_t;</pre>
描述	Ble sdk 收到的手机 app 发送的 wifi 配网信息，例如：{"wifi_ssid":"tuya","password":"12345678"} "
备注	只适用于 wifi/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 WIFI ssid 更新。

6.7 TUYA_BLE_CB_EVT_TIME_STAMP

Event	TUYA_BLE_CB_EVT_TIME_STAMP
对应数据结构	<pre>typedef struct{ uint8_t timestamp_string[14]; int16_t time_zone; //actual time zone Multiply by 100. }tuya_ble_timestamp_data_t;</pre>
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 123456ms，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。

6.8 TUYA_BLE_CB_EVT_TIME_NORMAL

Event	TUYA_BLE_CB_EVT_TIME_NORMAL
对应数据结构	<pre>typedef struct{ uint16_t nYear; 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;</pre>
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	

6.9 TUYA_BLE_CB_EVT_DATA_PASSTHROUGH

Event	TUYA_BLE_CB_EVT_DATA_PASSTHROUGH
对应数据结构	<pre>typedef struct{ uint16_t data_len; uint8_t *p_data; }tuya_ble_passthrough_data_t;</pre>
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，数据格式由应用和手机 app 协商定义。

6.10 TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE
对应数据结构	<pre>typedef struct{ uint8_t status; }tuya_ble_dp_data_report_response_t;</pre>
描述	设备应用程序调用 <code>tuya_ble_dp_data_report()</code> 函数发送完 dp 点数据后，如果需要确认是否发送成功，就需要等待该 EVENT， <code>status=0</code> 表示成功，其他失败，设备应用程序如果等待发送结果同时必须加入超时机制，不能无限制等待。
备注	

6.11 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	<pre>typedef struct{ uint8_t status; }tuya_ble_dp_data_with_time_report_response_t;</pre>
描述	设备应用程序调用 <code>tuya_ble_dp_data_with_time_report()</code> 和 <code>tuya_ble_dp_data_with_time_ms_string_report()</code> 函数发送完 dp 点数据后，如果需要确认是否发送成功，就需要等待该 EVENT， <code>status=0</code> 表示成功，其他

	失败，设备应用程序如果等待发送结果同时必须加入超时机制，不能无限制等待。
备注	

6.12 TUYA_BLE_CB_EVT_UNBOUND

Event	TUYA_BLE_CB_EVT_UNBOUND
对应数据结构	<pre>typedef struct{ uint8_t data; }tuya_ble_unbound_data_t;</pre>
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	

6.13 TUYA_BLE_CB_EVT_ANOMALY_UNBOUND

Event	TUYA_BLE_CB_EVT_ANOMALY_UNBOUND
对应数据结构	<pre>typedef struct{ uint8_t data; }tuya_ble_anomaly_unbound_data_t;</pre>
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	

6.14 TUYA_BLE_CB_EVT_DEVICE_RESET

Event	TUYA_BLE_CB_EVT_DEVICE_RESET
对应数据结构	<pre>typedef struct{ uint8_t data; }tuya_ble_device_reset_data_t;</pre>

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。

6.15 TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID

Event	TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID
对应数据结构	<pre>typedef struct{ uint8_t login_key_len; uint8_t vid_len; uint8_t login_key[LOGIN_KEY_LEN]; uint8_t vid[DEVICE_VIRTUAL_ID_LEN]; }tuya_ble_login_key_vid_data_t;</pre>
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	单 BLE 设备无需处理该信息。

7 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。

7.1 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。

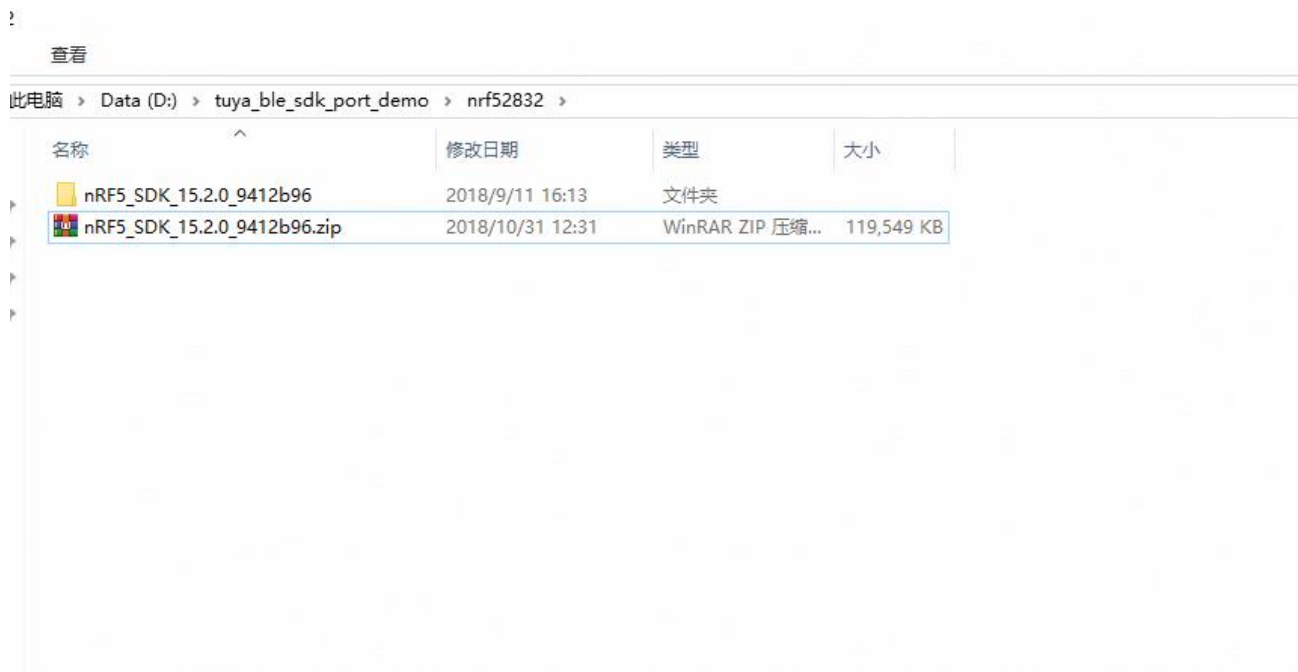


图 7- 1 nrf52832 移植示例图 1

- 3、依次进入 examples->ble_peripheral 目录，如图 7-2 所示。
- 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。
- 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 3.2 节要求的 tuya ble service，修改 main.c 文件中广播相关的代码，按照 3.4 节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。
- 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。

查看

电脑 > Data (D:) > tuyu_ble_sdk_port_demo > nrf52832 > nRF5_SDK_15.2.0_9412b96 > examples > ble_peripheral

名称	修改日期	类型	大小
ble_app_alert_notification	2018/9/8 12:38	文件夹	
ble_app_ancs_c	2018/9/8 12:39	文件夹	
ble_app_beacon	2018/9/8 12:41	文件夹	
ble_app_blinky	2018/9/8 12:42	文件夹	
ble_app_bms	2018/9/8 12:44	文件夹	
ble_app_bps	2018/9/8 12:46	文件夹	
ble_app_buttonless_dfu	2018/9/8 12:06	文件夹	
ble_app_cscs	2018/9/8 12:48	文件夹	
ble_app_cts_c	2018/9/8 12:49	文件夹	
ble_app_eddystone	2018/9/8 12:52	文件夹	
ble_app_gatts_c	2018/9/8 12:54	文件夹	
ble_app_gls	2018/9/8 12:56	文件夹	
ble_app_hids_keyboard	2018/9/8 12:59	文件夹	
ble_app_hids_mouse	2018/9/8 13:01	文件夹	
ble_app_hrs	2018/9/8 13:03	文件夹	
ble_app_hrs_freertos	2018/9/8 13:07	文件夹	
ble_app_hts	2018/9/8 13:08	文件夹	
ble_app_ias_c	2018/9/8 13:10	文件夹	
ble_app_ipsp_acceptor	2018/9/8 13:11	文件夹	
ble_app_proximity	2018/9/8 13:12	文件夹	
ble_app_pwr_profiling	2018/9/8 13:14	文件夹	
ble_app_rscs	2018/9/8 13:15	文件夹	
ble_app_template	2018/9/8 13:17	文件夹	
ble_app_uart	2018/9/8 13:18	文件夹	
experimental	2018/9/8 11:15	文件夹	

图 7- 2 nrf52832 移植示例图 2

查看

电脑 > Data (D:) > tuyu_ble_sdk_port_demo > nrf52832 > nRF5_SDK_15.2.0_9412b96 > examples > ble_peripheral >

名称	修改日期	类型	大小
ble_app_alert_notification	2018/9/8 12:38	文件夹	
ble_app_ancs_c	2018/9/8 12:39	文件夹	
ble_app_beacon	2018/9/8 12:41	文件夹	
ble_app_blinky	2018/9/8 12:42	文件夹	
ble_app_bms	2018/9/8 12:44	文件夹	
ble_app_bps	2018/9/8 12:46	文件夹	
ble_app_buttonless_dfu	2018/9/8 12:06	文件夹	
ble_app_cscs	2018/9/8 12:48	文件夹	
ble_app_cts_c	2018/9/8 12:49	文件夹	
ble_app_eddystone	2018/9/8 12:52	文件夹	
ble_app_gatts_c	2018/9/8 12:54	文件夹	
ble_app_gls	2018/9/8 12:56	文件夹	
ble_app_hids_keyboard	2018/9/8 12:59	文件夹	
ble_app_hids_mouse	2018/9/8 13:01	文件夹	
ble_app_hrs	2018/9/8 13:03	文件夹	
ble_app_hrs_freertos	2018/9/8 13:07	文件夹	
ble_app_hts	2018/9/8 13:08	文件夹	
ble_app_ias_c	2018/9/8 13:10	文件夹	
ble_app_ipsp_acceptor	2018/9/8 13:11	文件夹	
ble_app_proximity	2018/9/8 13:12	文件夹	
ble_app_pwr_profiling	2018/9/8 13:14	文件夹	
ble_app_rscs	2018/9/8 13:15	文件夹	
ble_app_template	2018/9/8 13:17	文件夹	
ble_app_uart	2018/9/8 13:18	文件夹	
experimental	2018/9/8 11:15	文件夹	
tuya_ble_standard_nordic	2020/3/11 17:50	文件夹	

图 7- 3 nrf52832 移植示例图 3

bootloader_project	2019/11/26 19:58	文件夹	
extern_components	2020/3/6 15:26	文件夹	
hex	2019/11/20 11:50	文件夹	
pca10040	2019/11/20 11:50	文件夹	
SDK15.2.0_tuya_patch	2019/12/4 17:25	文件夹	
tuya_ble_app	2020/3/11 15:37	文件夹	
tuya_ble_sdk	2020/3/11 15:04	文件夹	
tuya_ble_services	2019/11/20 20:24	文件夹	
flash.c	2019/12/26 15:00	C Source File	9 KB
flash.h	2019/7/18 21:20	C/C++ Header F...	1 KB
main.c	2020/3/10 17:12	C Source File	31 KB
main.h	2019/12/4 17:26	C/C++ Header F...	1 KB
rtc.c	2019/11/20 20:44	C Source File	4 KB
rtc.h	2019/5/20 11:18	C/C++ Header F...	1 KB

图 7- 4 nrf52832 移植示例图 4

- 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。
- 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。

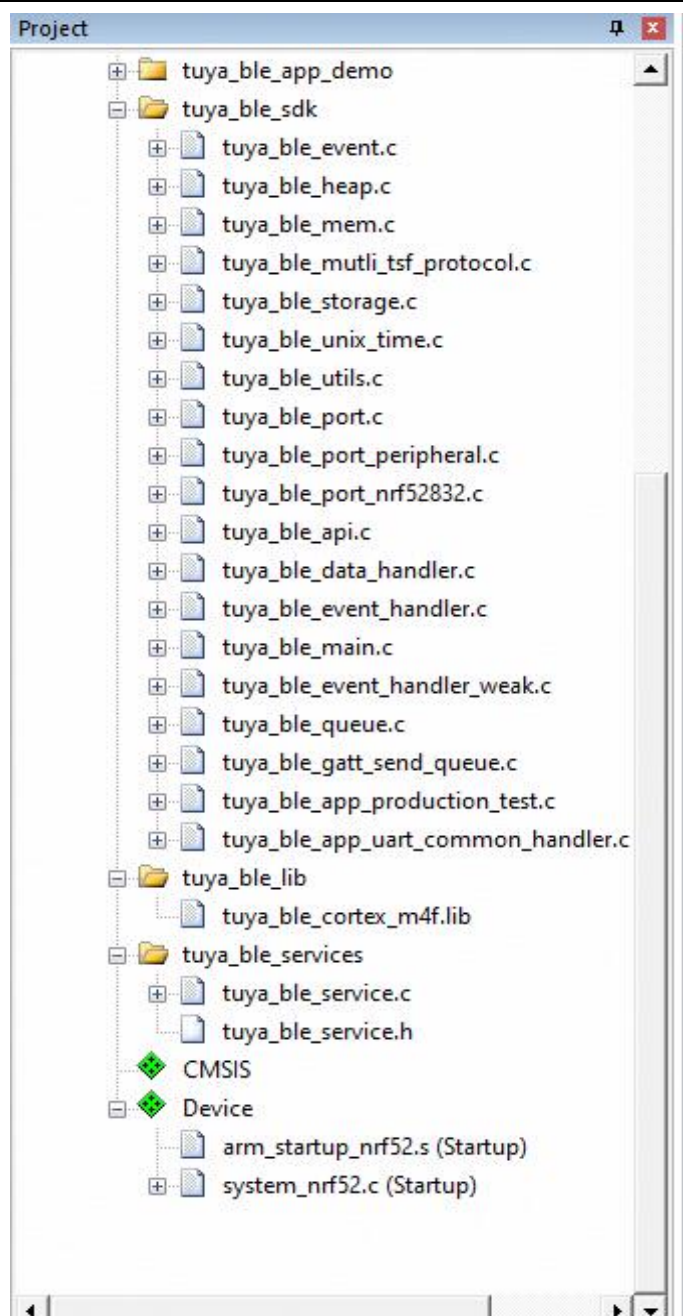


图 7- 5 nrf52832 移植示例图 5

- 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。

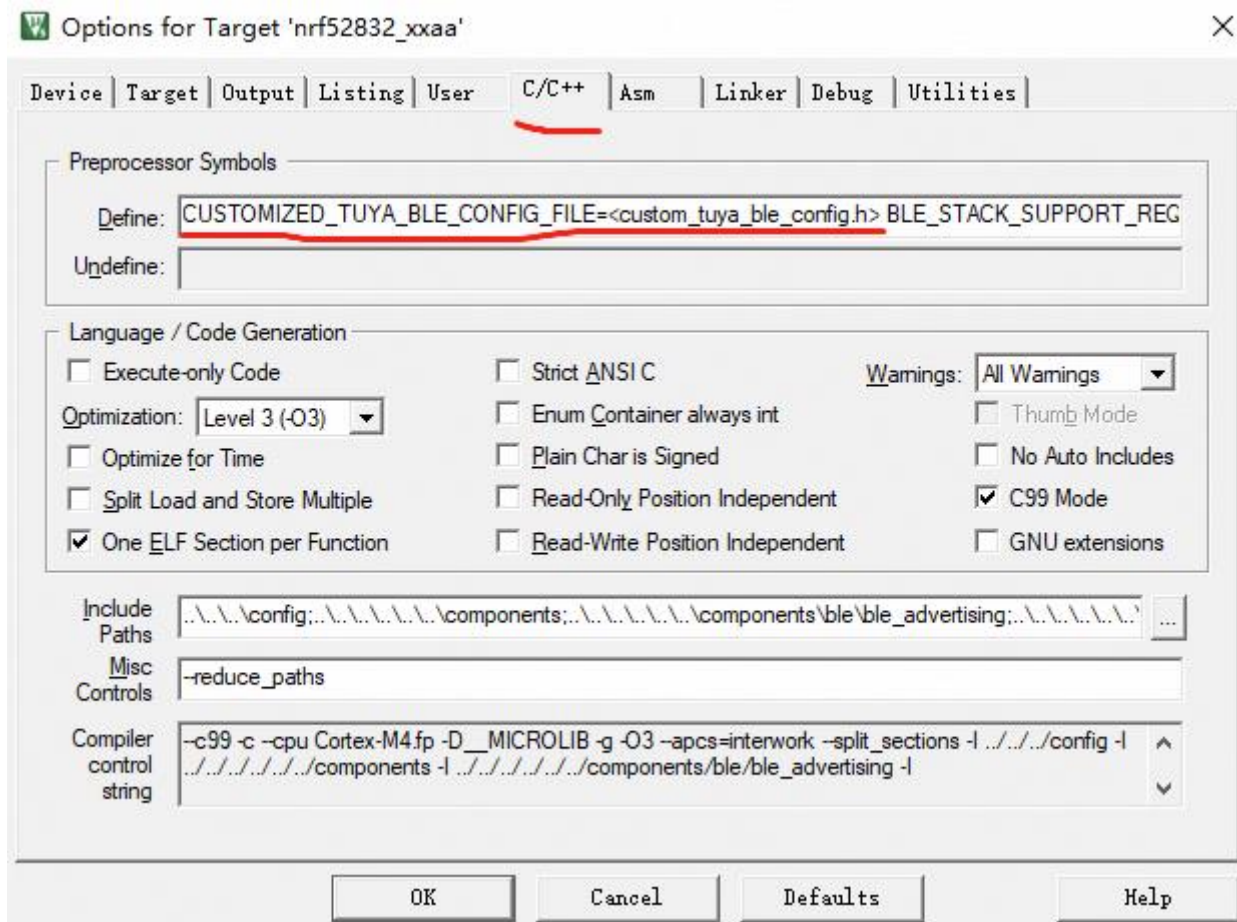


图 7- 6 nrf52832 移植示例图 6



图 7- 7 nrf52832 移植示例图 7

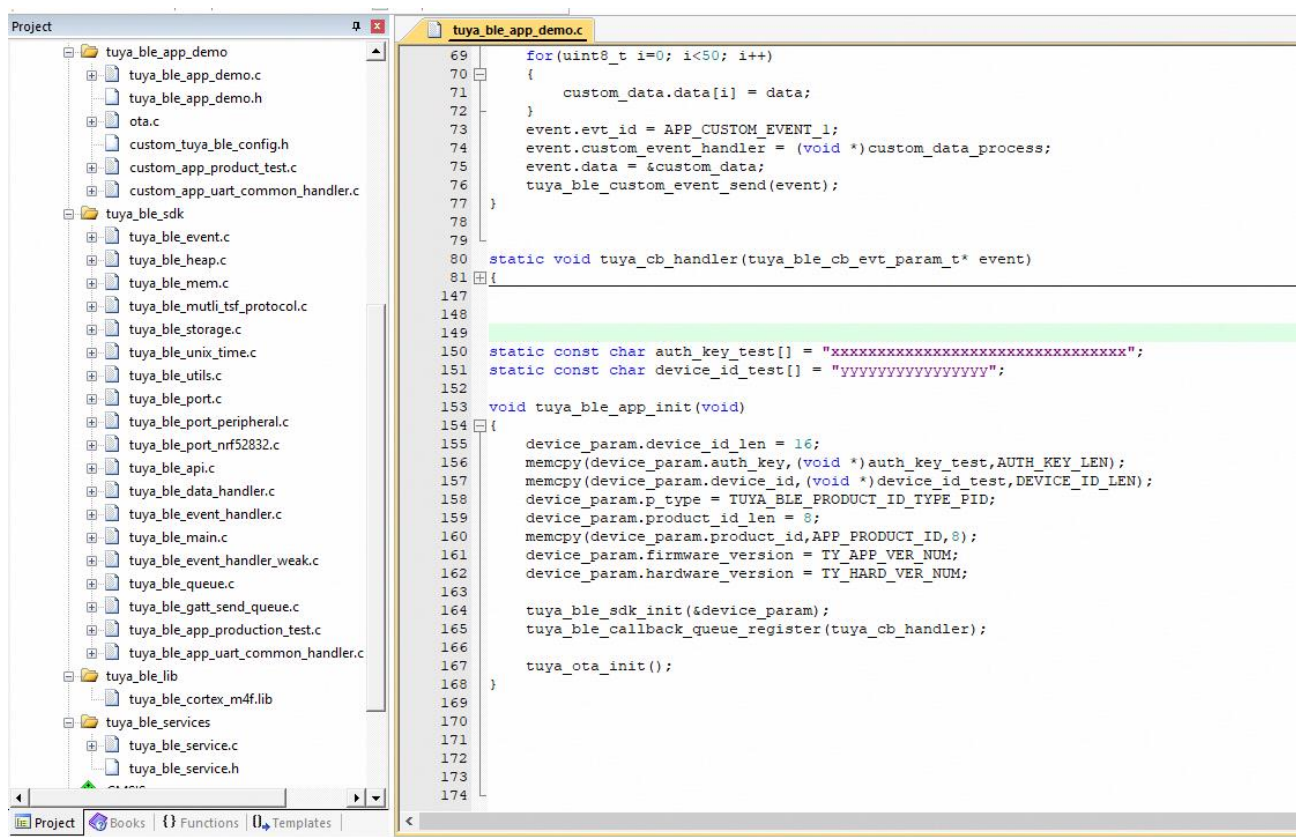
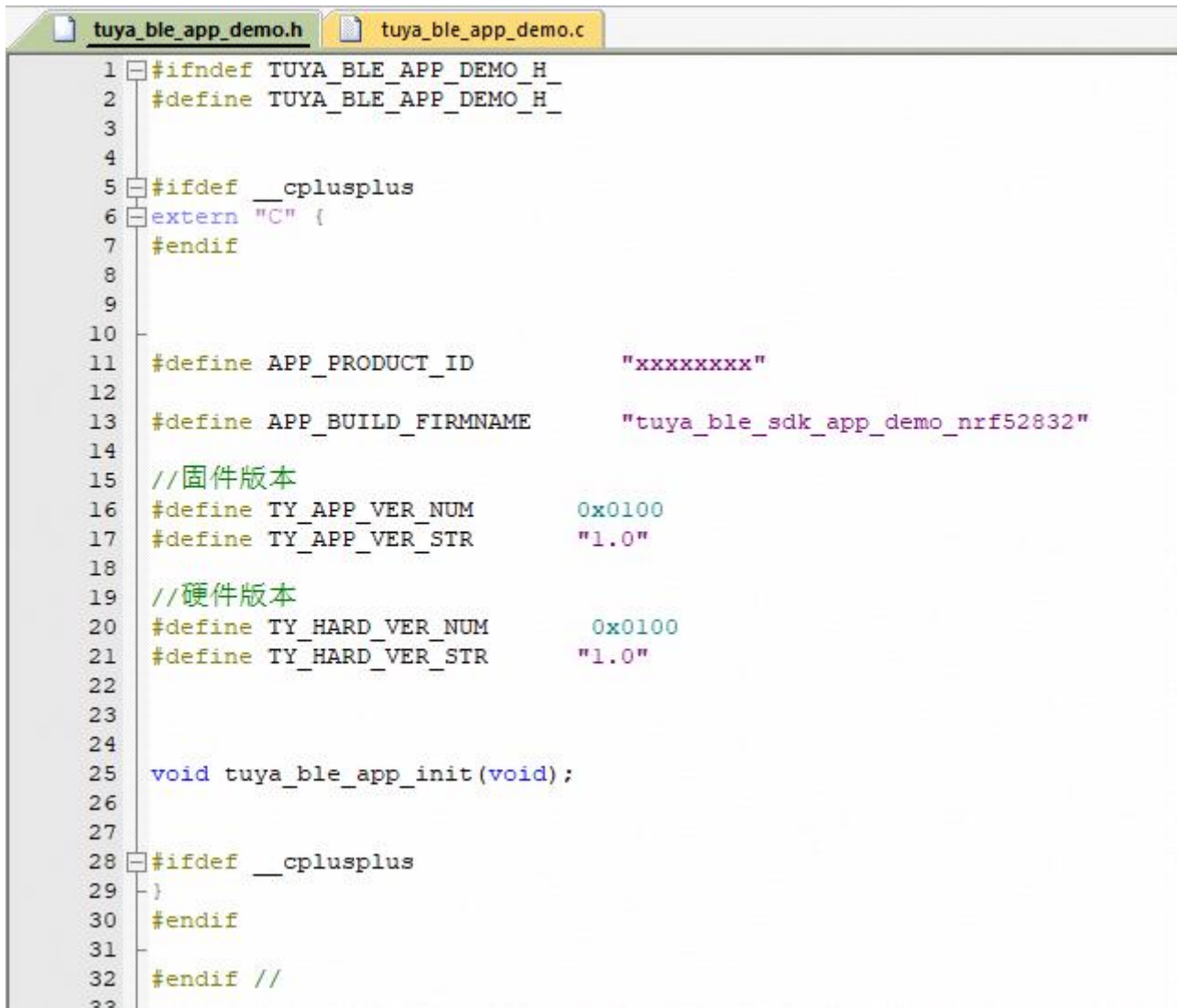


图 7- 8 nrf52832 移植示例图 8

- 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，“xxxxxxx”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）。
- 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。
- 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。



```
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 //
```

图 7- 9 nrf52832 移植示例图 9


```
/**@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 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

7.2 其他平台移植示例

请参考各平台完整示例 demo.

8 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 tuyu 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 8.2 节，OTA 响应数据通过 `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` 函数发送。

8.1 OTA 升级流程

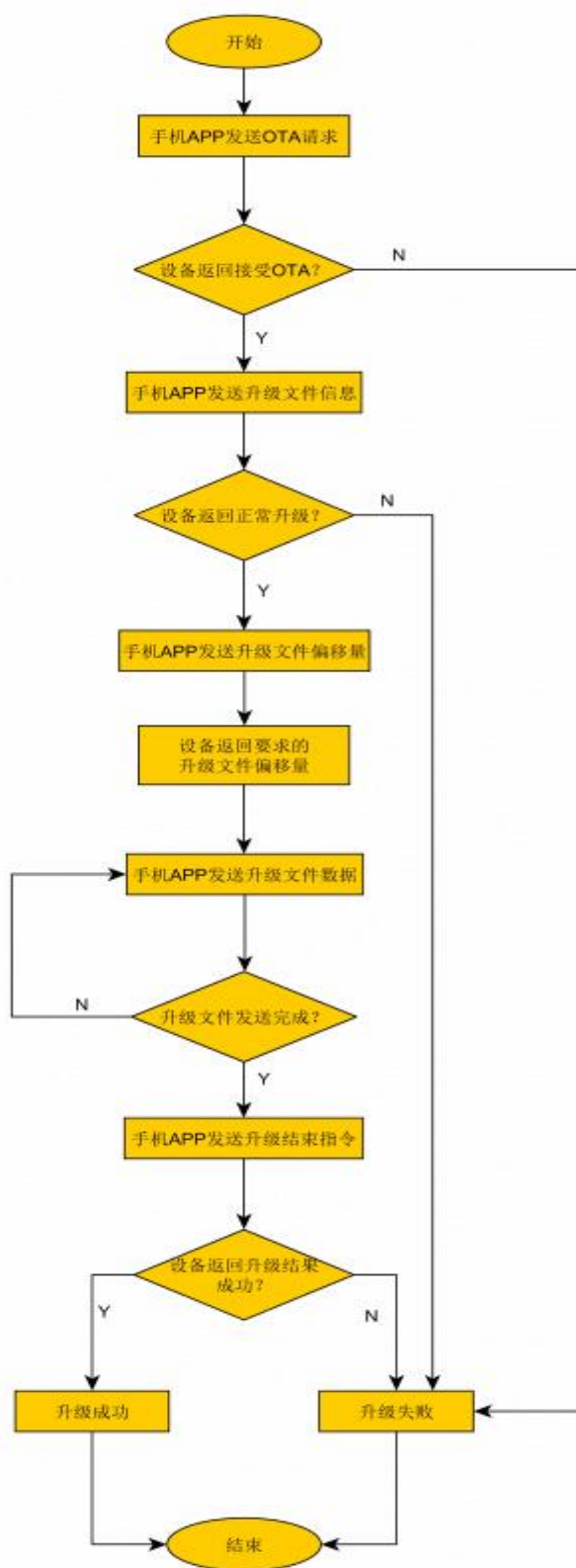


图 8- 1 OTA 升级流程

8.2 OTA 升级协议

8.2.1 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
```

8.2.2 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。**

8.2.3 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。

8.2.4 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。

8.2.5 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

8.2.6 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。

8.3 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`, 数据格式见 8.2 节, 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 tuyu_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 tuyu ble connctet 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);
            tuyu_ble_dp_data_report(dp_data_array,dp_data_len);
            custom_evt_l_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_WITH_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");
            tuyu_ble_dp_data_report(dp_data_array,dp_data_len);
            break;
        case TUYA_BLE_CB_EVT_OTA_DATA:
            tuyu_ota_proc(event->ota_data.type,event->ota_data.p_data,event->ota_data.data_len);
            break;
        case TUYA_BLE_CB_EVT_NETWORK_TUEO:
    }
}

```

图 8- 2 OTA 数据接口

```

void tuyu_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:
            tuyu_ota_start_req(recv_data,recv_len);
            break;
        case TUYA_BLE_OTA_FILE_INFO:
            tuyu_ota_file_info_req(recv_data,recv_len);
            break;
        case TUYA_BLE_OTA_FILE_OFFSET_REQ:
            tuyu_ota_offset_req(recv_data,recv_len);
            break;
        case TUYA_BLE_OTA_DATA:
            tuyu_ota_data_req(recv_data,recv_len);
            break;
        case TUYA_BLE_OTA_END:
            tuyu_ota_end_req(recv_data,recv_len);
            break;
        default:
            break;
    }
}

```

图 8- 3 OTA 处理函数参考

9 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。

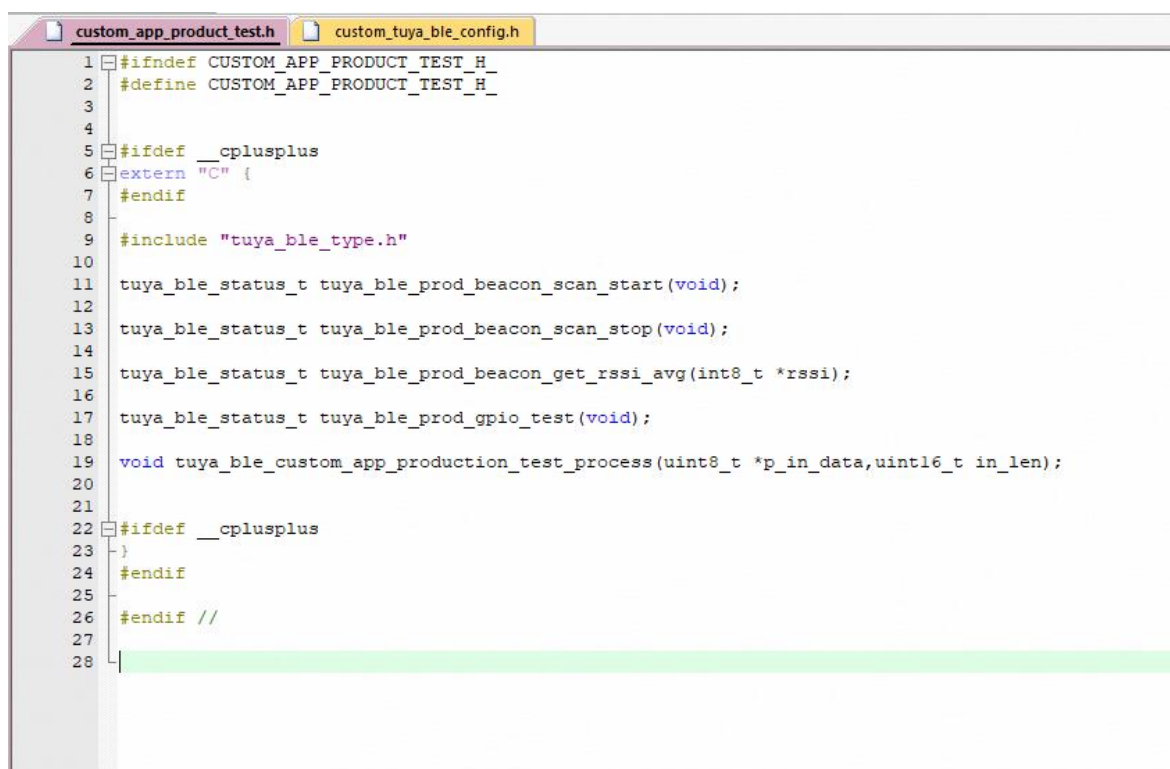
产测分为“通用产测授权”和“通用整机产测”，“通用整机产测协议”是“通用产测授权协议”的子集，“通用产测授权”主要包括烧录授权信息、GPIO 测试以及 RSSI 测试；通用整机产测包含产品定制附加的一些测试，具体协议格式请参考《蓝牙通用产测授权协议》和《蓝牙通用整机产测协议》。

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 tuyable_status_t tuyable_prod_beacon_scan_start(void)
19 {
20     //
21     return TUYA_BLE_SUCCESS;
22 }
23
24 tuyable_status_t tuyable_prod_beacon_scan_stop(void)
25 {
26     //
27     return TUYA_BLE_SUCCESS;
28 }
29
30 tuyable_status_t tuyable_prod_beacon_get_rssi_avg(int8_t *rssi)
31 {
32     //
33     *rssi = -30;
34     return TUYA_BLE_SUCCESS;
35 }
36
37 tuyable_status_t tuyable_prod_gpio_test(void)
38 {
39     //
40     return TUYA_BLE_SUCCESS;
41 }
42
43 void tuyable_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
```

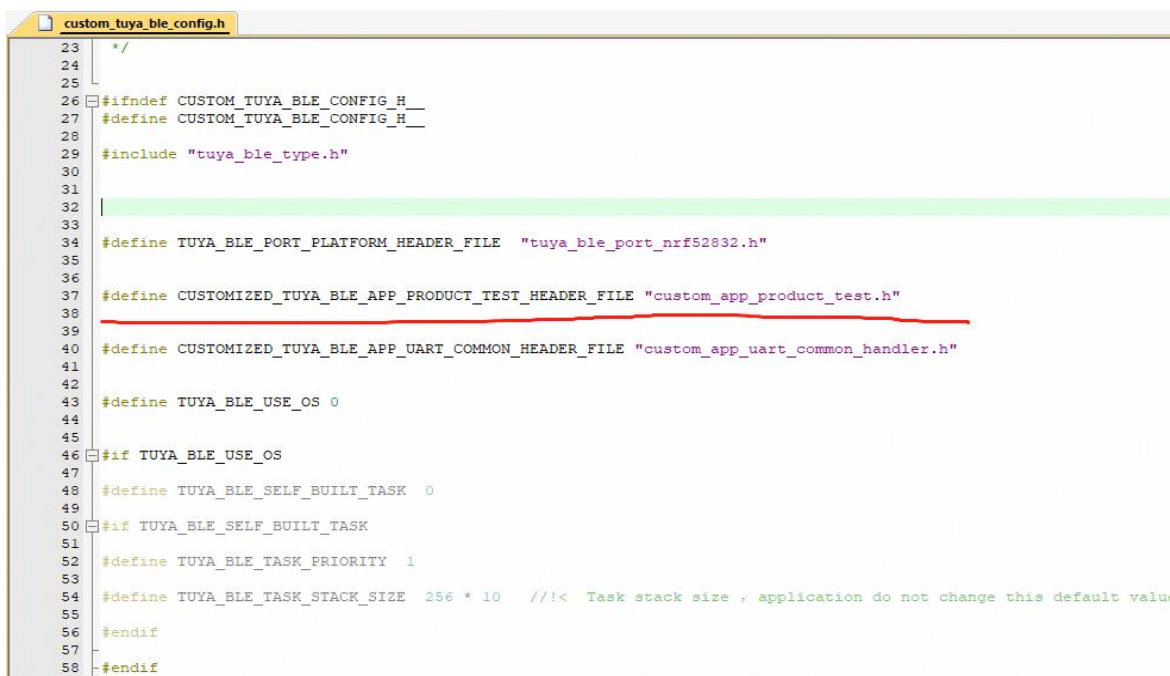
图 9- 1 应用实现的相关产测函数源文件示例



The image shows a code editor window with two tabs: 'custom_app_product_test.h' and 'custom_tuya_ble_config.h'. The 'custom_app_product_test.h' tab is active, displaying the following C code:

```
1 #ifndef CUSTOM_APP_PRODUCT_TEST_H_
2 #define CUSTOM_APP_PRODUCT_TEST_H_
3
4
5 #ifdef __cplusplus
6 extern "C" {
7 #endif
8
9 #include "tuya_ble_type.h"
10
11 tuy_ble_status_t tuy_ble_prod_beacon_scan_start(void);
12
13 tuy_ble_status_t tuy_ble_prod_beacon_scan_stop(void);
14
15 tuy_ble_status_t tuy_ble_prod_beacon_get_rssi_avg(int8_t *rssi);
16
17 tuy_ble_status_t tuy_ble_prod_gpio_test(void);
18
19 void tuy_ble_custom_app_production_test_process(uint8_t *p_in_data, uint16_t in_len);
20
21
22 #ifdef __cplusplus
23 }
24 #endif
25
26 #endif //
```

图 9- 2 应用实现的相关产测函数头文件示例



The image shows a code editor window with a tab 'custom_tuya_ble_config.h'. The file contains the following C code:

```
23 */
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
46 #if TUYA_BLE_USE_OS
47 #define TUYA_BLE_SELF_BUILT_TASK 0
48
49
50 #if TUYA_BLE_SELF_BUILT_TASK
51 #define TUYA_BLE_TASK_PRIORITY 1
52 #define TUYA_BLE_TASK_STACK_SIZE 256 * 10 //!< Task stack size , application do not change this default value
53 #endif
54 #endif
55
56 #endif
```

图 9- 3 应用配置文件引用自定义产测文件示例

附录