



BLE SDK Guide

Version: 20240612

Contents

1 Overview of tuyu ble sdk	2
1.1 Framework	2
1.2 OS compatibility	3
1.3 Event queue	3
1.4 Directories	4
2 The concepts of tuyu ble service	5
2.1 MTU	5
2.2 Broadcast data format	5
3 How to port and configure tuyu ble sdk	8
3.1 Reference of porting APIs	8
3.2 Reference of configuration APIs	39
4 API reference	53
5 The callback event of tuyu ble sdk	93
6 Example of SDK port in nrf52832	106
7 OTA protocol	116
7.1 OTA upgrade process	116
7.2 OTA upgrade protocol	118
7.3 OTA upgrade APIs	123
8 Reference of production testing APIs	125

This topic was no longer updated as of August 24, 2021. If you want to see the updated content, please refer to [Bluetooth Device Access](#) of TuyaOS.

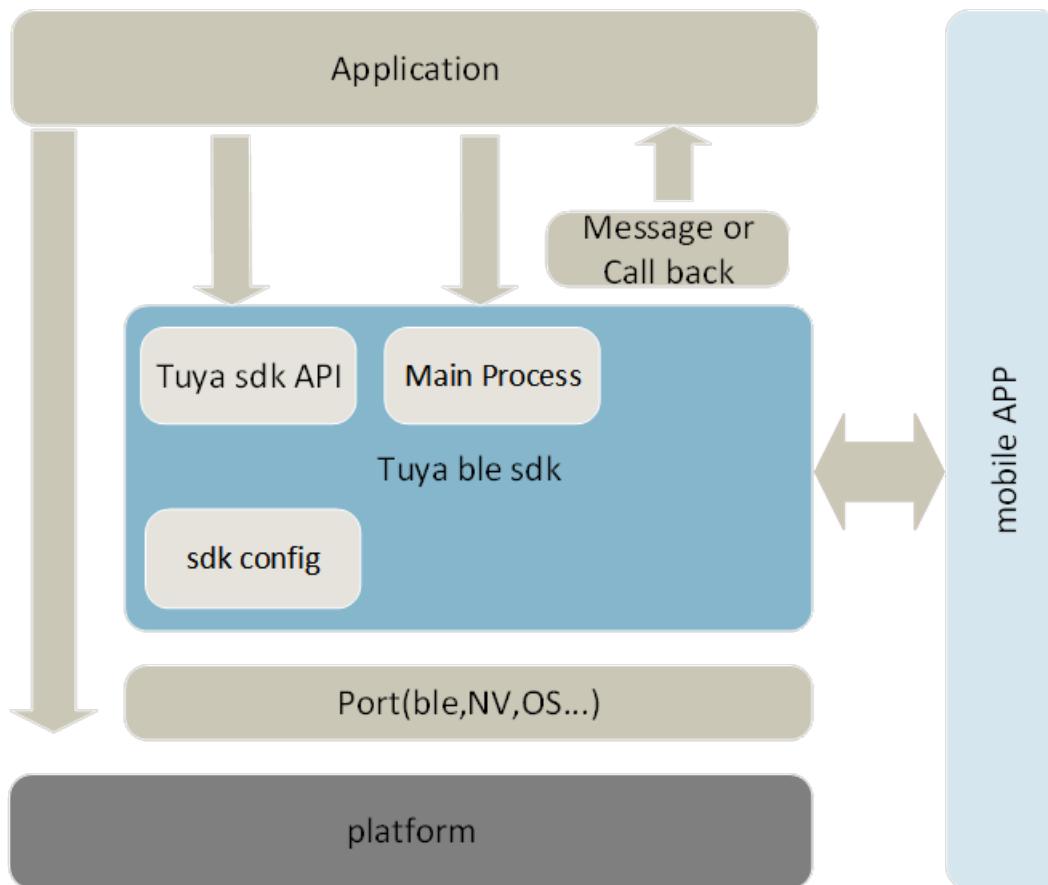
The tuya ble sdk encapsulates the communication protocol with Tuya Smart mobile App and implements event scheduling abilities. The device using tuya ble sdk does not need to care about the specific communication protocol implementation details. It can be interconnected with Tuya Smart App by calling the API and callback provided by the tuya ble sdk.

This topic gives details of the components, porting instruction, SDK configuration, API description, and usage of tuya ble sdk.

1 Overview of tuyu ble sdk

1.1 Framework

The following figure shows the Application framework based on tuyu ble sdk:



- **platform**: the chip platform. The chip and protocol stack are maintained by the chip company.
- **Port**: the abstract interfaces needed by the tuyu ble sdk. You must implement them according to the chip-specific platform.
- **tuya ble sdk**: encapsulates the communication protocol of Tuyu BLE and provides the service interface to develop Tuyu BLE devices.
- **Application**: your application, built by using tuyu ble sdk.
- **Tuya sdk API**: implements BLE related management, communication, and so

forth. The calls of API are based on asynchronous messages, and the result of API will be notified to the Application of device by message or call back.

- **sdk config:** by setting the macro in the configuration file, you can configure tuya ble sdk to different modes, for example, the general network configuration mode applicable to multi-protocol devices, Bluetooth singlepoint devices mode, ECDH key based encryption method, whether to use OS, and so forth.
- **Main process function:** the engine of tuya ble sdk, to which the Application will call all the time. If the platform architecture has an OS, The tuya ble sdk will automatically create a task to run the main process based on the OS related interface provided by the port layer. If the platform does not have an OS, the device Application needs to be called circularly.
- **Message or Call back:** SDK sends the data of status, data, and others to device Application through call back function registered by device Application or messages.

1.2 OS compatibility

The tuya ble sdk can run on OS based chip platform besides Linux. If an OS is used, the API requests are based on asynchronous messages. When tuya ble sdk is initialized, the SDK automatically creates a task based on ' `tuya_ble_config.h` file to process the message events of the SDK, and creates a message queue to receive the responses of the Application API. The results of the API are notified to the Application of the device in the form of message, so your Application needs to create a message queue and call `tuya_ble_callback_queue_register()` after calling `tuya_ble_sdk_init()` or `tuya_ble_sdk_init_async()` to register the message queue to the SDK.

In the chip platform that has an OS, you can also configure the tuya ble sdk to process messages using the task provided by Application instead of tasks within the tuya ble sdk. By doing so, the Application must implement the outbound message interface at the port layer.

1.3 Event queue

The earlier event takes precedence to leave (FIFO). Event queue caches the messages sent by the Application and platform layer, the event can be API calls, data response from BLE devices, and so forth. The main process function module circularly queries the message queue and takes it out for processing.

1.4 Directories

Directory	Description
app	Stores Applications that managed by the tuya ble sdk, such as Tuya test and production module, general connection modules and so forth.
doc	Help file.
extern_components	External components, for example, the extension for security-specific algorithm.
port	The abstract interfaces which must be implemented by Applications.
sdk	The core code of the tuya ble sdk.
tuya_ble_config.h	The configuration file for tuya ble sdk. However, your Application needs to create another configuration files on demand.
tuya_ble_sdk_version.h	The version file.
README.md	A brief introduction of the tuya ble sdk.
tuya_ble_sdk_version.txt	Explains what are updated for each version in Chinese.
CHANGELOG.md	Explains what are updated for each version in English.

2 The concepts of tuya ble service

The tuya ble sdk does not provide the interfaces for initializing service. Your Application needs to implement the service characteristics defined in the following table before you initializing the SDK. Other than the services required by the tuya ble sdk, you can also define other services if needed. The initial format of broadcast data must be implemented according to the following table, otherwise the tuya ble sdk cannot work.

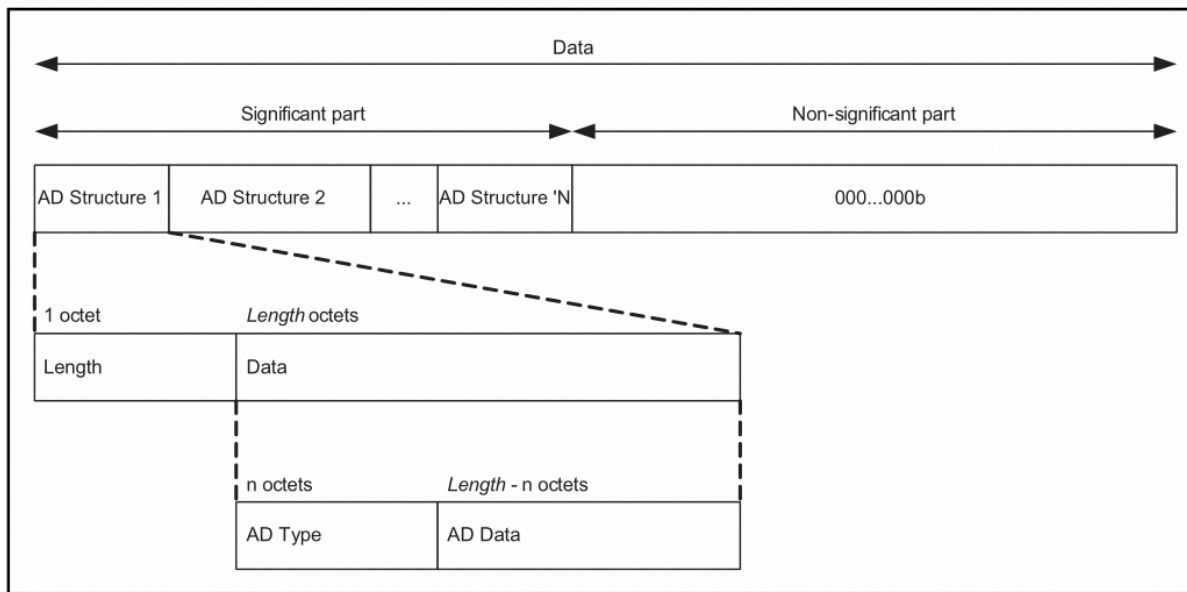
Service UUID	Characteristic UUID	Properties	Security Permissions
1910	2b10	Notify	None.
	2b11	Write,write without response.	None.

2.1 MTU

For a better compatibility, the ATT MTU used by tuya ble sdk is 23, and the GATT MTU (ATT DATA MAX) is 20.

2.2 Broadcast data format

The following picture illustrates the broadcast packet format of BLE.



The following table describes what are contained in the broadcast packet.

Broadcast data segment	Type	Description
Physical connection identifier of BLE device	0x01	Length: 0x02; Type: 0x01; Data: 0x16
Service UUID	0x02	Length: 0x03; Type: 0x02; Data: 0xA201
Service Data	0x16	Length: 0x0C or 0x14 Type: 0x16 Data: 0x01, 0xA2, type (0-pid, 1-product_key)PID, or product_key (in 8 or 16 byte)

Example of 8 byte PID: 02 01 05 03 02 01 A2 0C 16 01 A2 00 00 00 00 00 00 00 00 00










The following table describes what are contained in the scan response data.

Response data segment	Type	Description
Complete Local Name	0x09	Length: 0x03; Type: 0x09; Date: 0x54 or 0x59
Custom data defined by manufacturer	0xff	Length: 0x19 Type: 0xff Date: COMPANY ID:0x07D0 FLAG: 0x00 Protocol version: 0x03 Encryption method: 0x00 Communication capacity: 0x0000 Reserved field: 0x00 ID field: 6 or 16 bytes

Example of an unassociated devices: 03 09 54 59 19 FF D0 07 00 0300 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

3 How to port and configure tuyu ble sdk

As the following picture shows, the interfaces defined in the file `tuya_ble_port.h` and `tuya_ble_port_peripheral.h` must be ported and implemented according to the chip-specific platform. Note that if the platform used in the Application does not have an OS, the OS related interfaces do not need to be implemented. The `tuya_ble_port.c` and `tuya_ble_port_peripheral.c` are the weak implementation of the interfaces defined for the `tuya_ble_port.h` and `tuya_ble_port_peripheral.h`.

 bk	2019/9/10 10:30
 cypress	2019/9/10 10:30
 nordic	2019/9/14 16:23
 realtek	2019/9/14 16:23
 telink	2019/9/10 21:57
 <code>tuya_ble_port.c</code>	2019/10/3 16:16
 <code>tuya_ble_port.h</code>	2019/10/18 16:54
 <code>tuya_ble_port_peripheral.c</code>	2019/9/16 11:05
 <code>tuya_ble_port_peripheral.h</code>	2019/9/16 11:08

You cannot implement platform-specific interfaces in the preceding `.c` files, please create a new one, for example `tuya_ble_port_nrf52832.c`. If the file name contains the keyword `tuya`, it is the platform implementation file that Tuya Smart has adapted and transplanted, you can refer to it if needed.

3.1 Reference of porting APIs

3.1.1 TUYA_BLE_LOG

Function name	TUYA_BLE_LOG
Prototype	void TUYA_BLE_LOG(const char *format,...)

Function name	TUYA_BLE_LOG
Description	Formatted output.
Parameters	format[in]: format controller. ...[in]: variable parameters.
Responses	TUYA_BLE_ERR_INVALID_PARAM: invalid parameters

3.1.2 TUYA_BLE_HEXDUMP

Function name	TUYA_BLE_HEXDUMP
Prototype	void TUYA_BLE_HEXDUMP(uint8_t *p_data , uint16_t len)
Description	Print in hex values.
Parameters	p_data[in]: the data pointer to be printed. len[in]: data length.
Responses	TUYA_BLE_ERR_INVALID_PARAM: invalid parameters

3.1.3 tuyu_ble_gap_advertising_adv_data_update

Function name	tuyu_ble_gap_advertising_adv_data_update
Prototype	tuyu_ble_status_t tuyu_ble_gap_advertising_adv_data_update(uint8_t const * p_ad_data, uint8_t ad_len)
Description	Updates the BLE broadcast packets.
Parameters	p_ad_data[in]: new broadcast data. ad_len[in]: data length.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.4 tuyu_ble_gap_advertising_scan_rsp_data_update

Function name	tuyu_ble_gap_advertising_scan_rsp_data_update
Prototype	tuyu_ble_status_t tuyu_ble_gap_advertising_scan_rsp_data_update(uint8_t *p_sr_data, uint8_t sr_len)
Description	Updates the scan response data.
Parameters	p_sr_data[in]: new scan response data. sr_len[in]: data length.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.5 tuyu_ble_gap_disconnect

Function name	tuyu_ble_gap_disconnect
Prototype	tuyu_ble_status_t tuyu_ble_gap_disconnect(void)
Description	Break the BLE connection.
Parameters	None.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.6 tuyu_ble_gatt_send_data

Function name	tuyu_ble_gatt_send_data
Prototype	tuyu_ble_status_t tuyu_ble_gatt_send_data(const uint8_t *p_data, uint16_t len)
Description	Sends data via BLE GATT.

Function name	tuya_ble_gatt_send_data
Parameters	p_data[in]: the data pointer to send. len[in]: the data length, less than 20 bytes.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	It must be done in the notification method.

3.1.7 tuya_ble_timer_create

Function name	tuya_ble_timer_create
Prototype	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)
Description	create a timer.
Parameters	p_timer_id[out]: the timer pointer. timeout_value_ms[in]: the timing, unit: ms. mode[in]: - TUYA_BLE_TIMER_SINGLE_SHOT: one-time mode. - TUYA_BLE_TIMER_REPEATED: recurring mode. - timeout_handler [in]: the timer callback.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.8 tuyu_ble_timer_delete

Function name	tuya_ble_timer_delete
Prototype	tuya_ble_status_t tuya_ble_timer_delete(void* timer_id)
Description	Deletes a timer.
Parameters	timer_id [in]: the timer ID.
Responses	TUYA_BLE_SUCCESS: success. TUYA_BLE_ERR_INVALID_PARAM: invalid parameters. Others: failure.

3.1.9 tuyu_ble_timer_start

Function name	tuya_ble_timer_start
Prototype	tuya_ble_status_t tuya_ble_timer_start(void* timer_id)
Description	Starts a timer.
Parameters	timer_id [in]: the timer ID.
Responses	TUYA_BLE_SUCCESS: success. TUYA_BLE_ERR_INVALID_PARAM: invalid parameters. Others: failure.
Notice	If a timer has been started, a <code>tuya_ble_timer_start</code> request will start the timer again.

3.1.10 tuyu_ble_timer_restart

Function name	tuya_ble_timer_restart
Prototype	tuya_ble_status_t tuya_ble_timer_restart(void* timer_id,uint32_t timeout_value_ms)
Description	Restarts a timer with a new time.
Parameters	timer_id [in]: the timer ID. timeout_value_ms[in]: the timing, unit: ms.
Responses	TUYA_BLE_SUCCESS: success. TUYA_BLE_ERR_INVALID_PARAM: invalid parameters. Others: failure.

3.1.11 tuya_ble_timer_stop

Function name	tuya_ble_timer_stop
Prototype	tuya_ble_status_t tuya_ble_timer_stop(void* timer_id)
Description	Stops a timer.
Parameters	timer_id [in]: the timer ID.
Responses	TUYA_BLE_SUCCESS: success. TUYA_BLE_ERR_INVALID_PARAM: invalid parameters. Others: failure.

3.1.12 tuya_ble_device_delay_ms

Function name	tuya_ble_device_delay_ms
Prototype	void tuya_ble_device_delay_ms(uint32_t ms)
Description	Delays an activity at the millisecond level.
Parameters	ms [in]: the delay time, unit: ms.
Responses	None.
Notice	If the platform has an OS, it must be non-blocking delay. This method only applies to specific cases, for example, SDK initialization or device restart.

3.1.13 tuya_ble_device_delay_us

Function name	tuya_ble_device_delay_us
Prototype	void tuya_ble_device_delay_us(uint32_t us)
Description	Delays an activity at the microsecond level.
Parameters	us [in]: the delay time, unit: us.
Responses	None.
Notice	If the platform has an OS, it must be non-blocking delay. This method only applies to specific cases, for example, SDK initialization or device restart.

3.1.14 tuya_ble_device_reset

Function name	tuya_ble_device_reset
Prototype	tuya_ble_status_t tuya_ble_device_reset(void)
Description	Restarts devices.
Parameters	None.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.15 tuya_ble_gap_addr_get

Function name	tuya_ble_gap_addr_get
Prototype	tuya_ble_status_t tuya_ble_gap_addr_get(tuya_ble_gap_addr_t *p_addr);
Description	Obtains the MAC address of devices.
Parameters	p_addr [out]: the MAC address pointer.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

The following sample illustrates how to use the `tuya_ble_gap_addr_get` method.

```

1 typedef enum
2 {
3 {
4 TUYA_BLE_ADDRESS_TYPE_PUBLIC, // The public address
5 TUYA_BLE_ADDRESS_TYPE_RANDOM, // The random address
6 } tuyu_ble_addr_type_t;
7
8 typedef struct
9 {
10 tuyu_ble_addr_type_t addr_type;
11 uint8_t addr[6];
12 }tuyu_ble_gap_addr_t;

```

3.1.16 tuyu_ble_gap_addr_set

Function name	tuyu_ble_gap_addr_set
Prototype	tuyu_ble_status_t tuyu_ble_gap_addr_set(tuyu_ble_gap_addr_t *p_addr);
Description	Updates the MAC address of the device.
Parameters	p_addr [in]: the MAC address pointer.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

3.1.17 tuyu_ble_device_enter_critical

Function name	tuyu_ble_device_enter_critical
Prototype	voidtuyu_ble_device_enter_critical(void)
Description	Enters the critical zone.
Parameters	None.

Function name	tuya_ble_device_enter_critical
----------------------	--------------------------------

Responses	None.
------------------	-------

3.1.18 tuya_ble_device_exit_critical

Function name	tuya_ble_device_exit_critical
----------------------	-------------------------------

Prototype	void tuya_ble_device_exit_critical(void)
------------------	--

Description	Quits the critical zone.
--------------------	--------------------------

Parameters	None.
-------------------	-------

Responses	None.
------------------	-------

3.1.19 tuya_ble_rand_generator

Function name	tuya_ble_rand_generator
----------------------	-------------------------

Prototype	tuya_ble_status_t tuya_ble_rand_generator(uint8_t* p_buf, uint8_t len)
------------------	--

Description	Generates a random digit.
--------------------	---------------------------

Parameters	p_buf [out]: the array of random digit pointer. len[in]: the byte number of the random value.
-------------------	--

Responses	TUYA_BLE_SUCCESS: success. Others: failure.
------------------	--

3.1.20 tuya_ble_rtc_get_timestamp

Function name	tuya_ble_rtc_get_timestamp
Prototype	tuya_ble_status_t tuya_ble_rtc_get_timestamp(uint32_t timestamp,int32_t timezone);
Description	Obtains the Unix timestamp.
Parameters	timestamp [out]: the timestamp. timezone [out]: the time zone, data type: signed integer, the value must be 100 times of the real time.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	The data comes from the RTC (real-time clock) maintained by the Application itself. If the Application does not have an RTC, it is not necessary to use this interface.

3.1.21 tuya_ble_rtc_set_timestamp

Function name	tuya_ble_rtc_set_timestamp
Prototype	tuya_ble_status_t tuya_ble_rtc_set_timestamp(uint32_t timestamp,int32_t timezone)
Description	Updates the Unix timestamp.
Parameters	timestamp [in]: Unix timestamp. timezone [in]: the time zone, data type: signed integer, the value must be 100 times of the real time.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

Function name	tuya_ble_rtc_set_timestamp
Notice	The tuya ble sdk uses this API to update the RTC clock for the Application. If the Application does not have an RTC, it is not necessary to use this interface.

3.1.22 tuya_ble_nv_init

Function name	tuya_ble_nv_init
Prototype	tuya_ble_status_t tuya_ble_nv_init(void)
Description	Initializes the NV.
Parameters	None.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	It is used together with the NV space address that is defined in the configuration file. The tuya ble sdk calls NV related functions to store and manage authorization and other details.

3.1.23 tuya_ble_nv_erase

Function name	tuya_ble_nv_erase
Prototype	tuya_ble_status_t tuya_ble_nv_erase(uint32_t addr,uint32_t size)
Description	Erases the NV.

Function name	tuya_ble_nv_erase
Parameters	addr[in]: the initial address of NV space to be erased. size[in]: the space length to be erased, unit: byte.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	It is used together with the NV space address that is defined in the configuration file. The tuya ble sdk calls NV related functions to store and manage authorization and other details.

3.1.24 tuya_ble_nv_write

Function name	tuya_ble_nv_write
Prototype	tuya_ble_status_t tuya_ble_nv_write(uint32_t addr,const uint8_t * p_data, uint32_t size)
Description	Writes data to NV.
Parameters	addr[in]: the initial address of NV space to write data. p_data[in]: the initial address of NV space to write data. size[in]: the data size, unit: byte.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.

Function name	tuya_ble_nv_write
Notice	It is used together with the NV space address that is defined in the configuration file. The tuyu ble sdk calls NV related functions to store and manage authorization and other details.

3.1.25 tuyu_ble_nv_read

Function name	tuya_ble_nv_read
Prototype	tuya_ble_status_t tuya_ble_nv_read(uint32_t addr,uint8_t * p_data, uint32_t size)
Description	Reads data from NV.
Parameters	addr[in]: the initial address of NV space to read data. p_data[out]: the initial address of NV space to write data. size[in]: the data size, unit: byte.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	It is used together with the NV space address that is defined in the configuration file. The tuyu ble sdk calls NV related functions to store and manage authorization and other details.

3.1.26 tuyu_ble_nv_erase_async

Function name	tuya_ble_nv_erase_async
Prototype	void tuya_ble_nv_erase_async(uint32_t addr,uint32_t size,void *p_context,tuya_ble_nv_async_callback_t callback)
Description	Erases data from NV asynchronously.
Parameters	addr[in]: the initial address of NV space to erase data. size[in]: the data size, unit: byte. p_context[in]: user custom data. callback[in]: returns the API results.
Responses	None.
Notice	It is used together with the NV space address that is defined in the configuration file. The tuyu ble sdk calls NV related functions to store and manage authorization and other details. The interface is used in the asynchronous flash based platform.

3.1.27 tuyu_ble_nv_write_async

Function name	tuya_ble_nv_write_async
Prototype	void tuyu_ble_nv_write_async(uint32_t addr,const uint8_t * p_src, uint32_t size,void *p_context,tuya_ble_nv_async_callback_t callback)
Description	Writes data to NV asynchronously.

Function name	tuya_ble_nv_write_async
Parameters	<p>addr[in]: the initial address of NV space to write data.</p> <p>p_src[in]: the address to write date.</p> <p>size[in]: the data size, unit: byte.</p> <p>p_context[in]: user custom data.</p> <p>callback[in]: returns the API results.</p>
Responses	None.
Notice	<p>It is used together with the NV space address that is defined in the configuration file. The tuyu ble sdk calls NV related functions to store and manage authorization and other details. The interface is used in the asynchronous flash based platform.</p>

3.1.28 tuyu_ble_nv_read_async

Function name	tuya_ble_nv_read_async
Prototype	<pre>void tuyu_ble_nv_read_async(uint32_t addr, uint8_t* p_dest, uint32_t size,void *p_context,tuyu_ble_nv_async_callback_t callback)</pre>
Description	Reads data from NV asynchronously.
Parameters	<p>addr[in]: the initial address of NV space to read data.</p> <p>p_dest[out]: the address to read data.</p> <p>size[in]: the data size, unit: byte.</p> <p>p_context[in]: user custom data.</p> <p>callback[in]: returns the API results.</p>
Responses	None.

Function name	tuya_ble_nv_read_async
Notice	It is used together with the NV space address that is defined in the configuration file. The tuyu ble sdk calls NV related functions to store and manage authorization and other details. The interface is used in the asynchronous flash based platform.

3.1.29 tuyu_ble_common_uart_init

Function name	tuya_ble_common_uart_init
Prototype	tuya_ble_status_t tuya_ble_common_uart_init(void)
Description	Initializes the UART.
Parameters	None.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	Under the following circumstances, there is no need to implement the <code>tuya_ble_common_uart_init</code> interface: - The functions of test and production authorization are not used. - The SDK' s production testing authorization function are used, and the Application has initialized the UART before initializing the SDK.

3.1.30 tuyu_ble_common_uart_send_data

Function name	tuya_ble_common_uart_send_data
Prototype	tuya_ble_status_t tuya_ble_common_uart_send_data(const uint8_t *p_data,uint16_t len)
Description	Sends data via UART.
Parameters	p_data[in]: the data pointer to send. len[in]: the data size.
Responses	TUYA_BLE_SUCCESS: success. Others: failure.
Notice	If the functions of test and production authorization are not used, there is no need to implement the tuya_ble_common_uart_send_data interface.

3.1.31 tuya_ble_os_task_create

Function name	tuya_ble_os_task_create
Prototype	bool tuya_ble_os_task_create(void **pp_handle, const char <i>p_name</i> , void(<i>p_routine</i>)(void),void <i>p_param</i> , uint16_t stack_size, uint16_t priority)
Description	Creates tasks.

Function name	tuya_ble_os_task_create
Parameters	<p>pp_handle [out]: passes back a handle by which the created task can be referenced.</p> <p>p_name[in]: A descriptive name for the task.</p> <p>p_routine [in]: pointer to task routine function that must be implemented to never return.</p> <p>p_param[in]: pointer parameter passed to the task routine function.</p> <p>stack_size[in]: the size of the task stack that is specified as the number of bytes.</p> <p>priority[in]: the priority at which the task should run. Higher priority task has higher priority value.</p>
Responses	<p>True: task was created successfully and added to task ready list.</p> <p>False: task was failed to create.</p>
Notice	This interface is required only in the platform that has an OS.

3.1.32 tuya_ble_os_task_delete

Function name	tuya_ble_os_task_delete
Prototype	bool tuya_ble_os_task_delete(void *p_handle)
Description	Removes a task from RTOS' s task management. The task being deleted will be removed from RUNNING, READY, or WAITING state.
Parameters	pp_handle [in]: the handle of the task to be deleted.

Function name	tuya_ble_os_task_delete
Responses	True: task was deleted successfully False: task was failed to delete.
Notice	This interface is required only in the platform that has an OS.

3.1.33 tuya_ble_os_task_suspend

Function name	tuya_ble_os_task_suspend
Prototype	bool tuya_ble_os_task_suspend(void *p_handle)
Description	Suspends the task. The suspended task will not be scheduled and never get any microcontroller processing time.
Parameters	pp_handle [in]: the handle of the task to be suspend.
Responses	True: task was suspend successfully. False: task was failed to suspend.
Notice	This interface is required only in the platform that has an OS.

3.1.34 tuya_ble_os_task_resume

Function name	tuya_ble_os_task_resume
Prototype	bool tuya_ble_os_task_resume(void *p_handle)
Description	Resumes the suspended task.
Parameters	pp_handle [in]: the handle of the task to be resumed.

Function name	tuya_ble_os_task_resume
Responses	True: task was resumed successfully. False: task was failed to resume.
Notice	This interface is required only in the platform that has an OS.

3.1.35 tuya_ble_os_msg_queue_create

Function name	tuya_ble_os_msg_queue_create
Prototype	bool tuya_ble_os_msg_queue_create(void **pp_handle, uint32_t msg_num, uint32_t msg_size)
Description	Creates a message queue instance. This allocates the storage required by the new queue and passes back a handle for the queue.
Parameters	pp_handle [out]: passes 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.
Responses	True: message queue was created successfully. False: message queue was failed to create.
Notice	This interface is required only in the platform that has an OS.

3.1.36 tuya_ble_os_msg_queue_delete

Function name	tuya_ble_os_msg_queue_delete
Prototype	bool tuya_ble_os_msg_queue_delete(void *p_handle)
Description	Creates a message queue instance. This allocates the storage required by the new queue and passes back a handle for the queue.
Parameters	pp_handle [in]: the handle to the message queue being deleted.
Responses	True: message queue was deleted successfully. False: message queue was failed to delete.
Notice	This interface is required only in the platform that has an OS.

3.1.37 tuya_ble_os_msg_queue_peek

Function name	tuya_ble_os_msg_queue_peek
Prototype	bool tuya_ble_os_msg_queue_peek(void p_handle, uint32_t p_msg_num)
Description	Peeks the number of items sent and resided on the message queue.
Parameters	pp_handle [in]: the handle to the message queue being peeked. p_msg_num[out]: passes back the number of items residing on the message queue.

Function name	tuya_ble_os_msg_queue_peek
Responses	True: message queue was peeked successfully. False: message queue was failed to peek.
Notice	This interface is required only in the platform that has an OS.

3.1.38 tuya_ble_os_msg_queue_send

Function name	tuya_ble_os_msg_queue_send
Prototype	bool tuya_ble_os_msg_queue_send(void p_handle, void p_msg, uint32_t wait_ms)
Description	Sends an item to the back of the specified message queue.
Parameters	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: blocks infinitely until the item received. - others: the timeout value in milliseconds.

Function name	tuya_ble_os_msg_queue_send
Responses	True: message item was sent successfully. False: message item was failed to send.
Notice	This interface is required only in the platform that has an OS.

3.1.39 tuya_ble_os_msg_queue_recv

Function name	tuya_ble_os_msg_queue_recv
Prototype	bool tuya_ble_os_msg_queue_recv(void p_handle, void p_msg, uint32_t wait_ms)
Description	Receives an item from the specified message queue.
Parameters	<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 be received on the queue.</p> <ul style="list-style-type: none"> - 0: no blocking and return immediately. - 0xFFFFFFFF: blocks infinitely until the item is received. - others: the timeout value in milliseconds.
Responses	True: message item was received successfully. False: message item was failed to receive.

Function name	tuya_ble_os_msg_queue_rcv
Notice	This interface is required only in the platform that has an OS.

3.1.40 tuya_ble_event_queue_send_port

Function name	tuya_ble_event_queue_send_port
Prototype	bool tuya_ble_event_queue_send_port(tuya_ble_evt_param_t *evt, uint32_t wait_ms)
Description	If the TUYA_BLE_SELF_BUILT_TASK is undefined, Application should provide the task to sdk to process the event. The sdk will use this port to send event to the task of provided by Application.
Parameters	evt [in]: the message data point to be send. wait_ms[in]: the maximum amount of time in milliseconds that the task should block waiting for the item to sent on the queue.
Responses	True: message item was sent successfully. False: message item was failed to send.
Notice	This interface is required only in the platform that has an OS.

3.1.41 tuya_ble_aes128_ecb_encrypt

Function name	tuya_ble_aes128_ecb_encrypt
Prototype	bool tuya_ble_aes128_ecb_encrypt(uint8_t key,uint8_t input,uint16_tinput_len,uint8_t *output)
Description	128 bit AES ECB encryption on specified plaintext and keys.
Parameters	key [in]: keys to encrypt the plaintext In_put[in]: specified plaintext 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.
Responses	True: successful. False: fail.

3.1.42 tuya_ble_aes128_ecb_decrypt

Function name	tuya_ble_aes128_ecb_decrypt
Prototype	bool tuya_ble_aes128_ecb_decrypt(uint8_t key,uint8_t input,uint16_tinput_len,uint8_t *output)
Description	128 bit AES ECB decryption on specified encrypted data and keys

Function name	tuya_ble_aes128_ecb_decrypt
Parameters	<p>key [in]: keys to decrypt the plaintext</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>
Responses	<p>True: successful.</p> <p>False: fail.</p>

3.1.43 tuya_ble_aes128_cbc_encrypt

Function name	tuya_ble_aes128_cbc_encrypt
Prototype	<p>bool</p> <p>tuya_ble_aes128_cbc_encrypt(uint8_t key,uint8_t iv,uint8_t input,uint16_t input_len,uint8_t output)</p>
Description	128 bit AES CBC encryption on specified plaintext and keys.
Parameters	<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>
Responses	<p>True: successful.</p> <p>False: fail.</p>

3.1.44 tuyu_ble_aes128_cbc_decrypt

Function name	tuyu_ble_aes128_cbc_decrypt
Prototype	bool tuyu_ble_aes128_cbc_decrypt(uint8_t key, uint8_t iv, uint8_t input, uint16_t input_len, uint8_t output)
Description	128 bit AES CBC decryption on specified plaintext and keys.
Parameters	key [in]: keys to decrypt the plaintext. iv[in]: initialization vector (IV) for CBC mode In_put[in]: specified encrypted data to be decrypted. in_put_len[in]: byte length of the data to be decrypted, must be multiples of 16. Out_put[out]: output buffer to store decrypted data.
Responses	True: successful. False: fail.

3.1.45 tuyu_ble_md5_crypt

Function name	tuyu_ble_md5_crypt
Prototype	bool tuyu_ble_md5_crypt(uint8_t input, uint16_t input_len, uint8_t output)
Description	MD5 checksum.

Function name	tuya_ble_md5_crypt
Parameters	<p>In_put[in]: specified plain text to be encrypted.</p> <p>in_put_len[in]: byte length of the data to be encrypted.</p> <p>Out_put[out]: output buffer to store md5 result data,output data length is always 16.</p>
Responses	<p>True: successful.</p> <p>False: fail.</p>

3.1.46 tuya_ble_hmac_sha1_crypt

Function name	tuya_ble_hmac_sha1_crypt
Prototype	<pre>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)</pre>
Description	Calculates the full generic HMAC on the input buffer with the provided key.
Parameters	<p>key [in]: The HMAC secret key.</p> <p>key_len[in]: The length of the HMAC secret key in bytes.</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.</p> <p>Out_put[out]: output buffer to store the result data.</p>
Responses	<p>True: successful.</p> <p>False: fail.</p>

Function name	tuya_ble_hmac_sha1_crypt
Notice	Not used currently, no need to implement.

3.1.47 tuya_ble_hmac_sha256_crypt

Function name	tuya_ble_hmac_sha1_crypt
Prototype	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)
Description	Calculates the full generic HMAC on the input buffer with the provided key.
Parameters	key [in]: The HMAC secret key. key_len[in]: The length of the HMAC secret key in Bytes. In_put[in]: specified 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.
Responses	True: successful. False: fail.
Notice	Not used currently, no need to implement.

3.1.48 tuya_ble_port_malloc

Function name	tuya_ble_port_malloc
Prototype	void *tuya_ble_port_malloc(uint32_t size)
Description	Allocate a memory block with required size.
Parameters	Size[in]: Required memory size.
Responses	The address of the allocated memory block. If the address is NULL, the memory allocation failed.
Notice	You need to implement this interface only when TUYA_BLE_USE_PLATFORM_MEMORY_HEAP is set to 1.

3.1.49 tuya_ble_port_free

Function name	tuya_ble_port_free
Prototype	void tuya_ble_port_free(void *pv)
Description	Frees a memory block that had been allocated.
Parameters	pv[in]: The address of memory block being freed.
Responses	None.
Notice	You need to implement this interface only when TUYA_BLE_USE_PLATFORM_MEMORY_HEAP is set to 1.

3.2 Reference of configuration APIs

The items in the `tuya_ble_config.h` file are used to configure the tuyu ble sdk for different use cases, for example, network configuration for multi-protocol devices, whether running an OS on the platform or not, device communication, whether self-manage the authorization, and so forth.

3.2.1 CUSTOMIZED_TUYA_BLE_CONFIG_FILE

Macro	CUSTOMIZED_TUYA_BLE_CONFIG_FILE
Dependency	None.
Description	Custom configuration file. This file overwrites the default settings in the <code>tuya_ble_config.h</code> file. Your Application must create a configuration file with a new name, and assign the file name to the <code>CUSTOMIZED_TUYA_BLE_CONFIG_FILE</code> , for example, <code>CUSTOMIZED_TUYA_BLE_CONFIG_FILE = <custom_tuya_ble_config.h></code> .

3.2.2 CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE

Macro	CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE
Dependency	None.
Description	The tuyu ble sdk provides basic test and production authorization services. If the Tuya testing protocol is used in your Application to perform many customized test activities, you must create a file to achieve that, and assign the header file name to <code>CUSTOMIZED_TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE</code> .

Macro	CUSTOMIZED TUYA BLE APP PRODUCT TEST HEAD
Notice	It must be completed in the custom configuration file.

3.2.3 CUSTOMIZED TUYA BLE APP UART COMMON HEADER FILE

Macro	CUSTOMIZED TUYA BLE APP UART COMMON HEAD
Dependency	None.
Description	At present, it only applies to the general module protocol of Tuya Smart.
Notice	It must be completed in the custom configuration file.

3.2.4 TUYA BLE USE OS

Macro	TUYA_BLE_USE_OS
Dependency	None.
Description	If the chip architecture is based on OS, such as FreeRTOS, set #define TUYA_BLE_USE_OS 1, otherwise, set #define TUYA_BLE_USE_OS 0.

3.2.5 TUYA BLE SELF_BUILT_TASK

Macro	TUYA_BLE_SELF_BUILT_TASK
Dependency	#define TUYA_BLE_USE_OS 1

Macro	TUYA_BLE_SELF_BUILT_TASK
Description	<p>For the OS architecture platform, if you use self-built tasks in the tuyu ble sdk to handle messages, please set</p> <pre>#define TUYA_BLE_SELF_BUILT_TASK 1,</pre> <p>otherwise, set</p> <pre>#define TUYA_BLE_SELF_BUILT_TASK 0.</pre>

3.2.6 TUYA_BLE_TASK_PRIORITY

Macro	TUYA_BLE_TASK_PRIORITY
Dependency	<pre>#define TUYA_BLE_USE_OS 1</pre> <pre>#define TUYA_BLE_SELF_BUILT_TASK 1</pre>
Description	<p>The priority of a task initiated by tuyu ble sdk. For example,</p> <pre>#define TUYA_BLE_TASK_PRIORITY 1.</pre>
Notice	<p>The priority value needs to be defined according to the OS used by the chip platform.</p>

3.2.7 TUYA_BLE_TASK_STACK_SIZE

Macro	TUYA_BLE_TASK_STACK_SIZE
Dependency	<pre>#define TUYA_BLE_USE_OS 1</pre> <pre>#define TUYA_BLE_SELF_BUILT_TASK 1</pre>
Description	<p>The stack size of a task initiated by tuyu ble sdk. For example,</p> <pre>#define TUYA_BLE_TASK_STACK_SIZE 256*10.</pre>

3.2.8 TUYA_BLE_DEVICE_COMMUNICATION_ABILITY

Macro	TUYA_BLE_DEVICE_COMMUNICATION_ABILITY
Dependency	None.
Description	<p>The communication of devices.</p> <p>Possible setting:</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_BLE</code> : whether use BLE or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_REGISTER_FROM_BLE</code> : whether register devices via ble or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_MESH</code> : whether use Mesh or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_24G</code> : whether support 2.4G Wi-Fi or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_WIFI_5G</code> : whether support 5G Wi-Fi or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_ZIGBEE</code> : whether support Zigbee or not.</p> <p><code>TUYA_BLE_DEVICE_COMMUNICATION_ABILITY_NB</code> : whether support NB-IoT or not.</p>

3.2.9 TUYA_BLE_DEVICE_SHARED

Macro	TUYA_BLE_DEVICE_SHARED
Dependency	None.
Description	Determines whether a device is shared.
Notice	<p>If you do not understand what are shared devices, please set</p> <p><code>#define TUYA_BLE_DEVICE_SHARED 0.</code></p>

3.2.10 TUYA_BLE_DEVICE_UNBIND_MODE

Macro	TUYA_BLE_DEVICE_UNBIND_MODE
Dependency	None.
Description	Determines whether the shared devices need to perform the unbind operation.
Notice	If you do not understand what are shared devices, please set <code>#define TUYA_BLE_DEVICE_UNBIND_MODE 1.</code>

3.2.11 TUYA_BLE_WIFI_DEVICE_REGISTER_MODE

Macro	TUYA_BLE_WIFI_DEVICE_REGISTER_MODE
Dependency	The devices must support Wi-Fi connection.
Description	Determines whether to send command to query the network configuration state or not when use BLE to configure network for Wi-Fi devices. If yes: <code>#define</code> <code>TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 1</code> If no: <code>#define</code> <code>TUYA_BLE_WIFI_DEVICE_REGISTER_MODE 0</code>
Notice	Currently not supported.

3.2.12 TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT

Macro	TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT
Dependency	None.

Macro	TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT
Description	<p>If you use self-built tasks in the tuya ble sdk to manage authorization, please set <code>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 1</code>, otherwise, set <code>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 0</code>.</p>
Notice	<p>For BLE devices without Wi-Fi capability, this value is recommended to be defined as 1.</p>

3.2.13 TUYA_BLE_SECURE_CONNECTION_TYPE

Macro	TUYA_BLE_SECURE_CONNECTION_TYPE
Dependency	None.
Description	<p>The communication encryption method of devices. Possible setting:</p> <p><code>TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY</code> : encrypt with <code>auth_key</code>.</p> <p><code>TUYA_BLE_SECURE_CONNECTION_WITH_ECC</code> : encrypt with ECDH.</p> <p><code>TUYA_BLE_SECURE_CONNECTION_WTIH_PASSTHROUGH</code> : no encrypt.</p> <p>For example:</p> <pre>#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY</pre>
Notice	<p>Currently, only <code>TUYA_BLE_SECURE_CONNECTION_WITH_AUTH_KEY</code> is supported.</p>

3.2.14 TUYA_BLE_DEVICE_MAC_UPDATE

Macro	TUYA_BLE_DEVICE_MAC_UPDATE
Dependency	None.
Description	Determines whether to use the MAC address in the Tuya authorization information as the device MAC address or not.

3.2.15 TUYA_BLE_DEVICE_MAC_UPDATE_RESET

Macro	TUYA_BLE_DEVICE_MAC_UPDATE_RESET
Dependency	<code>#define TUYA_BLE_DEVICE_MAC_UPDATE 1</code>
Description	Determines whether the device needs to restart after updating the MAC address to take effect or not. If yes: <code>#define TUYA_BLE_DEVICE_MAC_UPDATE_RESET 1</code> If no: <code>#define TUYA_BLE_DEVICE_MAC_UPDATE_RESET 0</code>

3.2.16 TUYA_BLE_USE_PLATFORM_MEMORY_HEAP

Macro	TUYA_BLE_USE_PLATFORM_MEMORY_HEAP
Dependency	None.
Description	Whether tuyu ble sdk uses its own memory heap. If yes: <code>#define TUYA_BLE_USE_PLATFORM_MEMORY_HEAP 0</code> If no: <code>#define TUYA_BLE_USE_PLATFORM_MEMORY_HEAP 1</code>

Macro	TUYA_BLE_USE_PLATFORM_MEMORY_HEAP
Notice	If it is defined as 1, your Application must port and implement memory management ability in the porting layer for tuya ble sdk.

3.2.17 TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE

Macro	TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE
Dependency	None.
Description	The size of the GATT send queue used by tuya ble sdk. Default value: 20. <code>#define TUYA_BLE_GATT_SEND_DATA_QUEUE_SIZE 20</code>
Notice	If you do not know much about the macro, it is recommended to keep it at 20.

3.2.18 TUYA_BLE_DATA_MTU_MAX

Macro	TUYA_BLE_DATA_MTU_MAX
Dependency	None.
Description	The size of GATT MTU: <code>#define TUYA_BLE_DATA_MTU_MAX 20</code>
Notice	Currently, only 20 bytes GATT MTU are supported, which will be expanded and upgraded later.

3.2.19 TUYA_BLE_LOG_ENABLE

Macro	TUYA_BLE_LOG_ENABLE
Dependency	None.
Description	Determines whether to enable log of tuyu ble sdk or not. If yes: <code>#define TUYA_BLE_LOG_ENABLE 1</code> If no: <code>#define TUYA_BLE_LOG_ENABLE 0</code>
Notice	Enabling the internal log consumes the code space. It is recommended to turn on it for the debug version and turn off for the release version.

3.2.20 TUYA_BLE_LOG_COLORS_ENABLE

Macro	TUYA_BLE_LOG_COLORS_ENABLE
Dependency	<code>#define TUYA_BLE_LOG_ENABLE 1</code>
Description	Determines whether to enable the multi-color display for logs of SDK or not. If yes: <code>#define TUYA_BLE_LOG_COLORS_ENABLE 1</code> If no: <code>#define TUYA_BLE_LOG_COLORS_ENABLE 0</code>
Notice	Some tools do not support it, for example, RTT of J-link.

3.2.21 TUYA_BLE_LOG_LEVEL

Macro	TUYA_BLE_LOG_LEVEL
Dependency	<code>#define TUYA_BLE_LOG_ENABLE 1</code>

Macro	TUYA_BLE_LOG_LEVEL
Description	<p>Defines the display level of the log of tuya ble sdk, which is divided into the following levels:</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>If only error information needs to be printed, set as:</p> <pre>#define TUYA_BLE_LOG_LEVEL TUYA_BLE_LOG_LEVEL_ERROR</pre>

3.2.22 TUYA_APP_LOG_ENABLE

Macro	TUYA_APP_LOG_ENABLE
Dependency	None.
Description	<p>Whether to enable Application log.</p> <p>If yes: <code>#define TUYA_APP_LOG_ENABLE 1</code></p> <p>If no: <code>#define TUYA_APP_LOG_ENABLE 0</code></p>
Notice	<p>Enabling the internal log consumes the code space. It is recommended to turn on it for the debug version and turn off for the release version.</p>

3.2.23 TUYA_APP_LOG_COLORS_ENABLE

Macro	TUYA_APP_LOG_COLORS_ENABLE
Dependency	<code>#define TUYA_APP_LOG_ENABLE 1</code>

Macro	TUYA_APP_LOG_COLORS_ENABLE
Description	<p>Determines whether to enable the multi-color display for logs of Application or not.</p> <p>If yes:</p> <pre>#define TUYA_APP_LOG_COLORS_ENABLE 1</pre> <p>If no:</p> <pre>#define TUYA_APP_LOG_COLORS_ENABLE 0</pre>
Notice	Some tools do not support it, for example, RTT of J-Link.

3.2.24 TUYA_APP_LOG_LEVEL

Macro	TUYA_APP_LOG_LEVEL
Dependency	<pre>#define TUYA_APP_LOG_ENABLE 1</pre>
Description	<p>Defines the display level of the log of tuy a ble sdk, which is divided into the following levels:</p> <pre>#define TUYA_APP_LOG_LEVEL_ERROR 1U</pre> <pre>#define TUYA_APP_LOG_LEVEL_WARNING 2U</pre> <pre>#define TUYA_APP_LOG_LEVEL_INFO 3U</pre> <pre>#define TUYA_APP_LOG_LEVEL_DEBUG 4U</pre> <p>If only error information needs to be printed, set as:</p> <pre>#define TUYA_APP_LOG_LEVEL</pre> <pre>TUYA_APP_LOG_LEVEL_ERROR</pre>

3.2.25 TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE

Macro	TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE
Dependency	None.

Macro	TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE
Description	<p>Determines whether to use advanced encryption method.</p> <p>If yes: <code>#define TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE 1</code></p> <p>If no: <code>#define TUYA_BLE_ADVANCED_ENCRYPTION_DEVICE 0</code></p>
Notice	It is not supported currently.

3.2.26 TUYA_NV_ERASE_MIN_SIZE

Macro	TUYA_NV_ERASE_MIN_SIZE
Dependency	None.
Description	<p>The minimum erasure unit of NV space allocated to tuyu ble sdk, for example:</p> <p><code>#define TUYA_NV_ERASE_MIN_SIZE 4096</code></p>
Notice	It must be defined according to the implementation of NV interface in the port layer.

3.2.27 TUYA_NV_WRITE_GRAN

Macro	TUYA_NV_WRITE_GRAN
Dependency	None.
Description	<p>The maximum write granularity in the NV space. For example,</p> <p><code>#define TUYA_NV_WRITE_GRAN 4</code> allows 4 bytes if you write to NV.</p>
Notice	It must be defined according to the implementation of NV interface in the port layer.

3.2.28 TUYA_NV_START_ADDR

Macro	TUYA_NV_START_ADDR
Dependency	None.
Description	The initial NV space address allocated to tuya ble sdk. For example: <code>#define TUYA_NV_START_ADDR 0x1000.</code>

3.2.29 TUYA_NV_AREA_SIZE

Macro	TUYA_NV_AREA_SIZE
Dependency	None.
Description	The NV space size allocated to tuya ble sdk. For example, <code>#define TUYA_NV_AREA_SIZE (4*</code> <code>TUYA_NV_ERASE_MIN_SIZE)</code> . The value must be an integer multiple of <code>TUYA_NV_ERASE_MIN_SIZE</code> .

3.2.30 TUYA_BLE_APP_VERSION_STRING

Macro	TUYA_BLE_APP_VERSION_STRING
Dependency	None.
Description	The Application version number string, for example, <code>#define TUYA_BLE_APP_VERSION_STRING "1.0".</code>
Notice	Currently, only two digit version numbers are supported.

3.2.31 TUYA_BLE_APP_BUILD_FIRMNAME_STRING

Macro	TUYA_BLE_APP_BUILD_FIRMNAME_STRING
Dependency	None.
Description	<p>The Application firmware name, for example, #define</p> <pre>TUYA_BLE_APP_BUILD_FIRMNAME_STRING " tuya_ble_sdk_app_demo_xxx".</pre>
Notice	It is only required when you use the Tuya production testing license agreement.

4 API reference

The tuya ble sdk provides packaged API for Application to achieve BLE related management, communication, and so forth. The APIs are defined in the [tuya_ble_api.c](#) and [tuya_ble_api.h](#) files, you can read the source code to understand how the APIs are implemented. The following tables describes the details of each API.

4.0.1 tuya_ble_main_tasks_exec

Function name	tuya_ble_main_tasks_exec
Prototype	void tuya_ble_main_tasks_exec(void)
Description	The tuya ble sdk event main scheduler when using the non-OS architecture chip platform. The Application can call it only in the main loop.
Parameters	None.
Responses	None.
Notice	This function must be called from within the main loop. It will execute all events scheduled since the last time it was called.

Example: the calling location under nrf52832 platform is as follows:

```

1  /**@brief Function for handling the idle state (main loop).
2   *
3   * @details If there is no pending log operation, then sleep until next
4   *          the next event occurs.
5   */
6  static void idle_state_handle(void)
7  {
8      tuya_ble_main_tasks_exec();
9      if (NRF_LOG_PROCESS() == false)
10     {
11         nrf_pwr_mgmt_run();
12     }
13 }
14 }
```

4.0.2 tuya_ble_gatt_receive_data

Function name	tuya_ble_gatt_receive_data
Prototype	tuya_ble_status_t tuya_ble_gatt_receive_data(uint8_t*p_data,uint16_t len);
Description	By calling this function, the GATT data received by Bluetooth is sent to tuya ble sdk.
Parameters	p_data[in]: point to the data to be sent. len[in]: the length of the data to be sent, cannot exceed TUYA_BLE_DATA_MTU_MAX .
Responses	TUYA_BLE_SUCCESS: send successfully. TUYA_BLE_ERR_INTERNAL: failed.
Notice	This function must be called from where the BLE data is received.

Example (nrf52832 example demo):


```

1  /**@brief Function for handling the data from the Nordic UART Service.
2  *
3  * @details This function will process the data received from the
4  *           Nordic UART BLE Service and send
5  *           it to the UART module.
6  *
7  * @param[in] p_evt      Nordic UART Service event.
8  */
9  /**@snippet [Handling the data received over BLE] */
10 static void nus_commdata_handler(ble_nus_evt_t * p_evt)
11 {
12     if (p_evt->type == BLE_NUS_EVT_RX_DATA)
13     {
14         tuyu_ble_gatt_receive_data((uint8_t*)(p_evt->params.rx_data.p_data),p_evt->params.rx_data.length);
15         TUYA_BLE_HEXDUMP("nus_commdata_handler :",20,(uint8_t*)(p_evt->params.rx_data.p_data),p_evt->params.rx_data.length);
16     }
17 }
18 }

```

4.0.3 tuyu_ble_common_uart_receive_data

Function name	tuyu_ble_common_uart_receive_data
Prototype	tuyu_ble_status_t tuyu_ble_common_uart_receive_data (uint8_t *p_data,uint16_t len)
Description	<p>If an Application uses the production testing authorization module provided by tuyu ble sdk, you need to send UART data to tuyu ble sdk by calling this function.</p> <p>If you do not use the production testing authorization function module provided by tuyu ble sdk, you do not need to call this function.</p>
Parameters	p_data[in]: point to the data to be sent. len[in]: length of data to be sent.
Responses	TUYA_BLE_SUCCESS: sent successfully. Other: failed to send.

Function name	tuya_ble_common_uart_receive_data
Notice	<p>This function internally calls malloc to apply for memory.</p> <p>If the Application configuration uses the malloc interface provided by the platform,</p> <p>confirm whether the platform malloc supports interrupt calls.</p> <p>If it does not support, it must not be called in UART interrupt</p> <p><code>tuya_ble_common_uart_receive_data()</code>.</p>

4.0.4 tuya_ble_common_uart_send_full_instruction_received

Function name	tuya_ble_common_uart_send_full_instruction_received
Prototype	<pre>tuya_ble_status_t. tuya_ble_common_uart_send_full_instruction_received(uint8_t *p_data,uint16_t len)</pre>
Description	<p>Sends the Tuya production testing protocol command (including the <code>cmd/data/checksum</code>) that are received and resolved by the UART to tuya ble sdk.</p> <p>If your Application does not use the protocol parsing function <code>tuya_ble_common_uart_receive_data()</code> provided by tuya ble sdk but uses a custom UART parsing function to parse the production testing protocol data, you need to call this function to send the parsed completed production testing instruction to tuya ble sdk.</p>

Function name	tuya_ble_common_uart_send_full_instruction_received
Parameters	<p>p_data[in]: Point to the complete command data to be sent.</p> <p>len[in]: Command data length to be sent.</p>
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_NO_MEM: Failed to apply for memory.</p> <p>TUYA_BLE_ERR_BUSY: BLE SDK busy.</p>
Notice	<p>This function internally calls the malloc interface. If the Application is configured to use the chip platform-provided malloc interface, make sure that platform malloc supports interrupt calls.</p> <p>If it does not, you must not call <code>tuya_ble_common_uart_send_full_instruction_received</code> () in the UART interrupt.</p>

Explanation:

- The tuya ble sdk integrates the production testing authorization function module. The production testing authorization module communicates through the UART and PC production testing authorization tools. UART communication has a complete set of command formats. For details, refer to the Bluetooth General Production Test Authorization Agreement.
- The tuya ble sdk includes the UART communication instruction parsing function. The Application only needs to call the `tuya_ble_common_uart_receive_data` () function where UART data is received. The Application can also parse out the complete UART communication instruction, and then call `tuya_ble_common_uart_send_full_instruction_received()` function to send a complete command to tuya ble sdk.

4.0.5 tuya_ble_device_update_product_id

Function name	tuya_ble_device_update_product_id
Prototype	tuya_ble_status_t tuya_ble_device_update_product_id. (tuya_ble_product_id_type_t type, uint8_t len, uint8_t* p_buf)
Description	Updates product id function.
Parameters	type [in]: id type. len[in]: id length. p_buf[in]: id data.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_PARAM: parameter error. TUYA_BLE_ERR_INTERNAL: internal error.
Notice	In the SoC development scheme based on tuya ble sdk, it is generally not necessary to call this function, because the product ID generally does not change.

4.0.6 tuya_ble_device_update_login_key

Function name	tuya_ble_device_update_login_key
Prototype	tuya_ble_status_t tuya_ble_device_update_login_key(uint8_t* p_buf, uint8_t len)
Description	Updates login key function.
Parameters	len[in]: length. p_buf[in]: loginkey.

Function name	tuya_ble_device_update_login_key
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_PARAM: parameter error. TUYA_BLE_ERR_INTERNAL: internal error.
Notice	This API is mainly used in Wi-Fi/BLE dual-protocol devices. Send network configuration information to Wi-Fi through BLE. The device registers the device with the cloud through Wi-Fi and calls. the login key after successful registration This function is sent to tuya ble sdk, but also needs to update the binding status.

4.0.7 tuya_ble_device_update_bound_state

Function name	tuya_ble_device_update_bound_state
Prototype	tuya_ble_status_t tuya_ble_device_update_bound_state(uint8_t state)
Description	Updates registration binding status.
Parameters	state[in]: 1-The device is registered for binding, 0-The device is not registered for binding.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_PARAM: parameter error. TUYA_BLE_ERR_INTERNAL: internal error.

Function name	tuya_ble_device_update_bound_state
Notice	<p>This API is mainly used in Wi-Fi/BLE dual-protocol devices.</p> <p>(After the device registers and binds successfully through the Wi-Fi link, this function is called to update the binding status to tuya ble sdk.</p> <p>It also requires updating login key).</p>

4.0.8 tuya_ble_device_update_mcu_version

Function name	tuya_ble_device_update_mcu_version
Prototype	<pre>tuya_ble_status_t tuya_ble_device_update_mcu_version(uint32_t mcu_firmware_version, uint32_t mcu_hardware_version)</pre>
Description	Updates external mcu version number.
Parameters	<p>mcu_firmware_version [in]: MCU Firmware version number, For example, 0x010101 indicates v1.1.1.</p> <p>mcu_hardware_version [in]: MCU hardware version number (PCBA version number).</p>
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>Other: failed.</p>
Notice	The API is mainly used to develop BLE module, SoC development program does not need to use.

4.0.9 tuya_ble_sdk_init

Function name	tuya_ble_sdk_init
Prototype	tuya_ble_status_t tuya_ble_sdk_init(tuya_ble_device_param_t * param_data)
Description	The tuya ble sdk initialization function.
Parameters	param_data [in]: Initialize parameter.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_PARAM: parameter error.
Notice	Initialization function, the Application must call this function to initialize the SDK, otherwise the SDK will not work. The initialization function may also be <code>tuya_ble_sdk_init_async()</code> .

Parameter explanation:

The structure of `tuya_ble_device_param_t` is as follows:

```
1 typedef struct{
2     uint8_t device_id_len;    //if ==20, Compressed into 16
3     uint8_t device_id[DEVICE_ID_LEN_MAX];
4     tuya_ble_product_id_type_t p_type;
5     uint8_t product_id_len;
6     uint8_t product_id[TUYA_BLE_PRODUCT_ID_MAX_LEN];
7     uint8_t device_vid[DEVICE_VIRTUAL_ID_LEN];
8     uint8_t auth_key[AUTH_KEY_LEN];
9     uint8_t login_key[LOGIN_KEY_LEN];
10    uint8_t bound_flag;
11    uint32_t firmware_version; //0x00010102: v1.1.2
12    uint32_t hardware_version;
13    uint8_t reserve_1;
14    uint8_t reserve_2;
15 }tuya_ble_device_param_t;
```

The following items describe the variables in the preceding snippet.

- `device_id` and `auth_key`: the `device_uuid` and the `auth_key` are unique identifier pairs assigned to the BLE devices by the Tuya Developer Platform. They are required when you perform the production testing authorization via Tuya testing tool. After a BLE device is authorized by production testing, the tuya ble

sdk takes control of and manages them automatically. Therefore, you can set the `device_uuid` and the `auth_key` to 0 when the tuya ble sdk is initialized. If they are not 0, the tuya ble sdk uses the pre-defined `device_id` and `auth_key`. For a Wi-Fi/BLE dual-mode device, its `device_id` and `auth_key` are managed by the Wi-Fi layer, therefore, the actual `device_id` and `auth_key` are needed when the tuya ble sdk is initialized.

- `product_id`: pid for short. It is automatically generated when a new product is created on the Developer Platform. In your Application, the PID needs to be saved in the form of a constant. It is required when initializing the tuya ble sdk.
- `device_vid`: the virtual ID of a BLE device, which is generated by Tuya Developer Platform after the binding of the notice volume. It can be used to find the data record of the cloud to the BLE device when the BLE device is added or removed. For BLE devices that need the authorization information of the tuya ble sdk management, the value is 0. For Wi-Fi/BLE dual mode devices, or BLE devices of which the authorization is managed by your Application, the actual `device_vid` is needed.
- `login_key` and `bound_flag`: if the authorization of BLE devices are managed by the tuya ble sdk, set the values to 0. For Wi-Fi/BLE dual mode devices, or BLE devices of which the authorization is managed by your Application, the actual values are needed.
- `firmware_version` and `hardware_version`: indicates the firmware version number and the hardware (PCBA) version number of BLE devices. They are required when initializing the tuya ble sdk. Currently, BLE devices only support two digit version numbers, for example 0x0100 represents v1.0.

The following snippet is the initialization example of tuya ble sdk for nrf52832 platform.


```
1 static const char auth_key_test[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
2 static const char device_id_test[] = "yyyyyyyyyyyyyyyyyy";
3
4 #define APP_PRODUCT_ID "vvvvvvvv"
5
6 void tuya_ble_app_init(void)
7 {
8     tuya_ble_device_param_t device_param = {0};
9     device_param.device_id_len = 16;
10    memcpy(device_param.auth_key, (void *)auth_key_test, AUTH_KEY_LEN);
11    memcpy(device_param.device_id, (void *)device_id_test, DEVICE_ID_LEN);
12
13    device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
14    device_param.product_id_len = 8;
15    memcpy(device_param.product_id, APP_PRODUCT_ID, 8);
16    device_param.firmware_version = TY_APP_VER_NUM;
17    device_param.hardware_version = TY_HARD_VER_NUM;
18
19    tuya_ble_sdk_init(&device_param);
20    tuya_ble_callback_queue_register(tuya_cb_handler);
21
22    tuya_ota_init();
23 }
24
25 /*nrf52832 example*/
26 int main(void)
27 {
28     bool erase_bonds;
29
30     // Initialize.
31     uart_init();
32     app_log_init();
33     timers_init();
34     buttons_leds_init(&erase_bonds);
35     power_management_init();
36     ble_stack_init();
37     gap_params_init();
38     gatt_init();
39     services_init();
40     advertising_init();
41     conn_params_init();
42     tuya_ble_app_init();
43     advertising_start();
44
45     NRF_LOG_INFO("App version: %s\r\n", TY_APP_VER_STR);
46
47     // Enter main loop.
48     for (;;)
49     {
50         idle_state_handle();
51     }
```

4.0.10 tuy_ble_sdk_init_async

Function name	tuya_ble_sdk_init_async
Prototype	void tuya_ble_sdk_init_async(tuya_ble_device_param_t * param_data, tuya_ble_nv_async_callback_t callback)
Description	The tuya ble sdk initialization function, mainly used in tuya ble sdk of asynchronous flash operation architecture.
Parameters	param_data [in]: Initialize parameter. callback[in]: Callback, This callback will be executed automatically after the initialization is completed.
Responses	None.
Notice	The tuya ble sdk Asynchronous initialization function, the Application must call this function to initialize the SDK, otherwise the SDK will not work.

Example of tuya ble sdk asynchronous initialization:

```
1 static const char auth_key_test[] = "xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx";
2 static const char device_id_test[] = "yyyyyyyyyyyyyyyy";
3
4 #define APP_PRODUCT_ID "vvvvvvvv"
5
6 static void tuya_ble_sdk_init_async_completed(void *p_param,
7        tuya_ble_status_t result)
8 {
9     if(result==TUYA_BLE_SUCCESS)
10     {
11         tuya_ble_callback_queue_register(tuya_cb_handler);
12
13         tuya_ota_init();
14
15         TUYA_APP_LOG_INFO("tuya sdk init succeed.");
16         TUYA_APP_LOG_INFO("App version: "TY_APP_VER_STR);
17         TUYA_APP_LOG_DEBUG("current free heap size = %d",
18             xTuyaPortGetFreeHeapSize());
19         //tuya_ble_device_factory_reset();
20     }
21     else
22     {
23         TUYA_APP_LOG_INFO("tuya sdk init failed.");
24     }
25 }
26
27 void tuya_ble_app_init(void)
28 {
29     tuya_ble_nv_init_custom();
30     memset(&device_param,0,sizeof(tuya_ble_device_param_t));
31     device_param.device_id_len = 16;
32     memcpy(device_param.auth_key,(void *)auth_key_test,AUTH_KEY_LEN);
33     memcpy(device_param.device_id,(void *)device_id_test,DEVICE_ID_LEN)
34     ;
35
36     device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
37     device_param.product_id_len = 8;
38     memcpy(device_param.product_id,APP_PRODUCT_ID,8);
39     device_param.firmware_version = TY_APP_VER_NUM;
40     device_param.hardware_version = TY_HARD_VER_NUM;
41
42     tuya_ble_sdk_init_async(&device_param,tuya_ble_sdk_init_async_completed);
43
44 }
45
46 /*nrf52832 example*/
47 int main(void)
48 {
49     bool erase_bonds;
50
51     // Initialize.
52     uart_init();
```

```

1  app_log_init();
2  timers_init();
3  buttons_leds_init(&erase_bonds);
4  power_management_init();
5  ble_stack_init();
6  gap_params_init();
7  gatt_init();
8  services_init();
9  advertising_init();
10 conn_params_init();
11 tuya_ble_app_init();
12 advertising_start();
13
14 NRF_LOG_INFO("App version: %s\r\n", TY_APP_VER_STR);
15
16 // Enter main loop.
17 for (;;)
18 {
19     idle_state_handle();
20 }
21 }
```

4.0.11 tuya_ble_dp_data_report

Function name	tuya_ble_dp_data_report
Prototype	tuya_ble_status_t tuya_ble_dp_data_report(uint8_t *p_data,uint32_t len)
Description	Reports dp point data to APP.
Parameters	p_data [in]: dp point data. len[in]: Data length, Cannot exceed TUYA_BLE_REPORT_MAX_DP_DATA_LEN.

Function name	tuya_ble_dp_data_report
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter invalid.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection.</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error .</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>
Notice	Application reports DP data to mobile App by calling this function.

parameter description:

- Tuya Developer Platform manages data in the form of dp points. The data generated by any device needs to be abstracted in the form of dp points. A complete dp point data consists of four parts (refer to the relevant introduction on the Developer Platform for details):

Dp_id: 1 byte, dp_id serial number registered in the development platform.

Dp_type: 1 byte, dp point type.

```
#define DT_RAW 0 //raw type
```

```
#define DT_BOOL 1 //Boolean type
```

```
#define DT_VALUE 2 //Value type
```

```
#define DT_STRING 3 //String type
```

```
#define DT_ENUM 4 //Enumeration type
```

```
#define DT_BITMAP 5 // Bitmap type
```

Dp_len: Bluetooth currently supports only one byte, that is, the longest data of a single dp point is 255 bytes.

Dp_data: Data, dp_len bytes.

- The dp point reporting function, the data pointed to by parameter p_data must be assembled and reported in the form of a table:

Dp point 1 data ~					Dp point n data			
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

- When calling this function, the maximum length of parameter len is `TUYA_BLE_REPORT_MAX_DP_DATA_LEN` (currently 255+3).

4.0.12 tuya_ble_dp_data_with_time_report

Function name	tuya_ble_dp_data_with_time_report
Prototype	<pre>tuya_ble_status_t tuya_ble_dp_data_with_time_report(uint32_t timestamp,uint8_t *p_data,uint32_t len)</pre>
Description	Reports dp point data with time stamp.
Parameters	<p>timestamp[in]: 4 byte Unix timestamp.</p> <p>p_data [in] :dp data.</p> <p>len[in]: The data length cannot exceed <code>TUYA_BLE_REPORT_MAX_DP_DATA_LEN</code>.</p>

Function name	tuya_ble_dp_data_with_time_report
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection;</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error.</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>
Notice	<p>The Application code reports dp point data to the App with time stamp by calling this function.</p> <p>Generally, data that is cached offline needs to be reported with time stamp.</p>

4.0.13 tuya_ble_dp_data_with_time_ms_string_report

Function name	tuya_ble_dp_data_with_time_ms_string_report
Prototype	<pre>tuya_ble_status_t tuya_ble_dp_data_with_time_ms_string_report(uint8_t time_string,uint8_t p_data,uint32_t len)</pre>
Description	Reports dp point data with string format time.
Parameters	<p>time_string[in]: 13 byte ms level string format time.</p> <p>p_data [in]: dp point data.</p> <p>Len[in]: Data length, the maximum cannot exceed</p> <p>TUYA_BLE_REPORT_MAX_DP_DATA_LEN.</p>

Function name	tuya_ble_dp_data_with_time_ms_string_report
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection.</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error.</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>
Notice	<p>The Application code reports dp point data to the App with string format time by calling this function.</p> <p>generally offline data only needs to be reported with time.</p> <p>13 byte ms level time string, such as "0000000123456" ", less than 13 bytes before the supplementary character 0.</p>

4.0.14 tuya_ble_dp_data_with_flag_report

Function name	tuya_ble_dp_data_with_flag_report
Prototype	<pre>tuya_ble_status_t tuya_ble_dp_data_with_flag_report(uint16_t sn,tuya_ble_report_mode_t mode,uint8_t *p_data,uint32_t len)</pre>
Description	Reports dp point data with flag.

Function name	tuya_ble_dp_data_with_flag_report
Parameters	<p>sn[in]: Application defined serial number.</p> <p>mode[in]: Reporting mode.</p> <p>p_data [in]: dp point data.</p> <p>len[in]: dp point data length, the maximum cannot exceed <code>TUYA_BLE_REPORT_MAX_DP_DATA_LEN</code>.</p>
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection;</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error.</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>
Notice	Application calls this function to report the marked DP point data to the App.

parameter description:

mode:

`REPORT_FOR_CLOUD_PANEL`: report to the panel and the cloud at the same time.

`REPORT_FOR_CLOUD`: only report to the cloud.

`REPORT_FOR_PANEL`: only report to the panel.

`REPORT_FOR_NONE`: neither reported to the panel nor to the cloud.

4.0.15 tuya_ble_dp_data_with_flag_and_time_report

Function name	tuya_ble_dp_data_with_flag_and_time_report
Prototype	<pre>tuya_ble_status_t tuya_ble_dp_data_with_flag_and_time_report (uint16_t sn,tuya_ble_report_mode_t mode,uint32_t timestamp,uint8_t *p_data,uint32_t len)</pre>
Description	Reports dp point data with time stamp.
Parameters	<p>sn[in]: Application defined serial number.</p> <p>mode[in]: reporting mode.</p> <p>timestamp[in]: 4 byte Unix timestamp.</p> <p>p_data [in]: dp data.</p> <p>len[in]: dp point data length, the maximum cannot exceed TUYA_BLE_REPORT_MAX_DP_DATA_LEN.</p>
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection.</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error.</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>
Notice	<p>The Application code reports dp point data to the App with timestamp by calling this function.</p> <p>Generally, data that is cached offline needs to be reported with a time stamp.</p>

4.0.16 tuy BLE dp data with flag and time ms string report

Function name	tuya_ble_dp_data_with_flag_and_time_ms_string_report
Prototype	<pre> tuya_ble_status_t tuya_ble_dp_data_with_flag_and_time_ms_string_report(uint16_t sn, tuya_ble_report_mode_t mode, uint8_t time_string, uint8_t p_data, uint32_t len) </pre>
Description	Reports dp point data with string formatted time with flag.
Parameters	<p>sn[in]: Application defined serial number.</p> <p>mode[in]: reporting mode.</p> <p>time_string[in]: 13-byte ms-level string format time;</p> <p>p_data [in]: dp point data.</p> <p>Len[in]: data length, the maximum cannot exceed TUYA_BLE_REPORT_MAX_DP_DATA_LEN.</p>
Responses	<p>TUYA_BLE_SUCCESS: Sent successfully.</p> <p>TUYA_BLE_ERR_INVALID_PARAM: parameter error.</p> <p>TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection.</p> <p>TUYA_BLE_ERR_NO_MEM: Memory Application failed.</p> <p>TUYA_BLE_ERR_INVALID_LENGTH: Data length error.</p> <p>TUYA_BLE_ERR_NO_EVENT: Other errors.</p>

Function name	tuya_ble_dp_data_with_flag_and_time_ms_string_report
----------------------	--

Notice

The Application code reports dp point data to the App with string format time by calling this function.
generally offline data only needs to be reported with time.
13 byte ms level time string, such as "0000000123456" ", less than 13 bytes before the supplementary character 0.

4.0.17 tuya_ble_connected_handler

Function name	tuya_ble_connected_handler
----------------------	----------------------------

Prototype

void tuya_ble_connected_handler(void)

Description

Bluetooth connection callback function.

Parameters

None.

Responses

None.

Notice

The Application code needs to call this function at the Bluetooth connection callback of the chip platform SDK.
tuya ble sdk is based on this function to manage the internal Bluetooth connection state.

4.0.18 tuya_ble_disconnected_handler

Function name	tuya_ble_disconnected_handler
----------------------	-------------------------------

Prototype

void
tuya_ble_disconnected_handler(void)

Description

Bluetooth disconnection callback function.

Function name	tuya_ble_disconnected_handler
Parameters	None.
Responses	None.
Notice	The Application code needs to call this function at the Bluetooth disconnection callback of the chip platform SDK. The tuya ble sdk manages the internal Bluetooth connection state according to the execution of this function.

Example of calling tuya_ble_connected_handler and tuya_ble_disconnected_handler on nrf52832 platform:

```
1  /**@brief Function for handling BLE events.
2   *
3   * @param[in]   p_ble_evt   Bluetooth stack event.
4   * @param[in]   p_context   Unused.
5   */
6  static void ble_evt_handler(ble_evt_t const * p_ble_evt, void *
7                               p_context)
8  {
9      uint32_t err_code;
10
11      switch (p_ble_evt->header.evt_id)
12      {
13      case BLE_GAP_EVT_CONNECTED:
14          NRF_LOG_INFO("Connected");
15
16          tuya_ble_connected_handler();
17
18          err_code = bsp_indication_set(BSP_INDICATE_CONNECTED);
19          APP_ERROR_CHECK(err_code);
20          m_conn_handle = p_ble_evt->evt.gap_evt.conn_handle;
21          err_code = nrf_ble_qwr_conn_handle_assign(&m_qwr, m_conn_handle
22                                                    );
23          APP_ERROR_CHECK(err_code);
24          break;
25
26      case BLE_GAP_EVT_DISCONNECTED:
27          NRF_LOG_INFO("Disconnected");
28
29          tuya_ble_disconnected_handler();
30
31          tuya_ota_init_disconnect();
32          // LED indication will be changed when advertising starts.
33          m_conn_handle = BLE_CONN_HANDLE_INVALID;
34          break;
35
36      case BLE_GAP_EVT_PHY_UPDATE_REQUEST:
37      {
38          NRF_LOG_DEBUG("PHY update request.");
39          ble_gap_phys_t const phys =
40          {
41              .rx_phys = BLE_GAP_PHY_AUTO,
42              .tx_phys = BLE_GAP_PHY_AUTO,
43          };
44          err_code = sd_ble_gap_phy_update(p_ble_evt->evt.gap_evt.
45                                          conn_handle, &phys);
46          APP_ERROR_CHECK(err_code);
47      }
48      break;
49
50      case BLE_GAP_EVT_SEC_PARAMS_REQUEST:
51          // Pairing not supported
```

```

1      err_code = sd_ble_gap_sec_params_reply(m_conn_handle,
2      BLE_GAP_SEC_STATUS_PAIRING_NOT_SUPP, NULL, NULL);
3      APP_ERROR_CHECK(err_code);
4      break;
5
6      case BLE_GATTS_EVT_SYS_ATTR_MISSING:
7          // No system attributes have been stored.
8          err_code = sd_ble_gatts_sys_attr_set(m_conn_handle, NULL, 0, 0)
9          ;
10         APP_ERROR_CHECK(err_code);
11         break;
12
13     case BLE_GATTC_EVT_TIMEOUT:
14         // Disconnect on GATT Client timeout event.
15         err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gattc_evt.
16         conn_handle,
17         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION
18         );
19         APP_ERROR_CHECK(err_code);
20         break;
21
22     case BLE_GATTS_EVT_TIMEOUT:
23         // Disconnect on GATT Server timeout event.
24         err_code = sd_ble_gap_disconnect(p_ble_evt->evt.gatts_evt.
25         conn_handle,
26         BLE_HCI_REMOTE_USER_TERMINATED_CONNECTION
27         );
28         APP_ERROR_CHECK(err_code);
29         break;
30
31     default:
32         // No implementation needed.
33         break;
34 }
35 }
```

4.0.19 tuya_ble_adv_data_connecting_request_set

Function name	tuya_ble_adv_data_connecting_request_set
Prototype	tuya_ble_status_t tuya_ble_adv_data_connecting_request_set(uint8_t on_off)
Description	Triggers broadcast packet change request connection flag.

Function name	tuya_ble_adv_data_connecting_request_set
Parameters	on_off[in]: 0 - Clear flag, 1 - set flag.
Responses	TUYA_BLE_SUCCESS: success. TUYA_BLE_ERR_INVALID_STATE: The current state is not supported, for example, it is currently connected. TUYA_BLE_ERR_INVALID_PARAM: parameter error. TUYA_BLE_ERR_NO_EVENT: Other errors.

4.0.20 tuya_ble_data_passthrough

Function name	tuya_ble_data_passthrough
Prototype	tuya_ble_status_t tuya_ble_data_passthrough(uint8_t *p_data, uint32_t len)
Description	Passes through the custom data of the Application.
Parameters	p_data [in]: data pointer that needs to be transparently transmitted. len[in]: data length, the maximum cannot exceed TUYA_BLE_TRANSMISSION_MAX_DATA_LEN .
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_INVALID_LENGTH: Data length error. TUYA_BLE_ERR_NO_EVENT: Other errors.

Function name	tuya_ble_data_passthrough
Notice	Application transparently transmits data to App by calling this function. The transparently transmitted data format is negotiated by the device Application and the mobile App, and tuya ble sdk does not parse it.

4.0.21 tuya_ble_production_test_asynchronous_response

Function name	tuya_ble_production_test_asynchronous_response
Prototype	tuya_ble_status_t tuya_ble_production_test_asynchronous_response (uint8_t channel,uint8_t *p_data,uint32_t len)
Description	Asynchronously respond to the production testing command.
Parameters	Channel[in]: Transmission channel, 0-uart;1-ble. p_data [in]: complete command data that needs to be responded to. len[in]: Data length.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_INVALID_LENGTH: Data length error. TUYA_BLE_ERR_NO_MEM: Failed to apply for memory. TUYA_BLE_ERR_NO_EVENT: Other errors.

Function name	tuya_ble_production_test_asynchronous_response
----------------------	--

Notice

When performing production testing (production testing authorization is through UART, the whole machine test passes BLE)
Some test items can not immediately respond to the results of the upper computer production testing tool, or some test items need to be processed by the Application. At this time, you need to call this function to send the test results to the upper computer production testing tool.

4.0.22 tuya_ble_net_config_response

Function name	tuya_ble_net_config_response
----------------------	------------------------------

Prototype	tuya_ble_status_t tuya_ble_net_config_response(int16_t result_code)
------------------	--

Description	Responds the Wi-Fi network configuration request.
--------------------	---

Parameters	Result_code[in]: status code.
-------------------	-------------------------------

Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_NO_EVENT: Other errors.
------------------	---

Notice	Suitable for Wi-Fi / BLE combo devices.
---------------	---

4.0.23 tuyabound_response

Function name	tuyabound_response
Prototype	tuya_ble_status_t tuyabound_response(uint8_t result_code)
Description	Responds to the removal requests of multi-protocol combo device.
Parameters	Result_code[in]: status code, 0 - success, 1 - fail.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_NO_EVENT: Other errors.
Notice	Suitable for Wi-Fi / BLE combo devices.

4.0.24 tuyabound_anomaly_response

Function name	tuyabound_anomaly_response
Prototype	tuya_ble_status_t tuyabound_anomaly_response (uint8_t result_code)
Description	Responds to the abnormal removal requests of multi-protocol combo device.
Parameters	Result_code[in]: status code, 0 - success, 1 - fail.

Function name	tuya_ble_anomaly_ubound_response
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_NO_EVENT: Other errors.
Notice	Suitable for Wi-Fi / BLE combo devices.

4.0.25 tuya_ble_device_reset_response

Function name	tuya_ble_device_reset_response
Prototype	tuya_ble_status_t tuya_ble_device_reset_response (uint8_t result_code)
Description	Responds to the reset requests of multi-protocol combo device.
Parameters	Result_code[in]: status code, 0 - success, 1 - fail.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: The current status does not support sending, such as Bluetooth disconnection. TUYA_BLE_ERR_NO_EVENT: Other errors.
Notice	Suitable for Wi-Fi / BLE combo devices.

4.0.26 tuya_ble_connect_status_get

Function name	tuya_ble_connect_status_get
Prototype	tuya_ble_connect_status_t tuya_ble_connect_status_get(void)
Description	Gets current BLE connection status
Parameters	None.
Responses	tuya_ble_connect_status_t . See the following response example for details.

Responses description:

```
1 typedef enum{
2     UNBONDING_UNCONN = 0,
3     UNBONDING_CONN,
4     BONDING_UNCONN,
5     BONDING_CONN,
6     BONDING_UNAUTH_CONN,
7     UNBONDING_UNAUTH_CONN,
8     UNKNOW_STATUS
9 }tuya_ble_connect_status_t;
```

4.0.27 tuya_ble_device_factory_reset

Function name	tuya_ble_device_factory_reset
Prototype	tuya_ble_status_t tuya_ble_device_factory_reset(void)
Description	Resets BLE devices.
Parameters	None.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INTERNAL: Other errors.

Function name

tuya_ble_device_factory_reset

Notice

For devices that define an external key reset function, the Application needs to call this function to notify tuya ble sdk to reset related information after the key triggers a reset, such as clearing binding information.

4.0.28 tuya_ble_time_req

Function name

tuya_ble_time_req

Prototype

```
tuya_ble_status_t
tuya_ble_time_req(uint8_t time_type)
```

Description

Requests cloud time.

Parameters

time_type[in]:
0 - Request time in 13-byte ms-level string format.
1 - Find the year, month, day, hour, minute, second, week format time.

Responses

TUYA_BLE_SUCCESS: Sent successfully.
TUYA_BLE_ERR_INVALID_PARAM: parameter error.
TUYA_BLE_ERR_INTERNAL: Other errors.

Function name	tuya_ble_time_req
Notice	<p>13 byte string time format: “0000000123456” means 123456 ms timestamp.</p> <p>normal time format: 0x13,0x04,0x1C,0x0C,0x17,0x19,0x02 means: April 28, 2019 :23:25 Tuesday.</p> <p>tuya ble sdk will send the corresponding instruction to the App after receiving the request, sdk will send a message to the device Application after receiving the time returned by the App.</p>

4.0.29 tuya_ble_ota_response

Function name	tuya_ble_ota_response
Prototype	tuya_ble_status_t tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)
Description	OTA response command.
Parameters	p_data[in]: OTA response data.
Responses	TUYA_BLE_SUCCESS: Sent successfully. TUYA_BLE_ERR_INVALID_STATE: State error. TUYA_BLE_ERR_INVALID_LENGTH: Data length error. TUYA_BLE_ERR_NO_MEM: Memory Application fail. TUYA_BLE_ERR_INTERNAL: Other errors.

Function name	tuya_ble_ota_response
Notice	See the OTA introduction chapter for the specific format.

4.0.30 tuya_ble_custom_event_send

Function name	tuya_ble_custom_event_send
Prototype	uint8_t tuya_ble_custom_event_send(tuya_ble_custom_evt_t evt)
Description	Sends Application custom message and callback to tuya ble sdk.
Parameters	evt[in]: Custom event.
Responses	0: Sent successfully. 1: fail.
Notice	typedef struct{ void *data; void (*custom_event_handler)(void*data); } tuya_ble_custom_evt_t;

Description: This function is mainly used when the Application wants to use the internal message scheduler of tuya ble sdk to process the Application' s message events.

[tuya_ble_custom_event_send](#) Application examples:


```
1  #define APP_CUSTOM_EVENT_1  1
2  #define APP_CUSTOM_EVENT_2  2
3  #define APP_CUSTOM_EVENT_3  3
4  #define APP_CUSTOM_EVENT_4  4
5  #define APP_CUSTOM_EVENT_5  5
6
7  typedef struct {
8      uint8_t data[50];
9  } custom_data_type_t;
10
11 void custom_data_process(int32_t evt_id,void *data)
12 {
13     custom_data_type_t *event_1_data;
14     TUYA_BLE_LOG_DEBUG("custom event id = %d",evt_id);
15     switch (evt_id)
16     {
17         case APP_CUSTOM_EVENT_1:
18             event_1_data = (custom_data_type_t *)data;
19             TUYA_BLE_LOG_HEXDUMP_DEBUG("received APP_CUSTOM_EVENT_1
20                                     data:",event_1_data->data,50);
21             break;
22         case APP_CUSTOM_EVENT_2:
23             break;
24         case APP_CUSTOM_EVENT_3:
25             break;
26         case APP_CUSTOM_EVENT_4:
27             break;
28         case APP_CUSTOM_EVENT_5:
29             break;
30         default:
31             break;
32     }
33 }
34
35 custom_data_type_t custom_data;
36
37 void custom_evt_1_send_test(uint8_t data)
38 {
39     tuya_ble_custom_evt_t event;
40
41     for(uint8_t i=0; i<50; i++)
42     {
43         custom_data.data[i] = data;
44     }
45     event.evt_id = APP_CUSTOM_EVENT_1;
46     event.custom_event_handler = (void *)custom_data_process;
47     event.data = &custom_data;
48     tuya_ble_custom_event_send(event);
49 }
```

4.0.31 tuya_ble_callback_queue_register

Function name	tuya_ble_callback_queue_register
Prototype	prototype 1: tuya_ble_status_t tuya_ble_callback_queue_register(void *cb_queue). prototype 2: tuya_ble_status_t tuya_ble_callback_queue_register(tuya_ble_callback_t cb).
Description	To register the message queue for receiving tuya ble sdk messages under the OS platform architecture, use prototype 1. To register callback function for receiving tuya ble sdk messages under OS-less platform architecture, use prototype 2.
Parameters	cb_queue [in]: message queue. cb[in]: callback Function address.
Responses	TUYA_BLE_ERR_RESOURCES: Register fail. TUYA_BLE_SUCCESS: Register success.

Application examples for OS platform:

```
1 void *tuya_custom_queue_handle;  
2  
3 os_msg_queue_create(&tuya_custom_queue_handle ,  
    MAX_NUMBER_OF TUYA_CUSTOM_MESSAGE , sizeof(tuya_ble_cb_evt_param_t));  
4  
5 tuya_ble_callback_queue_register(tuya_custom_queue_handle);
```

Application example without OS platform:

```
1 void tuya_cb_handler(tuya_ble_cb_evt_param_t* event).  
2  
3 tuya_ble_callback_queue_register(tuya_cb_handler);
```

4.0.32 tuya_ble_event_response

Function name	tuya_ble_event_response
Prototype	tuya_ble_status_t tuya_ble_event_response(tuya_ble_cb_evt_param_t *param)
Description	Responds to tuya ble sdk messages under the OS platform architecture.
Parameters	param [in]: message pointer.
Responses	TUYA_BLE_SUCCESS: success. other: fail.
Notice	Under the OS platform architecture, after the Application code processes the message sent by tuya ble sdk, Must call this function to give feedback to tuya ble sdk.

Application examples:

```
1  /*Application task to process ble sdk messages*/
2  void app_custom_task(void *p_param)
3  {
4      tuya_ble_cb_evt_param_t event;
5
6      while (true)
7      {
8          if (os_msg_rcv(tuya_custom_queue_handle, &event, 0xFFFFFFFF)
9              == true)
10         {
11             switch (event.evt)
12             {
13                 case TUYA_BLE_CB_EVT_CONNECTE_STATUS:
14                     break;
15                 case TUYA_BLE_CB_EVT_DP_WRITE:
16                     break;
17                 case TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE:
18                     break;
19                 case TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE:
20                     break;
21                 case TUYA_BLE_CB_EVT_UNBOUND:
22                     break;
23                 case TUYA_BLE_CB_EVT_ANOMALY_UNBOUND:
24                     break;
25                 case TUYA_BLE_CB_EVT_DEVICE_RESET:
26                     break;
27                 case TUYA_BLE_CB_EVT_DP_QUERY:
28                     break;
29                 case TUYA_BLE_CB_EVT_OTA_DATA:
30                     break;
31                 case TUYA_BLE_CB_EVT_NETWORK_INFO:
32                     break;
33                 case TUYA_BLE_CB_EVT_WIFI_SSID:
34                     break;
35                 case TUYA_BLE_CB_EVT_TIME_STAMP:
36                     break;
37                 case TUYA_BLE_CB_EVT_TIME_NORMAL:
38                     break;
39                 case TUYA_BLE_CB_EVT_DATA_PASSTHROUGH:
40                     break;
41                 default:
42                     break;
43             }
44             tuya_ble_event_response(&event);
45         }
46     }
47 }
48 }
```

4.0.33 tuya_ble_scheduler_queue_size_get

Function name	tuya_ble_scheduler_queue_size_get
Prototype	uint16_t tuya_ble_scheduler_queue_size_get(void)
Description	Gets the total size of tuya ble sdk scheduler queue in platforms that have no OS.
Parameters	None.
Responses	Total size of the scheduler queue.

4.0.34 tuya_ble_scheduler_queue_space_get

Function name	tuya_ble_scheduler_queue_space_get
Prototype	uint16_t tuya_ble_scheduler_queue_space_get(void)
Description	Gets the idle number of tuya ble sdk scheduler queue in platforms that have no OS.
Parameters	None.
Responses	Scheduler queue idle.

4.0.35 tuya_ble_scheduler_queue_events_get

Function name	tuya_ble_scheduler_queue_events_get
Prototype	uint16_t tuya_ble_scheduler_queue_events_get(void)
Description	Gets the number of unprocessed events in the tuya ble sdk scheduler queue in platforms that have no OS.

Function name	tuya_ble_scheduler_queue_events_get
Parameters	None.
Responses	The number of unprocessed events in the scheduler queue.

5 The callback event of tuya ble sdk

The tuya ble sdk sends message of status, data, and other information to Application through message (for OS architecture platform) or callback function registered by Application (for non-OS architecture platform). According to the previous section, the example of `tuya_ble_event_response` describes how the Application in the OS architecture platform handle the messages sent by tuya ble sdk. The chip platform without an OS can use the similar code to process messages, however it is not necessary to call the `tuya_ble_event_response()` to respond to tuya ble sdk after the messages is handled by the callback function. As the following snippet shows an example for callback registration in the platform without an OS.

```

1  /*Call back function for processing ble sdk messages*/
2  static void tuya_cb_handler(tuya_ble_cb_evt_param_t* event)
3  {
4      switch (event->evt)
5      {
6          case TUYA_BLE_CB_EVT_CONNECTE_STATUS:
7              break;
8          case TUYA_BLE_CB_EVT_DP_WRITE:
9              break;
10         case TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE:
11             break;
12         case TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE:
13             break;
14         case TUYA_BLE_CB_EVT_UNBOUND:
15             break;
16         case TUYA_BLE_CB_EVT_ANOMALY_UNBOUND:
17             break;
18         case TUYA_BLE_CB_EVT_DEVICE_RESET:
19             break;
20         case TUYA_BLE_CB_EVT_DP_QUERY:
21             break;
22         case TUYA_BLE_CB_EVT_OTA_DATA:
23             break;
24         case TUYA_BLE_CB_EVT_NETWORK_INFO:
25             break;
26         case TUYA_BLE_CB_EVT_WIFI_SSID:
27             break;
28         case TUYA_BLE_CB_EVT_TIME_STAMP:
29             break;
30         case TUYA_BLE_CB_EVT_TIME_NORMAL:
31             break;
32         case TUYA_BLE_CB_EVT_DATA_PASSTHROUGH:
33             break;
34         default:
35             break;
36     }
37 }
38
39 void tuya_ble_app_init(void)
40 {
41     device_param.device_id_len = 16;
42     memcpy(device_param.auth_key,(void *)auth_key_test,AUTH_KEY_LEN);
43     memcpy(device_param.device_id,(void *)device_id_test,
44            DEVICE_ID_LEN);
45     device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
46     device_param.product_id_len = 8;
47     memcpy(device_param.product_id,APP_PRODUCT_ID,8);
48     device_param.firmware_version = TY_APP_VER_NUM;
49     device_param.hardware_version = TY_HARD_VER_NUM;
50
51     tuya_ble_sdk_init(&device_param);

```



```

1      tuya_ble_callback_queue_register(tuya_cb_handler);
2
3      tuya_ota_init();
4  }
```

5.0.1 TUYA_BLE_CB_EVT_CONNECTE_STATUS

Event	TUYA_BLE_CB_EVT_CONNECTE_STATUS
Description	Every time the state changes, tuya ble sdk will send the event to the device Application

Corresponding data structure:

```

1  typedef enum{
2      UNBONDING_UNCONN = 0,
3      UNBONDING_CONN,
4      BONDING_UNCONN,
5      BONDING_CONN,
6      BONDING_UNAUTH_CONN,
7      UNBONDING_UNAUTH_CONN,
8      UNKNOW_STATUS
9  }tuya_ble_connect_status_t;
```

5.0.2 TUYA_BLE_CB_EVT_DP_WRITE

Event	TUYA_BLE_CB_EVT_DP_WRITE
Description	The tuya ble sdk received dp point data sent by mobile App.
Notice	See format tuya_ble_dp_data_report API introduction.

Corresponding data structure:

```
1  /*
2   * dp data  buffer:  (Dp_id,Dp_type,Dp_len,Dp_data),(Dp_id,Dp_type,
   *                    Dp_len,Dp_data),....
3   * */
4  typedef struct{
5      uint8_t *p_data;
6      uint16_t data_len;
7  }tuya_ble_dp_write_data_t;
```

5.0.3 TUYA_BLE_CB_EVT_DP_QUERY

Event	TUYA_BLE_CB_EVT_DP_QUERY
Description	The dp point to be checked, which is sent from the mobile App.
Notice	data_len=0 means to query all dp points, otherwise each byte pointed to by p_data represents a dp point to be queried, for example, data_len=3, p_data{0x01,0x02,0x03}, means to query dp_id=1, dp_id=2 , dp_id=3 data of 3 dp points.

Corresponding data structure:

```
1  /*
2   * query dp point data,if data_len is 0,means query all dp point
   *   data,otherwise query the dp point in p_data buffer.
3   * */
4  typedef struct{
5      uint8_t *p_data;
6      uint16_t data_len;
7  }tuya_ble_dp_query_data_t;
```

5.0.4 TUYA_BLE_CB_EVT_OTA_DATA

Event	TUYA_BLE_CB_EVT_OTA_DATA
Description	tuya ble sdk received the OTA data sent by the mobile App.
Notice	See the OTA section for details.

Corresponding data structure:

```

1 typedef struct{
2     tuya_ble_ota_data_type_t type;
3     uint16_t data_len;
4     uint8_t *p_data;
5 }tuya_ble_ota_data_t;
```

5.0.5 TUYA_BLE_CB_EVT_NETWORK_INFO

Event	TUYA_BLE_CB_EVT_NETWORK_INFO
Description	The tuya ble sdk received Wi-Fi distribution information sent by the mobile app, for example: {"wifi_ssid":"tuya","password":"12345678","token":"xxxxxxxxxx".}
Notice	Only applicable to Wi-Fi / BLE dual protocol combo devices.

Corresponding data structure:

```

1 /*
2  * network data,unformatted json data,for example " {"wifi_ssid":"
3  * */
4 typedef struct{
5     uint16_t data_len;//include '\0'
6     uint8_t *p_data;
7 }tuya_ble_network_data_t;
```

5.0.6 TUYA_BLE_CB_EVT_WIFI_SSID

Event	TUYA_BLE_CB_EVT_WIFI_SSID
Description	tuya ble sdk received Wi-Fi distribution information sent by the mobile app, for example: “{” wifi_ssid” : “tuya” , “password” : “12345678” } “.
Notice	Only applicable to Wi-Fi / BLE dual protocol combo devices, Compared with TUYA_BLE_CB_EVT_NETWORK_INFO, it lacks the token field, and is mainly used for Wi-Fi SSID update of the network-equipped devices.

Corresponding data structure:

```
1 /*
2  * wifi ssid data,unformatted json data,for example " {"wifi_ssid
3  * */
4  typedef struct{
5      uint16_t data_len;//include '\0'
6      uint8_t *p_data;
7  }tuya_ble_wifi_ssid_data_t;
```

5.0.7 TUYA_BLE_CB_EVT_TIME_STAMP

Event	TUYA_BLE_CB_EVT_TIME_STAMP
Description	The tuya ble sdk received the timestamp in the format of a string sent by the mobile phone app, for example, 0000000123456 means 123456ms, Unix timestamp at the millisecond level.
Notice	time_zone The time zone is 100 times the actual time zone, for example, -8 zone is -800.

Corresponding data structure:

```

1  /*
2   * Unix timestamp
3   * */
4  typedef struct{
5      uint8_t timestamp_string[14];
6      int16_t time_zone;    //actual time zone Multiply by 100.
7  }tuya_ble_timestamp_data_t;

```

5.0.8 TUYA_BLE_CB_EVT_TIME_NORMAL

Event	TUYA_BLE_CB_EVT_TIME_NORMAL
Description	<p>tuya ble sdk received the time in the conventional format sent by the mobile app, for example: UTC +8, Tuesday, April 28, 2019 12:23:25. corresponding data: 0x13, 0x04, 0x1C, 0x0C, 0x17, 0x19, 0x02 (week), 0x0320 (time zone time_zone).</p>

Corresponding data structure:

```

1  /*
2   * normal time formatted
3   * */
4  typedef struct{
5      uint16_t nYear;
6      uint8_t nMonth;
7      uint8_t nDay;
8      uint8_t nHour;
9      uint8_t nMin;
10     uint8_t nSec;
11     uint8_t DayIndex; /* 0 = Sunday */
12     int16_t time_zone;    //actual time zone Multiply by 100.
13 }tuya_ble_time_noram1_data_t;

```

5.0.9 TUYA_BLE_CB_EVT_DATA_PASSTHROUGH

Event	TUYA_BLE_CB_EVT_DATA_PASSTHROUGH
Description	The tuya ble sdk received the transparent transmission data sent by the mobile app.
Notice	The tuya ble sdk does not parse transparently transmitted data, and the data format is negotiated and defined by the device Application and mobile App.

Corresponding data structure:

```

1 typedef struct{
2     uint16_t data_len;
3     uint8_t *p_data;
4 }tuya_ble_passthrough_data_t;

```

5.0.10 TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_REPORT_RESPONSE
Description	After the Device Application calls the <code>tuya_ble_dp_data_report()</code> function to send dp point data, if you need to confirm whether it is sent successfully, you need to wait for the Event. status=0 means success, and other failures. If the Device Application waits for the sending result, you need to add a timeout mechanism. Unlimited waiting.

Corresponding data structure:

```
1 typedef struct{
2     uint8_t status;
3 }tuya_ble_dp_data_report_response_t;
```

5.0.11 TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RESPONSE

Event

TUYA_BLE_CB_EVT_DP_DATA_WTTH_TIME_REPORT_RE

Description

Device Application calls `tuya_ble_dp_data_with_time_report()` and `tuya_ble_dp_data_with_time_ms_string_report()` function after sending the time-stamped dp point data, if you need to confirm whether it is sent successfully, you need to wait for the Event, status=0 means success, other failure, device Application You must add a timeout mechanism for waiting for the results to be sent, and you can't wait indefinitely.

Corresponding data structure:

```
1 typedef struct{
2     uint8_t status;
3 }tuya_ble_dp_data_with_time_report_response_t;
```

5.0.12 TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_REPORT_RESPONSE

EventTUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_REPORT_RE

Description

After the Application calls the `tuya_ble_dp_data_with_flag_report()` function to send dp point data, if you need to confirm whether it is sent successfully, wait for the Event. The `sn` and the `mode` are the serial number and mode of dp point data sent by the Application. `status=0` indicates success, other values indicate failures.

Notice

Device Application can determine which response point of dp point data is sent according to the serial number.

Corresponding data structure:

```
1 typedef struct{
2     uint16_t sn;
3     tuya_ble_report_mode_t mode;
4     uint8_t status;
5 }tuya_ble_dp_data_with_flag_report_response_t;
```

5.0.13 TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_AND_TIME_REPORT_RESPONSE

Event	TUYA_BLE_CB_EVT_DP_DATA_WITH_FLAG_AND_TIME_P
--------------	--

Description

Device Application calls

`tuya_ble_dp_data_with_flag_and_time_report`
() and

`tuya_ble_dp_data_with_flag_and_time_ms_string_report`

() function after sending dp point data with flag and timestamp, if you need to confirm whether Sent successfully, you need to wait for this Event;

sn and mode are device Application Sending dp point data is the incoming serial number and mode; status=0 means success, other failures.

Notice

Device Application can determine which response point of dp point data is sent according to the serial number.

Corresponding data structure:

```
1 typedef struct{
2     uint16_t sn;
3     tuya_ble_report_mode_t mode;
4     uint8_t status;
5 }tuya_ble_dp_data_with_flag_and_time_report_response_t;
```

5.0.14 TUYA_BLE_CB_EVT_UNBOUND

Event

TUYA_BLE_CB_EVT_UNBOUND

Description

Receiving the Event means that the mobile phone App sends an instruction to learn the binding. The data field is a reserved field and is not used for the time being.

Corresponding data structure:

```
1 typedef struct{
2     uint8_t data;
3 }tuya_ble_unbound_data_t;
```

5.0.15 TUYA_BLE_CB_EVT_ANOMALY_UNBOUND

Event	TUYA_BLE_CB_EVT_ANOMALY_UNBOUND
Description	Receiving the Event indicates that the mobile App sent an abnormal unbinding instruction, in which the data field is a reserved field and is not used for the time being.

Corresponding data structure:

```
1 typedef struct{
2     uint8_t data;
3 }tuya_ble_anomaly_unbound_data_t;
```

5.0.16 TUYA_BLE_CB_EVT_DEVICE_RESET

Event	TUYA_BLE_CB_EVT_DEVICE_RESET
Description	Receiving the Event indicates that the mobile App has sent a reset command, where the data field is a reserved field and is not used for the time being.
Notice	After the device Application receives the reset event, it needs to perform some operations defined by the reset function.

Corresponding data structure:

```
1 typedef struct{
2     uint8_t data;
3 }tuya_ble_device_reset_data_t;
```

5.0.17 TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID

Event	TUYA_BLE_CB_EVT_UPDATE_LOGIN_KEY_VID
Description	After the Notification book is bound to success, the mobile App will send the login key and device virtual ID to the device.
Notice	The BLE device does not need to process this information.

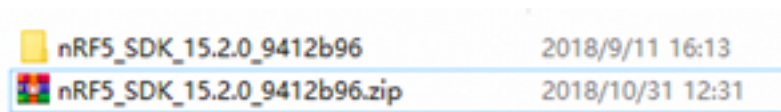
Corresponding data structure:

```
1 typedef struct{
2     uint8_t login_key_len;
3     uint8_t vid_len;
4     uint8_t login_key[LOGIN_KEY_LEN];
5     uint8_t vid[DEVICE_VIRTUAL_ID_LEN];
6 }tuya_ble_login_key_vid_data_t;
```























6 Example of SDK port in nrf52832

This section takes nrf52832 as an example to describe the porting steps for the OS-free architecture platforms. You can contact your account manager in Tuya for the detailed procedure of nrf52832 and the demo of other platforms.
























1. Download the original SDK of nrf52832 chip, for example, nRF5_SDK_15.2.0_9412b96, and prepare a nrf52832 development board.
2. Unzip the SDK to a custom directory, as shown in the following picture.



3. Open the **examples** > **ble_peripheral** directories. In this directory, you can see tutorials of ble peripheral.

 ble_app_blinky	2018/9/8 12:42
 ble_app_bms	2018/9/8 12:44
 ble_app_bps	2018/9/8 12:46
 ble_app_buttonless_dfu	2018/9/8 12:06
 ble_app_cscs	2018/9/8 12:48
 ble_app_cts_c	2018/9/8 12:49
 ble_app_eddystone	2018/9/8 12:52
 ble_app_gatts_c	2018/9/8 12:54
 ble_app_gls	2018/9/8 12:56
 ble_app_hids_keyboard	2018/9/8 12:59
 ble_app_hids_mouse	2018/9/8 13:01
 ble_app_hrs	2018/9/8 13:03
 ble_app_hrs_freertos	2018/9/8 13:07
 ble_app_hts	2018/9/8 13:08
 ble_app_ias_c	2018/9/8 13:10
 ble_app_ipsp_acceptor	2018/9/8 13:11
 ble_app_proximity	2018/9/8 13:12
 ble_app_pwr_profiling	2018/9/8 13:14
 ble_app_rscs	2018/9/8 13:15
 ble_app_template	2018/9/8 13:17
 ble_app_uart	2018/9/8 13:18
 experimental	2018/9/8 11:15
















4. Create a project with `ble_app_uart` as the template, and copy the `ble_app_uart` folder and rename it (for example, `tuya_ble_standard_nordic`).

	ble_app_bms	2018/9/8 12:44
	ble_app_bps	2018/9/8 12:46
	ble_app_buttonless_dfu	2018/9/8 12:06
	ble_app_cscs	2018/9/8 12:48
	ble_app_cts_c	2018/9/8 12:49
	ble_app_eddystone	2018/9/8 12:52
	ble_app_gatts_c	2018/9/8 12:54
	ble_app_gls	2018/9/8 12:56
	ble_app_hids_keyboard	2018/9/8 12:59
	ble_app_hids_mouse	2018/9/8 13:01
	ble_app_hrs	2018/9/8 13:03
	ble_app_hrs_freertos	2018/9/8 13:07
	ble_app_hts	2018/9/8 13:08
	ble_app_ias_c	2018/9/8 13:10
	ble_app_ipsp_acceptor	2018/9/8 13:11
	ble_app_proximity	2018/9/8 13:12
	ble_app_pwr_profiling	2018/9/8 13:14
	ble_app_rscs	2018/9/8 13:15
	ble_app_template	2018/9/8 13:17
	ble_app_uart	2018/9/8 13:18
	experimental	2018/9/8 11:15
	tuya_ble_sdk_Demo_Project_nrf52832	2020/5/7 21:25
	tuya_ble_standard_nordic	2020/4/20 16:52

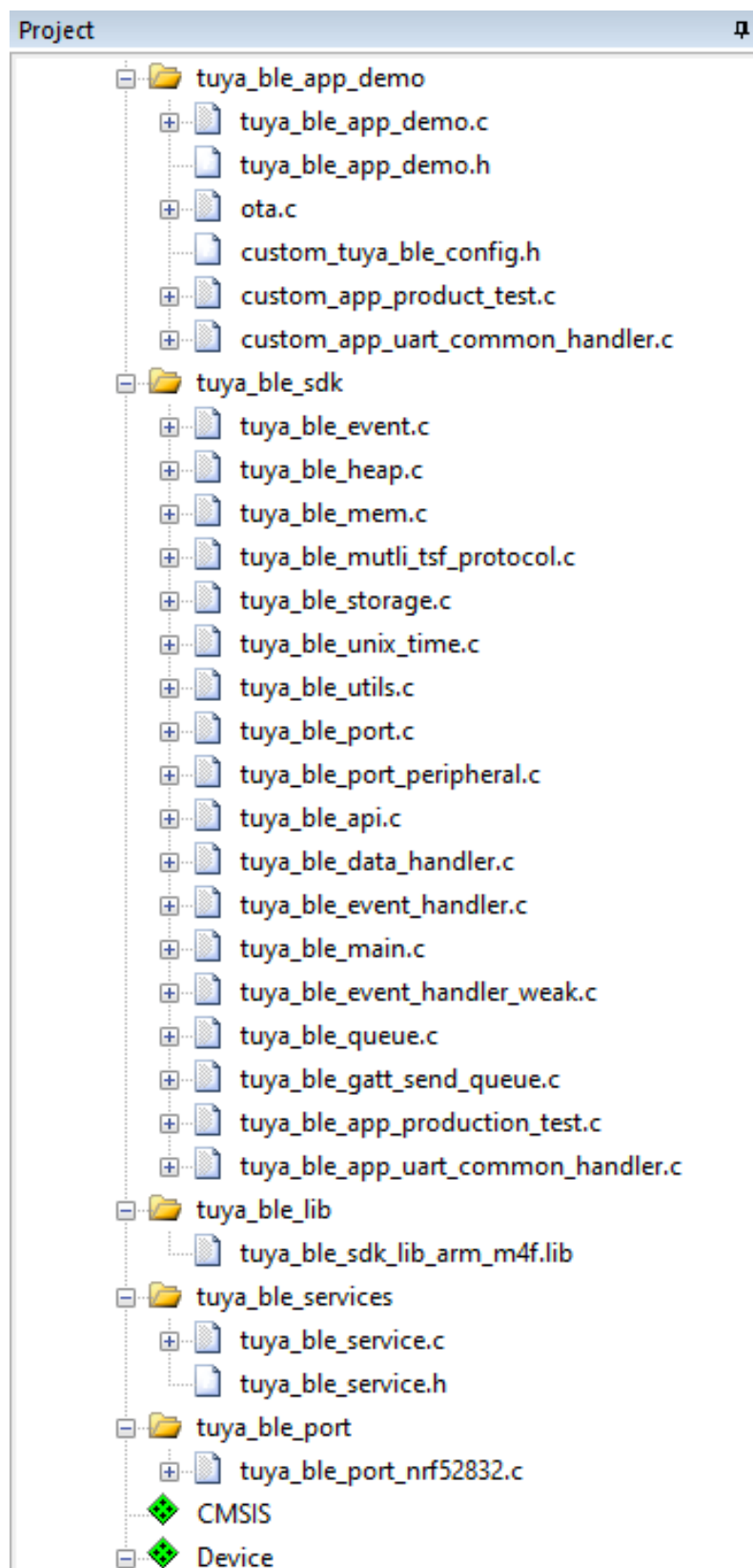
5. Enter the **PCA10040** > **S123** directories, open the project and compile it. You must confirm that it can be compiled and run correctly on the development board.
6. Create a file with the `ble_nus.c` in the **nRF_BLE_Services** directory as the template and `tuya_ble_service.c` as the name, modify your code to cater for the `tuya ble service`, modify the code in the `main.c` file and broadcast the changes

according the broadcasting content specified in the previous introduction.

7. Compile and download to your development board and run, use Bluetooth scanning tool in your mobile phone (for example, LightBlue for iOS) to scan devices. Make sure that your scanning result meet the requirement of broadcasting and service.
8. Download the tuyu ble sdk to the directory of a new project, add the source files to the project and compile for once.

	bootloader_project	2020/3/19 15:30
	extern_components	2020/3/19 15:30
	hex	2020/3/19 15:30
	pca10040	2020/3/19 15:30
	SDK15.2.0_tuya_patch	2020/3/19 15:30
	tuya_ble_app	2020/4/22 10:20
	tuya_ble_sdk	2020/3/19 15:30
	tuya_ble_services	2020/3/19 15:30
	flash.c	2019/12/26 15:00
	flash.h	2019/7/18 21:20
	main.c	2020/4/20 16:52
	main.h	2019/12/4 17:26
	README.md	2020/3/19 11:04
	rtc.c	2019/11/20 20:44
	rtc.h	2019/5/20 11:18

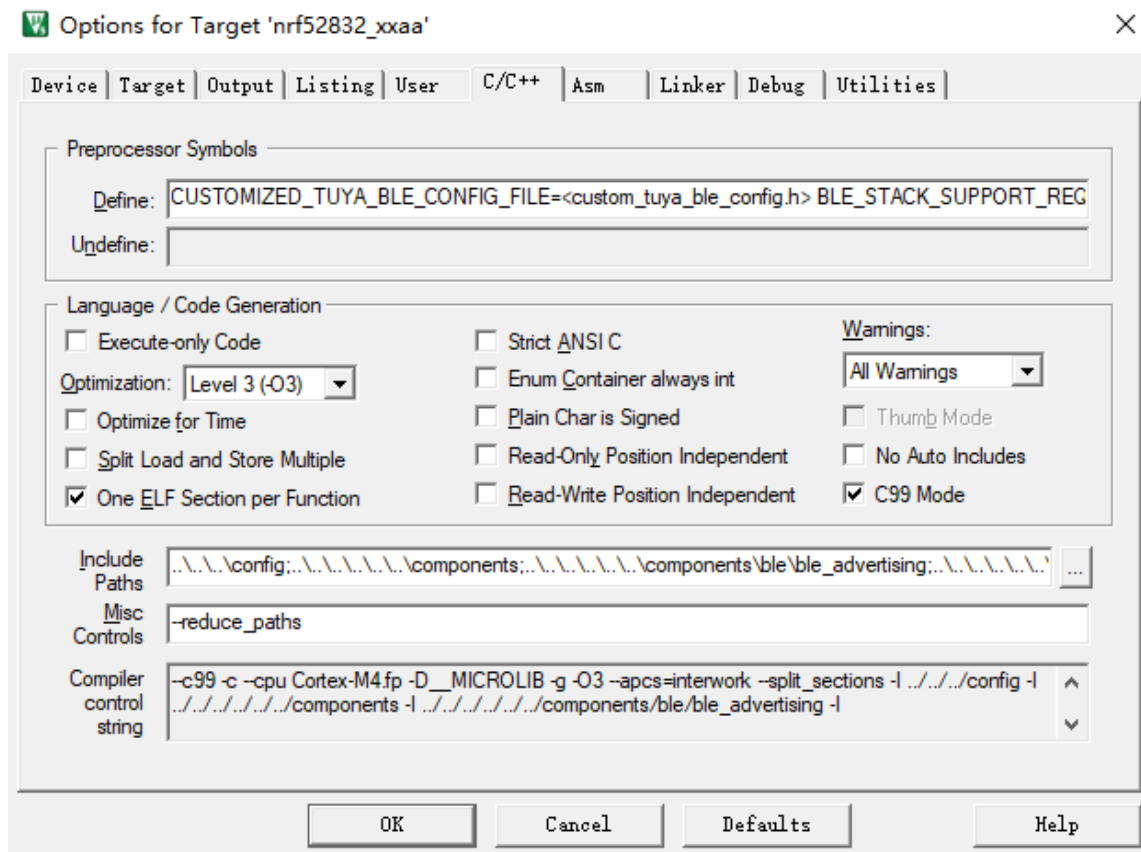
9. Your project must have the same directories as the following pictures. Note that select correct library files.



10. Create a `custom_tuya_ble_config.h` file, and save to the directories of your project, for example, `tuya_ble_app`.

You must configure the items in the `custom_tuya_ble_config.h` according to your actual demand and environment. The `tuya ble sdk` provides references for some chip platforms. The reference configuration files are stored in each platform directory under the `port` directory in the `tuya ble sdk` folder.

11. Assign the name of `custom_tuya_ble_config.h` to the `CUSTOMIZED TUYA_BLE_CONFIG_FILE` file, and add to the macro of your project.



12. Create port files, and the name can be `tuya_ble_port_nrf52832.h` and `tuya_ble_port_nrf52832.c`, or replace the `nrf52832` with other platforms.

In the port file, implement the interfaces listed in the `tuya_ble_port.h` file according to your configurations. Not all of the listed interfaces need to be implemented, for example, this porting tutorial does not involve OS, therefore, the

OS-specific APIs are not needed. This tutorial is configured to use the internal memory management module of tuya ble sdk, therefore the memory allocation and release APIs are not needed. Under the port file of the tuya ble sdk, every platform is provided with porting reference file with the platform as the file name.

13. After the port file is completed, define the port file in the `custom_tuya_ble_config.h`.

```
#ifndef CUSTOM TUYA_BLE_CONFIG_H_
#define CUSTOM TUYA_BLE_CONFIG_H_

#include "tuya_ble_type.h"
#include "tuya_ble_app_demo.h"

#define TUYA_BLE_APP_VERSION_STRING    TY_APP_VER_STR

#define TUYA_BLE_APP_BUILD_FIRMNAME_STRING  APP_BUILD_FIRMNAME

#define TUYA_BLE_PORT_PLATFORM_HEADER_FILE  "tuya_ble_port_nrf52832.h"

#define CUSTOMIZED TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE "custom_app_product_test.h"

#define CUSTOMIZED TUYA_BLE_APP_UART_COMMON_HEADER_FILE "custom_app_uart_common_handler.h"

#define TUYA_BLE_USE_OS 0
```

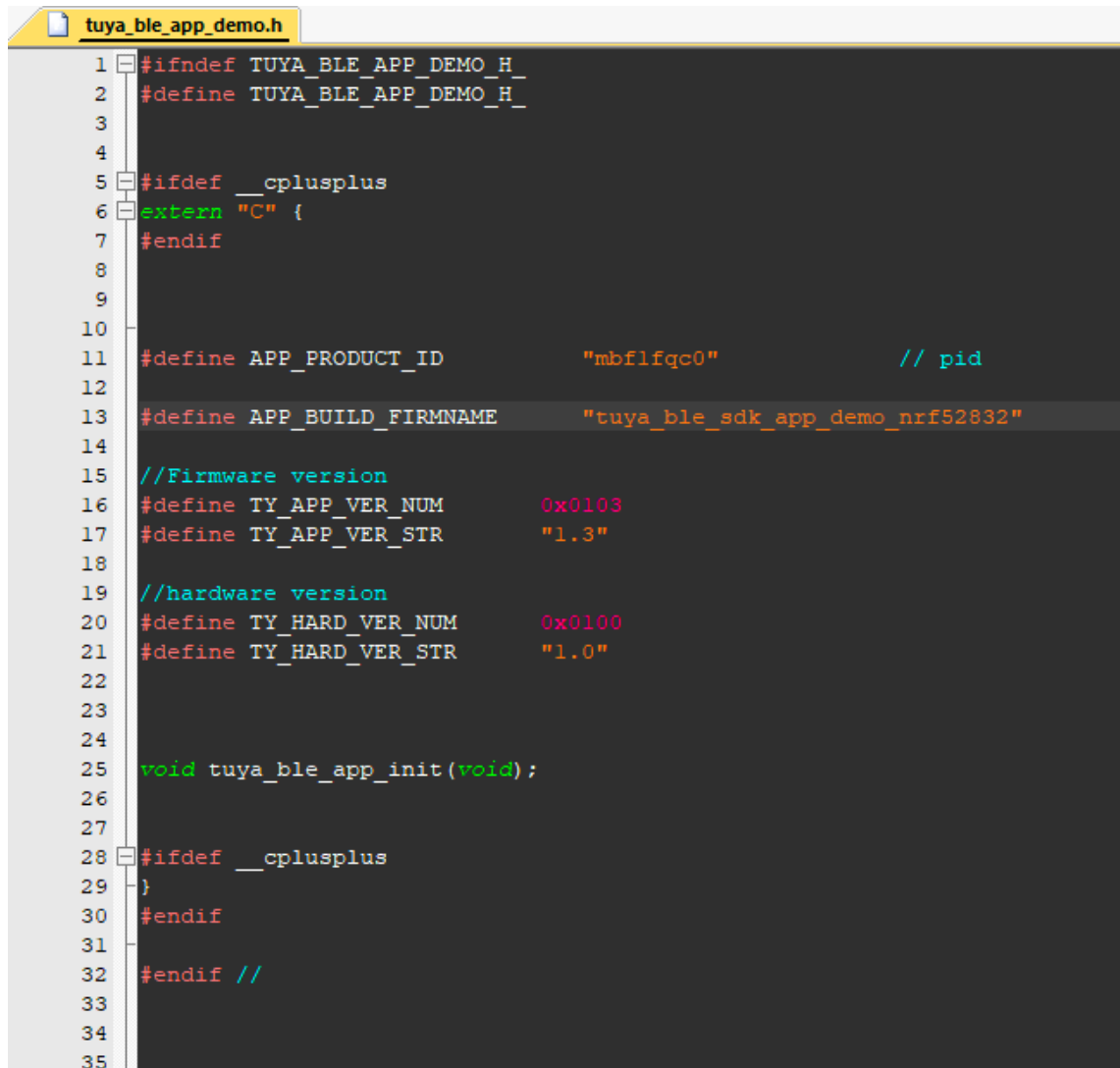
14. Compile. If the compiling fails, examine your code first.
15. In this tutorial, a specific file is created to handle the initialization of the tuya ble sdk, call back function registration, and callback message handling. As the following picture shows.

```

tuya_ble_app_demo.c
188     TUYA_APP_LOG_INFO("tuya sdk init succeed.");
189     TUYA_APP_LOG_INFO("app version : "TY_APP_VER_STR);
190     TUYA_APP_LOG_DEBUG("current free heap size = %d",xTuyaPortGetFreeHeapSize());
191     //tuya_ble_device_factory_reset();
192 }
193 else
194 {
195     TUYA_APP_LOG_INFO("tuya sdk init failed.");
196 }
197 }
198
199 extern void tuyu_ble_nv_init_custom(void);
200
201
202 static const char auth_key_test[] = "eJkymuEV...DPGnbOs27VQn8rN";
203 static const char device_id_test[] = "1c1161...189ba13";
204
205
206 void tuyu_ble_app_init(void)
207 {
208     tuyu_ble_nv_init_custom();
209     memset(&device_param,0,sizeof(tuya_ble_device_param_t));
210     device_param.device_id_len = 16;
211     memcpy(device_param.auth_key,(void *)auth_key_test,AUTH_KEY_LEN);
212     memcpy(device_param.device_id,(void *)device_id_test,DEVICE_ID_LEN);
213
214     device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
215     device_param.product_id_len = 8;
216     memcpy(device_param.product_id,APP_PRODUCT_ID,8);
217     device_param.firmware_version = TY_APP_VER_NUM;
218     device_param.hardware_version = TY_HARD_VER_NUM;
219
220     tuyu_ble_sdk_init_async(&device_param,tuya_ble_sdk_init_async_completed);
221
222

```

16. Register the product in the Tuya Developer Platform and copy the product ID to your project code. The macro name `APP_PRODUCT_ID`, `APP_BUILD_FIRMNAME`, `TY_APP_VER_NUM`, `TY_APP_VER_STR`, `TY_HARD_VER_NUM`, and `TY_HARD_VER_STR` cannot be changed. Replace the `xxxxxxxx` as the product ID. If you use the test and production tooling authorization, contact your Tuya account manager to create a project for you, and replace the `tuya_ble_sdk_app_demo_nrf52832` as the project name.



```
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          "mbflfq0"          // pid
12
13 #define APP_BUILD_FIRMNAME      "tuya_ble_sdk_app_demo_nrf52832"
14
15 //Firmware version
16 #define TY_APP_VER_NUM         0x0103
17 #define TY_APP_VER_STR         "1.3"
18
19 //hardware version
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 //
```

17. Call the `tuya_ble_app_init()`, `tuya_ble_main_tasks_exec()`, `tuya_ble_gatt_receive_data()`, `tuya_ble_common_uart_receive_data()`, `tuya_ble_disconnected_handler()`, and `tuya_ble_connected_handler()` respectively in your code, as the following picture shows.

```

main.c
552 /**@brief Function for handling app_uart events.
553 *
554 * @details This function will receive a single character from the app_uart module and append it to
555 * a string. The string will be sent over BLE when the last character received was a
556 * 'new line' '\n' (hex 0x0A) or if the string has reached the maximum data length.
557 */
558 /**@snippet [Handling the data received over UART] */
559 void uart_event_handle(app_uart_evt_t * p_event)
560 {
561     //static uint8_t data_array[BLE_NUS_MAX_DATA_LEN];
562     //static uint8_t index = 0;
563     uint8_t rx_char=0;
564     uint32_t err_code;
565
566     switch (p_event->evt_type)
567     {
568     case APP_UART_DATA_READY:
569         UNUSED_VARIABLE(app_uart_get(&rx_char));
570         //UNUSED_VARIABLE(app_uart_get(&data_array[index]));
571         // index++;
572         tuya_ble_common_uart_receive_data(&rx_char,1);
573         break;
574
575     case APP_UART_COMMUNICATION_ERROR:
576         // APP_ERROR_HANDLER(p_event->data.error_communication);
577         break;
578
579     case APP_UART_FIFO_ERROR:
580         //APP_ERROR_HANDLER(p_event->data.error_code);
581         break;
582
583     default:
584         break;
585     }
586 }

```

18. During the development and debug stage, contact your Tuya account manager to obtain the `auth_key_test` and `device_id_test`.
19. Compile your code, download the code to the development board, download Tuya Smart App, and scan to add devices.

7 OTA protocol

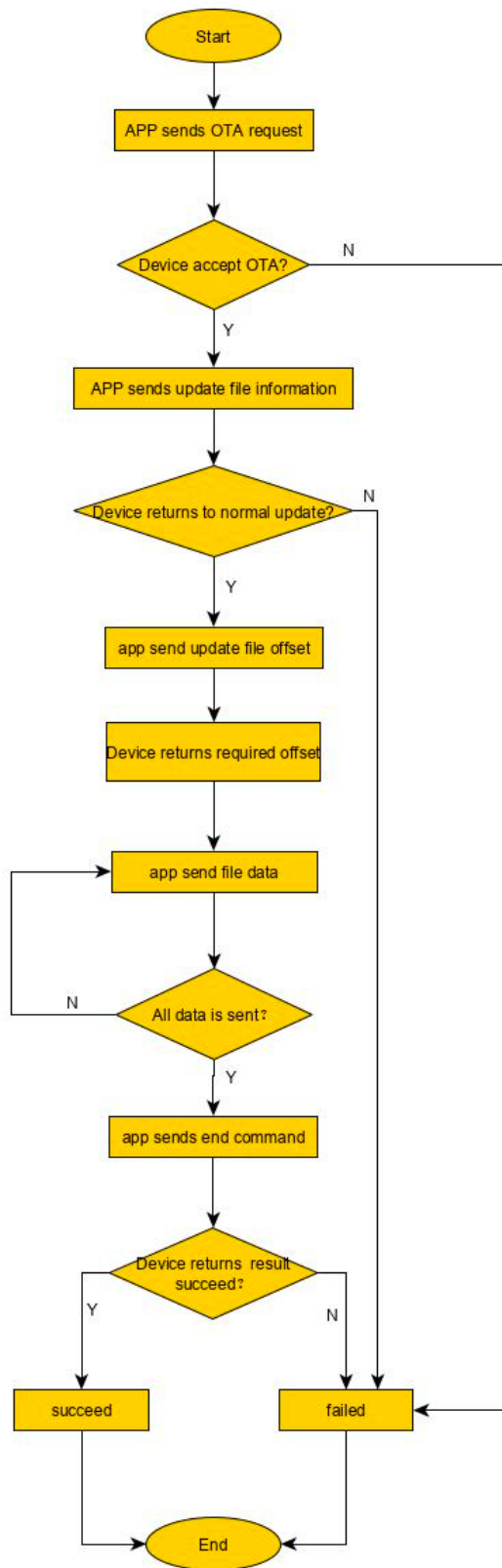
The firmware upgrade and the chip platform architecture are related, therefore the tuya ble sdk only provides a firmware upgrade interface, and your Application only needs to be implemented according to the OTA protocol described below through the OTA communication interface provided by tuya ble sdk.

The Application receives OTA data through the registered callback function (if the chip platform does not have OS) or registered queue (if the chip platform has an OS), the EVENT ID is `TUYA_BLE_CB_EVT_OTA_DATA`.

For the data format, see the upgrade protocol section. And the OTA response data is sent through the `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` function.

7.1 OTA upgrade process

The following picture illustrates the process of OTA upgrade.



7.2 OTA upgrade protocol

7.2.1 OTA data types

```
1  typedef enum
2
3  {
4
5      TUYA_BLE_OTA_REQ, // OTA Upgrade request command
6
7      TUYA_BLE_OTA_FILE_INFO, // OTA Upgrade file information command
8
9      TUYA_BLE_OTA_FILE_OFFSET_REQ, // OTA Upgrade file offset command
10
11     TUYA_BLE_OTA_DATA, // OTA Upgrade data command
12
13     TUYA_BLE_OTA_END, // End of OTA upgrade command
14
15     TUYA_BLE_OTA_UNKONWN,
16
17 }tuya_ble_ota_data_type_t;
18
19 typedef struct{
20
21     tuya_ble_ota_data_type_t type;
22
23     uint16_t data_len;
24
25     uint8_t *p_data;
26
27 }tuya_ble_ota_data_t; //The mobile app sends the data struct
                        //corresponding to the OTA upgrade EVENT (TUYA_BLE_CB_EVT_OTA_DATA)..
28
29 typedef struct{
30
31     tuya_ble_ota_data_type_t type;
32
33     uint16_t data_len;
34
35     uint8_t *p_data;
36
37 }tuya_ble_ota_response_t; //Data struct corresponding to OTA response
                        //data sending function tuya_ble_ota_response(tuya_ble_ota_response_t
                        // *p_data)
```


7.2.2 OTA upgrade request (TUYA_BLE_OTA_REQ)

From App to BLE devices

data_len=1	
Length:	1 byte
Data:	fixed 0

From BLE devices to App

data_len=9					
Length:	1 byte	1 byte	1 byte	4 byte	2 byte
Data:	Flag	OTA_Version	0	Version	Maximum packet length.

- **Flag:** 0x00 indicates upgrade confirmation, 0x01 indicates upgrade denial.
- **OTA_Version:** OTA Protocol version, for example 0x03 indicates the protocol version of 3.X.
- **Version:** the current firmware version number in the big-endian format. For example, 0x00 01 00 02 indicates the version number is V1.0.2.
- **Maximum packet length:** the maximum length of a single packet allowed by the device, unit: byte. The current version cannot exceed 512 bytes.

7.2.3 OTA upgrade file information (TUYA_BLE_OTA_FILE_INFO)

From App to BLE devices

data_len=37						
Length:	1 byte	8 byte	4 byte	16 byte	4 byte	4 byte
Data:	0	product id	file version	file MD5	file length	CRC32

- `product id`: the PID.
- `file version`: for example, `0x00010002` indicates the version is V1.0.2.

From BLE devices to App

data_len=26					
Length:	1 byte	1 byte	4 byte	4 byte	16 byte
Data:	0	STATE	Saved data length	Saved data CRC32	saved data MD5 (Not used currently)

- `STATE`:
 - `0x00`: upgrade success
 - `0x01`: PID different
 - `0x02`: the file version is lower than or equal to the current version
 - `0x03`: file size is out of range.
- `Other`: a reserved field.
- `File information has been saved`: to support the resuming of the breakpoint, the file information that are stored in the device will be returned. After receiving the information, the App first calculates the CRC32 of the corresponding length of the new file according to the length of the stored file returned by the device, and then compares it with the CRC32 returned by the device. If the both are consistent, then in the following file start transmission request, the start transmission offset is changed to the length value. Otherwise, the file start transmission offset is changed to 0, indicating that the transmission starts from the beginning.

7.2.4 OTA upgrade file offset (TUYA_BLE_OTA_FILE_OFFSET_REQ)

From App to BLE devices

data_len=5		
Length:	1 byte	4 byte
Data:	0	Offset

offset: upgrade file offset.

From BLE devices to App

data_len=5		
Length:	1 byte	4 byte
Data:	0	Offset

offset: the starting file offset required by the device. The offset address of the actual file transfer should be based on the device requirements, and the address required by the device will be less or equal to the offset given by the App.

7.2.5 OTA Upgrade data (TUYA_BLE_OTA_DATA)

From App to BLE devices

data_len=7+n					
Length:	1 byte	2 byte	2 byte	2 byte	n byte
Data:	0	Package number	Current package data length n	Current package data CRC16	Current package data

The packet number starts from 0, and the high byte is at the beginning of the packet.

From BLE devices to App

data_len=2		
Length:	1 byte	1 byte
Data:	0	STATE

STATE:

- 0x00: success
- 0x01: the package number is abnormal
- 0x02: the length is inconsistent
- 0x03: CRC check fails
- 0x04: others

7.2.6 OTA upgrade is over (TUYA_BLE_OTA_END)**From App to BLE devices**

data_len=1	
Length:	1 byte
Data:	0

From BLE devices to App

data_len=2		
Length:	1 byte	1 byte
Data:	0	STATE

STATE:

- 0x00: success
- 0x01: the total data length is wrong

- 0x02: the total CRC of data check fails
- 0x03: others

If you need to restart after verifying the success of the ble device OTA file, you can call the API `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` to respond to the App result at least no delay and restart after 2 seconds.

7.3 OTA upgrade APIs

The Application receives OTA data through the registered callback function (without RTOS environment) or registered queue (in the RTOS environment), the EVENT ID is `TUYA_BLE_CB_EVT_OTA_DATA`. And the OTA response data is sent through `tuya_ble_ota_response(tuya_ble_ota_response_t *p_data)` function.

As the following picture shows, the Application calls the custom OTA processing function.

```
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_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");
            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;
        case TUYA_BLE_CB_EVT_UNKNOWN_TUYA:
    }
```

An example of OTA processing function:

```
1 void tuya_ota_proc(uint16_t cmd,uint8_t*recv_data,uint32_t recv_len)
2 {
3     TUYA_BLE_LOG_DEBUG("ota cmd: 0x%04x , recv_len: %d",cmd,recv_len);
4     switch(cmd)
5     {
6     case TUYA_BLE_OTA_REQ:
7         tuya_ota_start_req(recv_data,recv_len);
8         break;
9     case TUYA_BLE_OTA_FILE_INFO:
10        tuya_ota_file_info_req(recv_data,recv_len);
11        break;
12        case TUYA_BLE_OTA_FILE_OFFSET_REQ:
13            tuya_ota_offset_req(recv_data,recv_len);
14            break;
15            case TUYA_BLE_OTA_DATA:
16                tuya_ota_data_req(recv_data,recv_len);
17                break;
18                case TUYA_BLE_OTA_END:
19                    tuya_ota_end_req(recv_data,recv_len);
20                    break;
21                default:
22                    break;
23            }
24
25 }
```

8 Reference of production testing APIs

Before BLE devices are connected to Tuya IoT development platform, they must be burned with license information (one set of licenses for one device), which is usually burned in factory production. You can also use Tuya production testing tools to burn license and test. Alternatively, you can purchase licenses in batches, and use custom protocol and interface to manage them.

If you use custom protocol, you must set the `auth_key` and `device_id` when you initialize the `tuya ble sdk`. As the following sample shows.

```
1  tuya_ble_device_param_t device_param ;
2
3  void tuya_ble_app_init(void)
4  {
5      memset(&device_param,0,sizeof(tuya_ble_device_param_t));
6      device_param.device_id_len = 16;
7      memcpy(device_param.auth_key,(void *)auth_key_test,AUTH_KEY_LEN);
8      memcpy(device_param.device_id,(void *)device_id_test,DEVICE_ID_LEN);
9      device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
10     device_param.product_id_len = 8;
11     memcpy(device_param.product_id,APP_PRODUCT_ID,8);
12     device_param.firmware_version = TY_APP_VER_NUM;
13     device_param.hardware_version = TY_HARD_VER_NUM;
14
15     tuya_ble_sdk_init(&device_param);
16     tuya_ble_callback_queue_register(tuya_cb_handler);
17
18     tuya_ota_init();
19
20     TUYA_APP_LOG_INFO("app version: "TY_APP_VER_STR);
21 }
```

If you use the authorization tool of Tuya production testing to burn the license and you use the `tuya ble sdk` to manage authorization, then the `auth_key` and `device_id` are not needed when you initialize the `tuya ble sdk`. As the following sample shows.

```
1  tuya_ble_device_param_t device_param ;
2
3  void tuya_ble_app_init(void)
4  {
5      memset(&device_param,0,sizeof(tuya_ble_device_param_t));
6      device_param.device_id_len = 0;
7      device_param.p_type = TUYA_BLE_PRODUCT_ID_TYPE_PID;
8      device_param.product_id_len = 8;
9      memcpy(device_param.product_id,APP_PRODUCT_ID,8);
10     device_param.firmware_version = TY_APP_VER_NUM;
11     device_param.hardware_version = TY_HARD_VER_NUM;
12
13     tuya_ble_sdk_init(&device_param);
14     tuya_ble_callback_queue_register(tuya_cb_handler);
15
16     tuya_ota_init();
17
18     TUYA_APP_LOG_INFO("app version: "TY_APP_VER_STR);
19 }
```

If you use the authorization tool of Tuya production testing to burn the license, make sure that the `TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT` is enabled by setting `#define TUYA_BLE_DEVICE_AUTH_SELF_MANAGEMENT 1`.

The production testing consists of the general authorization testing and the general device testing, and the general machine testing is subordinate to the general authorization testing. The general authorization testing includes authorization burning, GPIO testing and RSSI testing. The general device testing provides additional testing for tailored products, you can see the Bluetooth General Authorization Protocol of Production Testing or Bluetooth General Device Testing Protocol of Production Testing for more information.

The tuya ble sdk has achieved the protocol of general authorization testing, however, testing such as RSSI testing, GPIO testing and additional testing for tailored products provided by general device testing must be implemented according to the product configuration. The source file `tuya_ble_app_production_test.c` in the tuya ble sdk has prepared APIs for the preceding testings, and they are defined in the form of `__TUYA_BLE_WEAK` weak implement. Your Application can redefine those APIs in other source files, for example, creating a `custom_app_product_test.c` file, and referring to it in your custom configuration file. As the following pictures show.


```

custom_app_product_test.c
16 #include "custom_app_product_test.h"
17
18
19 tuya_ble_status_t tuya_ble_prod_beacon_scan_start(void)
20 {
21     //
22     return TUYA_BLE_SUCCESS;
23 }
24
25 tuya_ble_status_t tuya_ble_prod_beacon_scan_stop(void)
26 {
27     //
28     return TUYA_BLE_SUCCESS;
29 }
30
31 tuya_ble_status_t tuya_ble_prod_beacon_get_rssi_avg(int8_t *rssi)
32 {
33     //
34     *rssi = -30;
35     return TUYA_BLE_SUCCESS;
36 }
37
38 tuya_ble_status_t tuya_ble_prod_gpio_test(void)
39 {
40     //Add gpio test code here
41     return TUYA_BLE_SUCCESS;
42 }
43
44 void tuya_ble_custom_app_production_test_process(uint8_t channel, uint8_t *p_in_data, uint16_t in_len)
45 {
46     uint16_t cmd = 0;
47     uint8_t *data_buffer = NULL;
48     uint16_t data_len = ((p_in_data[4]<<8) + p_in_data[5]);
49
50     if((p_in_data[6] != 3) || (data_len<3))
51         return;
52
53     cmd = (p_in_data[7]<<8) + p_in_data[8];
54     data_len -= 3;
55     if(data_len>0)
56     {

```

```

custom_app_product_test.c  custom_app_product_test.h
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 channel,uint8_t *p_in_data,uint16_t in_len);
20
21
22 #ifdef __cplusplus
23 }
24 #endif
25
26 #endif //

```

```

custom_app_product_test.c  custom_app_product_test.h  custom_tuya_ble_config.h
25
26 #ifndef CUSTOM TUYA_BLE_CONFIG_H_
27 #define CUSTOM TUYA_BLE_CONFIG_H_
28
29 #include "tuya_ble_type.h"
30 #include "tuya_ble_app_demo.h"
31
32
33 #define TUYA_BLE_APP_VERSION_STRING    TY_APP_VER_STR
34
35
36 #define TUYA_BLE_APP_BUILD_FIRMNAME_STRING  APP_BUILD_FIRMNAME
37
38
39 #define TUYA_BLE_PORT_PLATFORM_HEADER_FILE  "tuya_ble_port_nrf52832.h"
40
41
42 #define CUSTOMIZED TUYA_BLE_APP_PRODUCT_TEST_HEADER_FILE "custom_app_product_test.h"
43
44
45 #define CUSTOMIZED TUYA_BLE_APP_UART_COMMON_HEADER_FILE "custom_app_uart_common_handler.h"
46
47
48 #define TUYA_BLE_USE_OS 0
49
50
51 #if TUYA_BLE_USE_OS
52
53 #define TUYA_BLE_SELF_BUILT_TASK 0
54
55 #if TUYA_BLE_SELF_BUILT_TASK
56
57 #define TUYA_BLE_TASK_PRIORITY 1
58
59 #define TUYA_BLE_TASK_STACK_SIZE 256 * 10  //!< Task stack size , application do not change this default value
60
61 #endif
62
63 #endif

```