WM_W800_Bluetooth system architecture and API description

V1.2

Beijing Lianshengde Microelectronics Co., Ltd. (winner micro)

Address: Room 1802, Yindu Building, No. 67, Fucheng Road, Haidian District, Beijing

Tel: +86-10-62161900

Company website: www.winnermicro.com

Document modification record

Version revision time	revision history	author review	
V0.1	2019/9/25 [C] Create document	Wangm	
V0.2	2020/7/7	1. Add the AT command to set the Bluetooth name 2. Change the sample code path	Pengxg
V0.3	2020/7/8 Unified font		Cuiyc
V0.4	2020/8/12 Add traditional Bluetooth function:	1. 2.6 Traditional Bluetooth Audio 2. 2.7 Traditional Bluetooth hands-free phone 3. 3.4.4 Legacy Bluetooth Audio 4. 3.4.5 Traditional Bluetooth hands-free phone	Pengxg
V0.5	2020/8/17 1, Add API usage example chapter		Pengxg
V0.6	2020/8/25 1, Add Bluetooth entry and exit test mode API	illustrate 2, Update and set the broadcast extension parameter API 3, Increase the scan mode parameter, specify the frequency point to scan scanning function 4, Supplementary scanning API sample code 5, Add non-connectable broadcast API parameters 6, Increase broadcast scan coexistence, in slave connection Coexistence of scanning and non-connectable advertising in connected state	Pengxg
V1.0	2020/12/02 1, add traditional Bluetooth Audio sink function description	Pengxg	

		Refer to the API/AT command description chapter 2, Increase the traditional Bluetooth HandFree function description Refer to the API/AT command description chapter 3, Add traditional Bluetooth SPP function description and API/AT command description chapter 4, Increase the traditional Bluetooth working mode setting API and AT command description		
V1.1	2021/03/05 1, add example server, client, UART Transparent transmission, multi-connection API description 2, Add example server, client, UART Transparent transmission, multi-connection AT command operation description 3. Increase broadcast scan coexistence , in Scanning and non-connection in the master connection state broadcast coexistence		Pengxg	
V1.2	2021/05/17 Modify the AT command: 1, set broadcast, scan response data 2, set broadcast parameters			

Table of contents

1 **Introduction**

1.1 Purpose of writing

This document is used to introduce the W800 Bluetooth software system, hardware system and its development Bluetooth application reference, and guide users to learn and understand w800 Bluetooth development.

1.2 Intended audience

Bluetooth application developers, Bluetooth protocol stack maintainers and test related personnel

1.3 Definition of terms

Ordinal term/abbreviation		Description/Definition
1	BT	BlueTooth
2	BECAME	Bluetooth Low Energy
3	HCI	Host Controller Interface
4 A2DP		Advanced Audio Distribution Profile
5 HFP		Hands-Free Profile

1.4 References

"W800 Chip Product Specifications"

ÿBluetooth Core spec4.0 and 4.2ÿ

"Bluetooth controller spec"

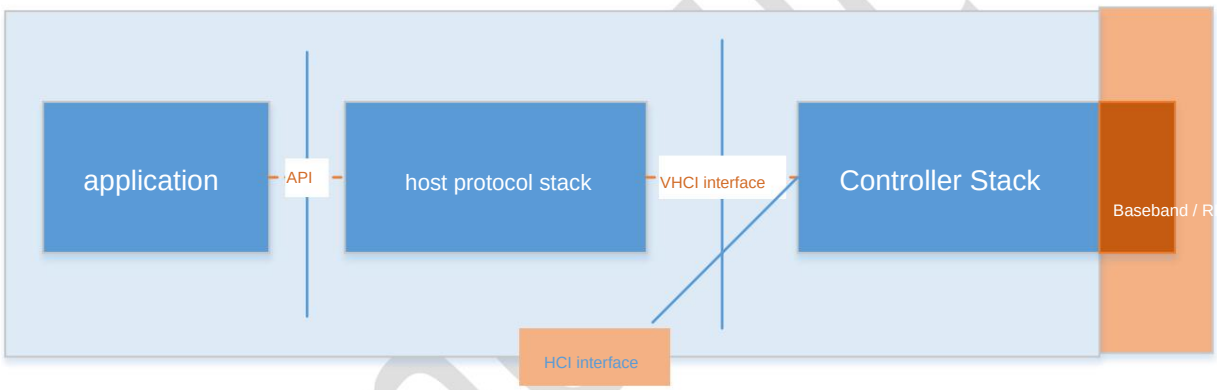
2 W800 Bluetooth system

2.1 Chip bluetooth design block diagram

2.2 W800 Bluetooth system block diagram

The W800 Bluetooth system can be divided into application program, host protocol stack, controller protocol stack, Bluetooth baseband, and radio frequency.

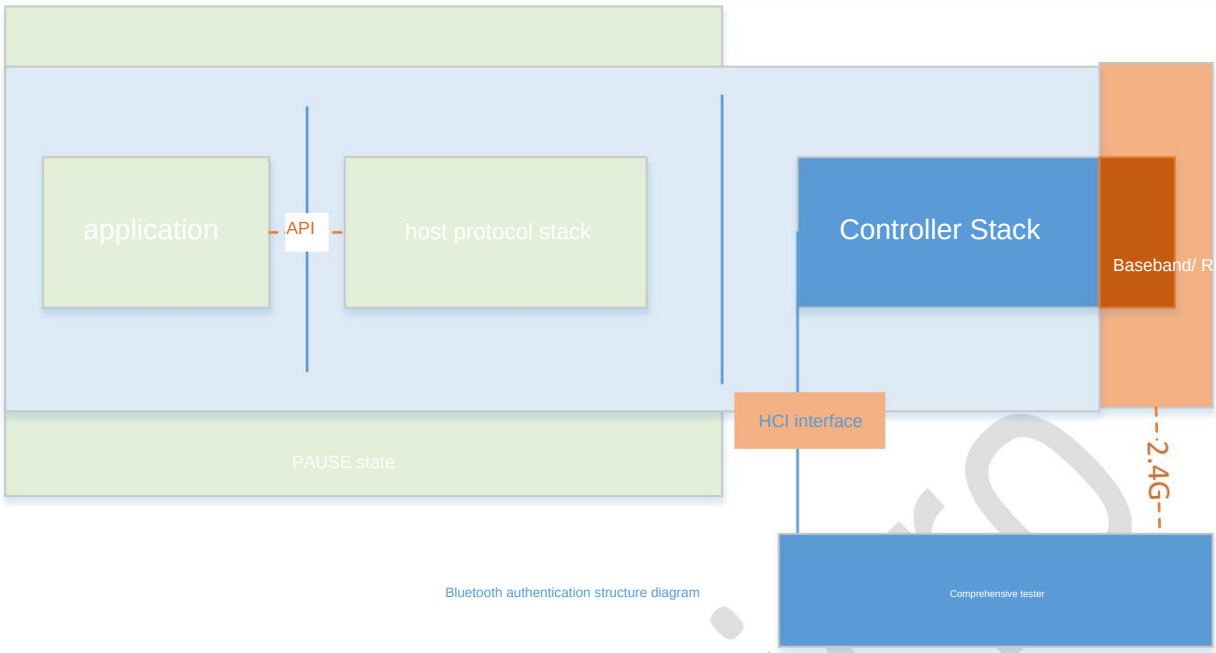
The radio frequency part of Bluetooth is shared with the WiFi system.



Bluetooth system structure diagram

For the certified HCI serial port operation instructions, refer to the traditional Bluetooth non-signaling test and BLE non-signaling test documents. Specific test method

As shown below:



Bluetooth authentication structure diagram

W800 provides a configurable UART port for responding to HCI commands. The comprehensive tester is directly controlled through the UART port controller. At this time, the host protocol stack is in the freeze state.

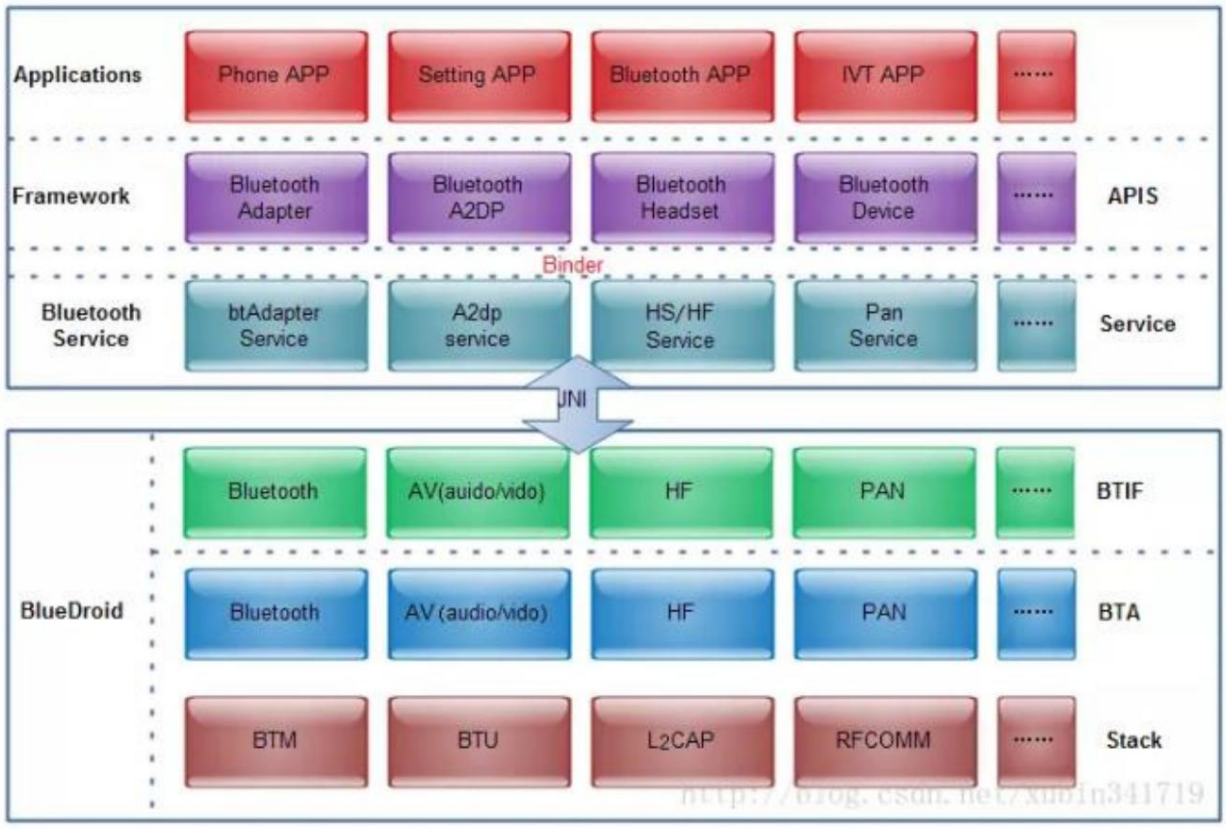
2.3 Introduction to Bluedroid

2.3.1 bluedroid

Bluedroid includes the traditional Bluetooth and Bluetooth low energy protocol stacks, and uses the standard HCI protocol to interact with the controller.

We ported bluedroid7.0, and replaced the internal task with our own microkernel mechanism.

2.3.2 Architecture diagram of bluedroid



After we transplanted, we kept the three layers of STACK, BTA and simplified BTIF. User-developed applications are directly based on the BTIF layer

carry out.

2.4 Description of each application layer protocol

Bluedroid is mainly divided into 3 parts: BTIF, BTA, Stack.

As a Bluetooth core service, the Bluetooth Stack module consists of Bluetooth Application Layer (BTA) and

Bluetooth Embedded System (abbreviated as BTE) consists of two parts.

BTE: the internal processing of bluedroid, which can be subdivided into BTA, BTU, BTM, HCI, etc.

2.4.1 BTIF (Bluetooth Profile Interface)

BTIF: The medium between the Bluetooth Application task (BTA) and the JNI layer (also known as the glue layer on the Internet).

Provide the interface of all profile function lines to the upper layer JNI. There is also Bluetooth Interface Instance in this layer, so

There are Profile operation interfaces registered in it (GAP, AV, DM, PAN, HF, HH, HL, Storage, Sockets). Client

The application operates the Profile through the Instance

2.4.2 BTA (Bluetooth Appication)

BTA: Bluetooth application layer. Refers to the implementation and processing of each profile in bluedroid. The request from the upper layer goes through the BTA Layer, the request is sent to the BTA layer for processing by means of message sending.

All BTA messages are sent to BTU_TASK and processed by bta_sys_event; if it is a Gatt related message, Handled by bta_gatt_hdl_event.

Stack: realizes the bottom layer operation of Bluetooth.

2.4.3 BTU ÿBluetooth Upper Layerÿ

BTU: Undertake BTA and HCI

2.4.4 BTMÿBluetooth Manager ÿ

BTM: Management layer in Bluedroid. Bluetooth pairing and link management

2.4.5 HCI

HCI: Read and write data to Bluetooth HW. Interface between host and BT controller.

2.4.6 GKI module

Kernel unified interface. This layer is an adaptation layer, adapted to OS-related processes and memory-related management, and can also be used for Pass messages between threads. The unified management of the process is mainly realized through the variable gki_cb. GKI module in Bluedroid Mainly used for inter-thread communication.

2.4.7 bluedroid protocol stack message passing and processing

Communication in the Bluetooth protocol stack is accomplished through message queues.

2.5 Introduction to BLE

BLE delivers short data packets as needed and then shuts down the link, one of the reasons for BLE's low power consumption. Compared to regular bluetooth In the traditional pairing method, BLE devices only connect when they need to send and receive information.

The BLE communication method is extremely tight. The device exposes services for sending and receiving data, which contain content called characteristics for

Define data that can be shared. Characteristics can contain descriptors that help define the data.

Most BLE APIs support searching for local devices and discovering services, characteristics and descriptors about the device.

2.5.1 ATTENTION

ATT is an optimized protocol designed specifically for BLE devices. ATT works by sending as few bytes of data as possible. Place

All attributes have a Universally Unique Identifier (UUID), which is a standard 128-bit string ID that uniquely identifies

don't message. The attributes transported by the ATT are formatted as characteristics and services.

- Characteristic: Contains a single data and 0 or more descriptors to describe the characteristic

  value.

- Descriptor (Descriptor): The descriptor specifies attributes that can describe characteristic values. A human-readable description can be noted as

  Specify units or measurements, or define acceptable ranges of values

- Service (Service): A service refers to a collection of characteristics. For example, a service called "Heart Rate

  Monitor", which may contain multiple Characteristics, which may contain a feature called "heart

  rate measurement"ÿ Characteristicÿ

2.5.2 GATT

A GATT profile is a general specification for sending and receiving short pieces of data (called attributes) over a Bluetooth low energy link. current

The BLE application profiles are all based on GATT. The SIG predefines the number of profiles for BLE devices. These

A profile is a specification that describes how a device is used.

2.5.3 GAP

Defines how the device discovers, establishes a connection, and implements binding.

2.5.4 SMÿSecurity Managerÿ

Responsible for security in BLE communication.

2.5.5 Central equipment and peripheral equipment

Central and peripheral;

GATT server and GATT client;

GAP is used for peripheral devices and central devices, each device can play multiple roles, and can only play one role at the same time colour.

2.6 Legacy Bluetooth Audio

A2DP defines the traditional Bluetooth audio transmission specification, describing how stereo audio is transmitted from the media output (source) to the input (sink).

A2DP defines two roles for audio devices: output and input.

• Output (SRC) - The device acts as an output device when it streams digitized audio to the piconet's output.

• Input (SNK) - A device acts as an input device when it inputs a digitized audio stream from an SRC in the same piconet.

A2DP defines protocols and procedures for mono or stereo distribution of high-quality audio content over ACL channels.

Baseband, LMP, L2CAP and SDP are Bluetooth protocols defined in the Bluetooth Core Specification. AVDTP includes a communication A signaling entity for stream parameters and a transport entity for handling streams.

AVRC/AVRC CTRL defines the audio and video control transmission protocol, describes the output and output audio and video playback control specification.

2.7 Traditional Bluetooth hands-free phone

HFP defines the traditional Bluetooth hands-free phone function, describes how the hands-free device can use the gateway device to make and receive calls. talk.

HFP defines two roles of audio gateway (AG) and hands-free component (HF):

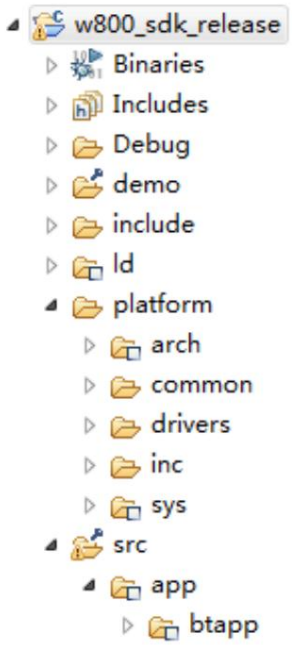
• Audio Gateway (AG) – This device is an input/output gateway for audio (especially mobile phones).

• Hands Free Kit (HF) – This device acts as a remote audio input/output mechanism for the audio gateway and provides several remote

Function.

2.8 Source Framework Description

2.8.1 Bluetooth system software code location



The Btapp directory is the bluetooth sample code, users can refer to or carry out secondary development based on this code.

List of application files:

No application module		illustrate
1	wm_bt_app.c	Host protocol stack main program entry
2	wm_ble_dm.c	Device Management Module
3	wm_ble_server.c	BLE server application management module, responsible for each prof registration and Event distribution processing.
4	wm_ble_server_wifi_prof.c	BLE distribution network service communication module
5	wm_ble_server_wifi_app.c	BLE distribution network application protocol processing module
6	wm_ble_server_demo_prof.c	BLE demo prof example, combined with AT command to give specific application Use example.

7	wm_ble_client.c	BLE client application management module, responsible for application registration and event distribution of files
8	wm_ble_client_huawei.c	BLE client sample application, connect to Huawei mobile phone, read A certain characteristic data
9	Wm_ble_client_demo.c	BLE client demo example, combined with AT commands to give the client end specific application example
10	wm_bt_audio_sink.c	Application example of traditional bluetooth audio sink demo
11	wm_bt_hfp_hsp.c	Example of a traditional Bluetooth hands-free phone application
12	wm_ble_client_api_demo.c	implements api to create demo server function
13	wm_ble_server_api_demo.c	implements api to create demo client function
14	wm_bt_spp_server.c	Realize the function of SPP server example
15	wm_bt_spp_client.c	Realize the function of SPP client example
16	wm_ble_uart_if.c	Example of implementing BLE-based UART transparent transmission

The header files involved are as follows

ordinal	header file name	describe
1	wm_bt.h	Bluetooth system host and controller api definition
2	wm_ble.h	Bluetooth system device management api definition
3	wm_ble_gatt.h	Bluetooth system GATT api definition
4	wm_bt_def.h	Bluetooth system data structure definition
5	Wm_bt_av.h	API definition of traditional Bluetooth Audio sink and source

6	Wm_bt_hf_client.h	Traditional Bluetooth hands-free phone client-side API definition
7	Wm_bt_spp.h	Traditional bluetooth SPP api definition

3 API Description

3.1 Bluetooth system API

No API name		describe
1	tls_bt_status_t tls_bt_enable(tls_bt_host_callback_t *scb, tls_bt_hci_if_t *uart, tls_bt_log_level_t log_level)	Run the Bluetooth system, this function will enable the host protocol and Controller protocol stack.
2	tls_bt_status_t tls_bt_disable(void)	To stop the Bluetooth system, the modified function will cancel the host protocol stack and Controller protocol stack.

3	Tls_bt_status_t
	tls_bt_host_cleanup()
		Clean up the host system, such as releasing resources and logging off tasks.
		Note: This function must wait for tls_bt_disable to be called,
		And wait for bt_adapter_state to change to OFF,
		can be called.
		tls_bt_disable();
		do
		{
		hal_system_sleep(100);

		}while(
		adapter_state != WM_BT_STATE_OFF);
		tls_bt_host_cleanup();

3.2 Host API

The host-side API describes functions such as host protocol stack startup and logout

No API name		describe
1	tls_bt_status_t tls_bt_host_enable(tls_bt_host_callback_t *p_callback, tls_bt_log_level_t log_level)	Initialize the host-side protocol stack and allocate memory and create tasks
2	tls_bt_status_t tls_bt_host_disable()	Unregister the host protocol stack
3	tls_bt_status_t tls_bt_pin_reply(const tls_bt_addr_t *bd_addr, uint8_t accept, uint8_t pin_len, tls_bt_pin_code_t *pin_code)	Reply to the pin code for BLE pairing
4	tls_bt_status_t tls_bt_ssp_reply(const tls_bt_addr_t *bd_addr, tls_bt_ssp_variant_t variant uint8_t accept,	reply pairing response

	uint32_t passkey)	
5	tls_bt_status_t tls_bt_set_adapter_property(const tls_bt_property_t *property)	Set the adapter attribute value
6	tls_bt_status_t tls_bt_get_adapter_property(tls_bt_property_type_t type)	Read the adapter attribute value
7	tls_bt_status_t tls_bt_start_discovery()	Traditional Bluetooth Scanning
8	tls_bt_status_t tls_bt_cancel_discovery()	stop scanning
9	tls_bt_status_t tls_bt_create_bond(const tls_bt_addr_t *bd_addr, int transport)	Initiate pairing
10	tls_bt_status_t tls_bt_cancel_bond(const tls_bt_addr_t *bd_addr)	Unpair operation
11	tls_bt_status_t tls_bt_remove_bond(const tls_bt_addr_t *bd_addr)	Delete pairing information

3.3 Controller API

No	API name	describe
1	tls_bt_status_t tls_bt_ctrl_enable(tls_bt_hci_if_t *p_hci_if, tls_bt_log_level_t log_level)	Initialize the controller-side protocol stack and allocate memory and create tasks
2	tls_bt_status_t tls_bt_ctrl_disable(void);	Unregister the controller stack
3	tls_bt_status_t tls_ble_set_tx_power(tls_ble_power_type_t power_type, int8_t power_level);	Set BLE transmit power index
4	int8_t tls_ble_get_tx_power(uint8_t power_type);	Read the transmit power index of the specified work type
5	tls_bt_status_t tls_bredr_set_tx_power(int8_t min_power_level, int8_t max_power_level);	Set the power index range when traditional bluetooth works value
6	tls_bt_status_t tls_bredr_get_tx_power(int8_t*min_power_level, int8_t* max_power_level);	An index to read the transmit power for legacy Bluetooth operation quoted range value
7	tls_bt_status_t tls_bredr_sco_datapath_set(	Set the output path of the traditional bluetooth sco link

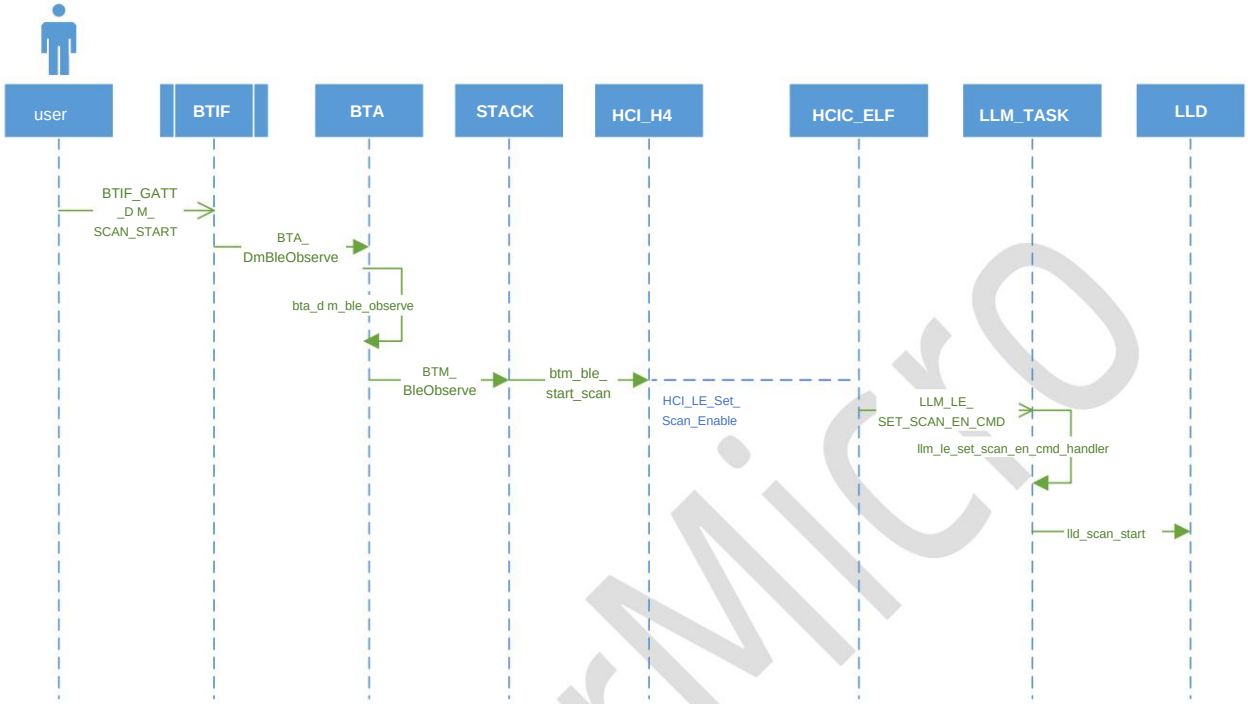
	wm_sco_data_path_t data_path);	
8	tls_bt_ctrl_status_t tls_bt_controller_get_status(void);	Read the current state of the controller,
9	bool wm_bt_vuart_host_check_send_available(void);	Used to determine whether the host can send instructions to the controller make
10	tls_bt_status_t tls_bt_vuart_host_send_packet (uint8_t *data, uint16_t len);	The host protocol stack sends data interface to the controller
11	tls_bt_status_t tls_bt_ctrl_if_register (const tls_bt_host_if_t *p_host_if);	Register the data sending interface of the controller, that is, the host Protocol stack receiving data interface
12	tls_bt_status_t tls_bt_ctrl_sleep (bool enable);	Whether to run the controller to enter on idle sleep mode
13	bool tls_bt_ctrl_is_sleep (void);	Reads whether the controller is in sleep mode
14	tls_bt_status_t tls_bt_ctrl_wakeup(void)	exit sleep mode
15	tls_bt_status_t enable_bt_test_mode(tls_bt_hci_if_t *p_hci_if)	Enter Bluetooth test mode
16	tls_bt_status_t exit_bt_test_mode()	Exit Bluetooth test mode

3.4 Application layer protocol API

3.4.1 Device Management

The device management layer is responsible for the general settings of the controller, such as broadcasting, scanning, device name modification and other functions. The following figure shows the scanning instructions

order invocation relationship.

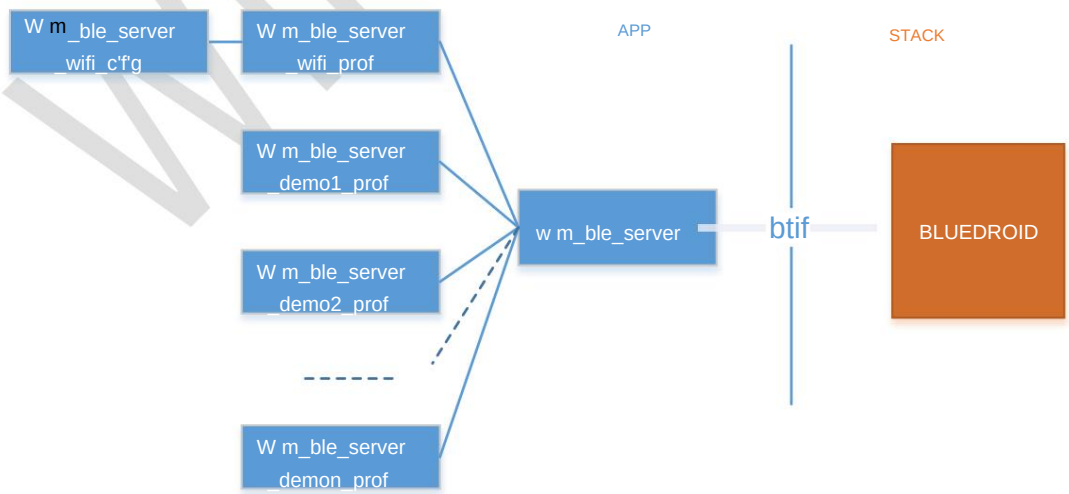


3.4.1.1 Device management layer API description

No API	name	describe
1.	tls_bt_status_t tls_ble_adv(uint8_t adv_type);	1: Enable discoverable connectable broadcast 2: Enable non-connectable broadcast, if configured If scan resp is set, it is scannable broadcast 0: turn off broadcasting
2.	tls_bt_status_t tls_ble_set_adv_data(tls_ble_dm_adv_data_t *data)	Set broadcast content
3.	tls_bt_status_t tls_ble_set_adv_param	Set broadcast parameters

	(tls_ble_dm_adv_param_t *param)	
4.	tls_bt_status_t tls_ble_set_adv_ext_param(tls_ble_dm_adv_ext_param_t *param);	Set broadcast extension parameters, if not Pretty sure how to populate the parameter body, strongly It is recommended to use 3.
5.	tls_bt_status_t tls_ble_set_scan_param(int window, int interval,uint8_t scan_mode);	Set scan parameters
6.	tls_bt_status_t tls_ble_scan(bool start);	start/stop scanning
6	tls_bt_status_t tls_dm_evt_triger (you hand, tls_ble_dm_triger_callback_t *p_callback)	Register for asynchronous processing events

3.4.2 BLE server



BLE server application relationship diagram

The BLE server assumes the role of slave, and the wm_ble_server module provides an interface description for user program development, based on

With this module, users can develop their own applications, such as wm_ble_server_demo1_prof, etc. which based on

The wifi distribution network of BLE is a specific application. In the figure above, the wm_ble_server_wifi_app module is used to process

Body distribution network protocol processing, wm_ble_server_wifi_prof undertakes the logic of service/character/descriptor

deal with.

3.4.2.1 BLE server API description

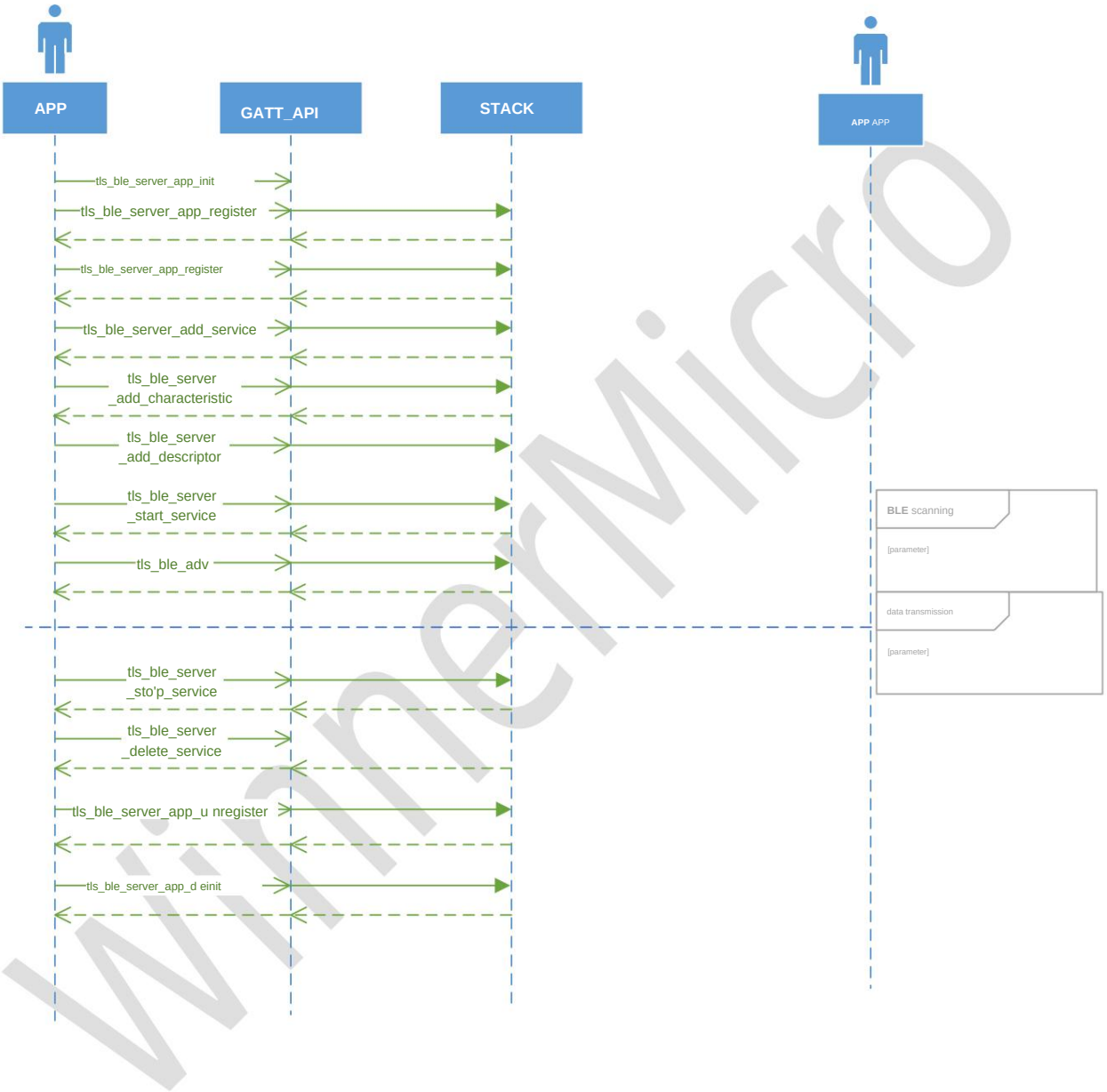
No API name		describe
1	tls_bt_status_t tls_ble_server_app_init (tls_ble_callback_t *p_callback)	Register the event response function of this layer with the btif layer. This function is in the main Called after the machine protocol stack starts.
2	tls_bt_status_t tls_ble_server_app_deinit();	Unregister the response function to the btif layer
3	tls_bt_status_t tls_ble_server_app_register (tls_bt_uuid_t *uuid)	Register an app server with a specified UUID to the btif layer
4 tls_bt_status_t	tls_ble_server_app_unregister(uint8_t server_if);	Unregister the app server registered in 3
5	tls_bt_status_t tls _ble_server_add_service(	Add service to registered server

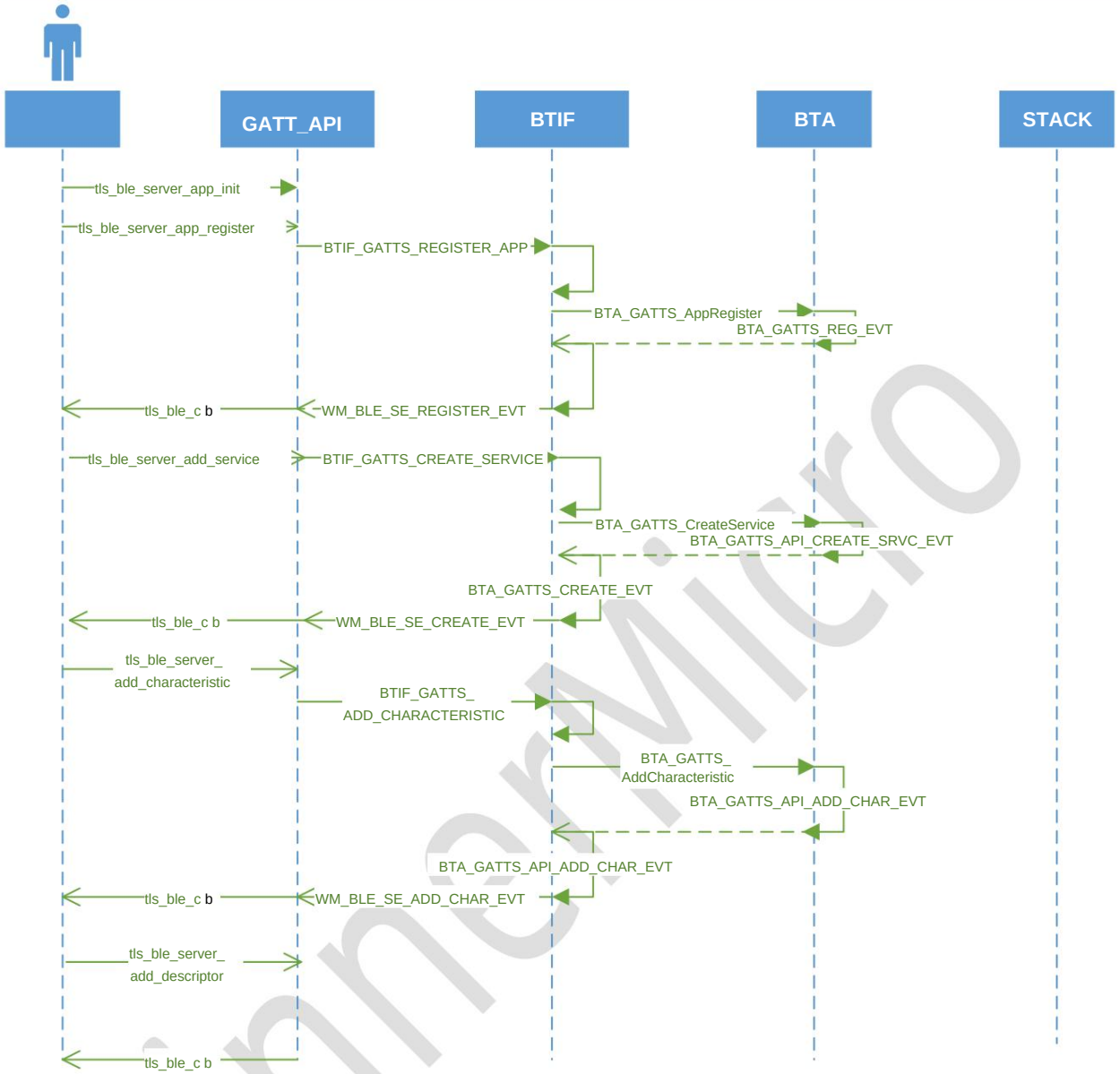
	int server_if, int inst_id, int primay, uint16_t uuid, int num_handles)	
6	tls_bt_status_t tls _ble_server_add_characteristic(int server_if, int service_handle, uint16_t handle, int properties, int permission)	Add a characteristic value to a specified service
7	tls_bt_status_t tls _ble_server_add_descriptor(int server_if, int service_handle, uint16_t uuid, int permissions)	Add a description of the characteristic value to a specified service
8	tls_bt_status_t tls _ble_server_start_service(int server_if, int service_handle, int transport)	Run the added service
9	tls_bt_status_t tls _ble_server_stop_service (int server_if, int service_handle)	Out of service

10	tls_bt_status_t tls _ble_server_delete_service (int server_if, int service_handle)	delete added service
12	tls_bt_status_t tls_ble_server_connect(int server_if, const bt_bdaddr_t *bd_addr, uint8_t is_direct, int transport)	Connect client operation [reserved]
13	tls_bt_status_t tls_ble_server_disconnect(int server_if, const bt_bdaddr_t *bd_addr, int conn_id)	Disconnect from the client
14	tls_bt_status_t tls _ble_server_send_indication(int server_if, int attribute_handle, int conn_id, int len, int confirm, char *p_value)	Send Indication, Notification message
15	tls_bt_status_t tls_ble_server_send_response(int conn_id, int trans_id, int offset, int attr_handle, int auth_req, uint8_t *value,	Send read/write response

	int length)	

3.4.2.2 BLE server creation/logout process description

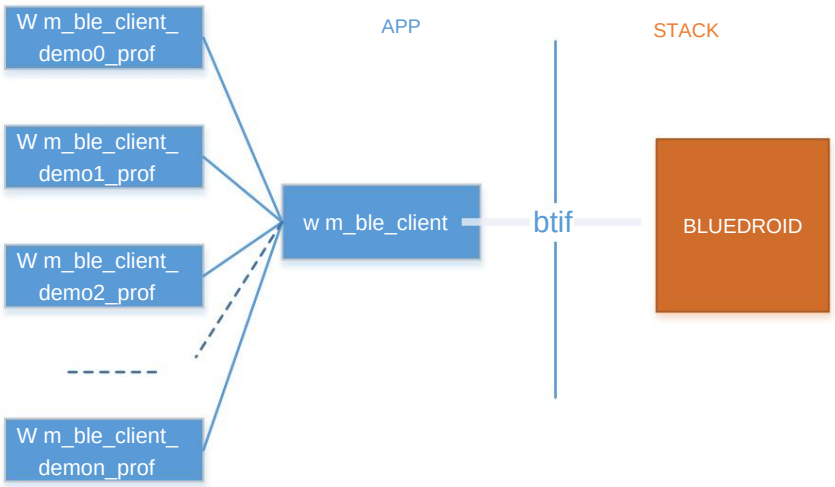




3.4.3 BLE client

The BLE client assumes the role of master, that is, actively initiates applications such as scanning, connection, and communication. wm_ble_client module

Provides an interface description for user program development, based on this module, users can develop their own application programs.



BLE client application structure diagram

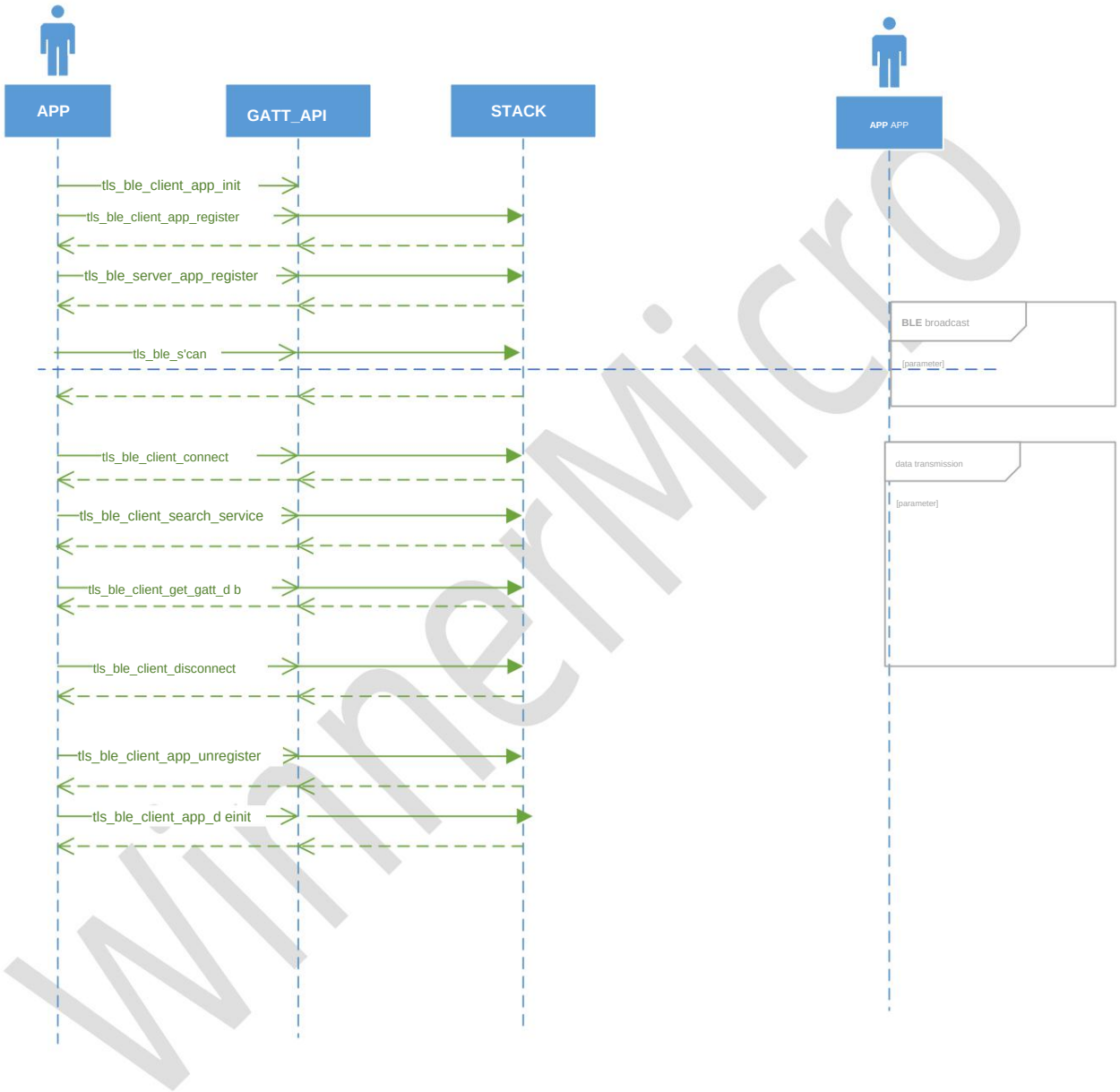
3.4.3.1 BLE client API description

No API name		describe
1.	tls_bt_status_t tls_ble_client_app_init(tls_ble_callback_t *p_callback)	Register the message on the client side with the btif layer message response function
2. tls_bt_status_t tls_ble_client_app_deinit(void)		Response function for unregistration
3	tls_bt_status_t tls_ble_client_app_register (tls_bt_uuid_t *uuid)	Register the specified client with the btif layer
4	tls_bt_status_t tls_ble_client_unregister_client(int client_if)	Log off the specified client
5	tls_bt_status_t tls_ble_client_connect (int client_if, const bt_bdaddr_t *bd_addr,	Connect to a specified server

	uint8_t is_direct, int transport)	
6	tls_bt_status_t tls_ble_client_disconnect(int client_if, const bt_bdaddr_t *bd_addr, int conn_id)	Disconnect from the server
7	tls_bt_status_t tls_ble_client_listen (int client_if, uint8_t start)	Listen for connections from the server
8	tls_bt_status_t tls_ble_client_refresh (int client_if, const bt_bdaddr_t *bd_addr)	Refresh the properties of the specified server list of values
9	tls_bt_status_t tls_ble_client_search_service(int conn_id, uint16_t filter_uuid)	Scan the list of server-side attribute values
10	tls_bt_status_t tls_ble_client_write_characteristic(int conn_id, uint16_t handle, int write_type, int len, int auth_req, char *p_value)	Write to a specified eigenvalue operate
11	tls_bt_status_t tls_ble_client_read_characteristic(int conn_id, uint16_t handle,int auth_req)	Read a specified characteristic value
12	tls_bt_status_t	Read a specified characteristic value description

	tls_ble_client_read_descriptor (int conn_id, uint16_t handle, int auth_req)	
13	tls_bt_status_t tls_ble_client_write_descriptor (int conn_id, uint16_t handle, int write_type, int len, int auth_req, char *p_value)	Describe the execution to a specified eigenvalue row write operation
14	tls_bt_status_t tls_ble_client_execute_write (int conn_id, int execute)	This instruction executes the prepare write operation do
15	tls_bt_status_t tls_ble_client_register_for_notification (int client_if, const bt_bdaddr_t *bd_addr, uint16_t handle)	Register the message ringing of a specified handle response function
16	tls_bt_status_t tls_ble_client_deregister_for_notification (int client_if, const bt_bdaddr_t *bd_addr, uint16_t handle)	Unregister the specified message response function
17	tls_bt_status_t tls_ble_client_configure_mtu(int conn_id, int mtu)	Set the maximum transfer order for a connection meta value
18	tls_bt_status_t tls_ble_client_get_gatt_db (int conn_id)	Read the attribute value of the specified connection id the list

3.4.3.2 BLE client creation/logout process description



3.4.4 Traditional Bluetooth Audio

The traditional Bluetooth audio API provides audio transmission and playback control interfaces, and the currently released version only supports the sink function.

No API	name	describe
1.	tls_bt_status_t	Register Audiosink message with btif layer
	tls_bt_av_sink_init (	Information response function, initialize sink function

	tls_bt_a2dp_sink_callback_t callback)	can
2. tls_bt_status_t tls_bt_av_sink_connect_src (	tls_bt_addr_t *bd_addr)	Create a connection with the source side
3	tls_bt_status_t tls_bt_av_sink_disconnect(tls_bt_addr_t *bd_addr)	Disconnect the same source side connection
4	tls_bt_status_t tls_bt_av_sink_deinit(void);	Unregister the Audio sink function
5	tls_bt_status_t tls_bt_av_src_init(tls_bt_a2dp_src_callback_t callback)	Register Audiosink message with btif layer Message response function, initialize source Function (not yet available)
6	tls_bt_status_t tls_bt_av_src_connect_sink(tls_bt_addr_t *bd_addr)	Create a connection with the sink side (not yet open)
7	tls_bt_status_t tls_bt_av_src_disconnect(tls_bt_addr_t *bd_addr)	Disconnect the connection with the sink side (not yet open)
8	tls_bt_status_t tls_bt_av_src_deinit(void)	Logout of souce function (not available yet)
9	tls_bt_status_t tls_btrc_init(tls_btrc_callback_t callback) initialize AVRC function	
10 tls_bt_status_t tls_btrc_get_play_status_rsp(	tls_btrc_play_status_t play_status, uint32_t song_len, uint32_t song_pos)	Send the call to GetPlayStatus should, specify the playback state of the track, Track time and elapsed playing time information
11 tls_bt_status_t	tls_btrc_get_element_attr_rsp(uint8_t num_attr,	Returns the element information of the current track

	tls_btrc_element_attr_val_t *p_attrs)	
12	tls_bt_status_t tls_btrc_register_notification_rsp(tls_btrc_event_id_t event_id, tls_btrc_notification_type_ttype, tls_btrc_register_notification_t *p_paramÿ	To register for an event of interest, after registration Can receive specific event notifications
13	tls_bt_status_t tls_btrc_set_volume(uint8_t volume) Set the playback volume of the other party, pay attention to the	The volume is an absolute value [0~127]
14	tls_bt_status_t tls_btrc_deinit(void)	Unregister the AVRC function
15	tls_bt_status_t tls_btrc_ctrl_init(tls_btrc_ctrl_callback_t callback)	Register AVRC Ctrl function
16	tls_bt_status_t tls_btrc_ctrl_send_passthrough_cmd(tls_bt_addr_t *bd_addr, uint8_t key_code, uint8_t key_state);	Send specific control to the source side instruction
17	tls_bt_status_t tls_btrc_ctrl_change_player_app_setting(tls_bt_addr_t *bd_addr, uint8_t num_attrib, uint8_t *attrib_ids, uint8_t *attrib_vals)	Change the configuration parameters of the player
18	tls_bt_status_t tls_btrc_ctrl_set_volume_rsp(tls_bt_addr_t *bd_addr, uint8_t abs_vol, uint8_t label)	Send the beep that sets the absolute volume command answer
19	tls_bt_status_t	After receiving the volume change notification, send

	tls_btrc_ctrl_volume_change_notification_rsp(tls_bt_addr_t *bd_addr, tls_btrc_notification_type_t rsp_type, uint8_t abs_vol, uint8_t label)	send response
20 tls_bt_status_t tls_btrc_ctrl_deinit(void);		Unregister avrc ctrl function

3.4.5 Traditional Bluetooth hands-free phone

The traditional Bluetooth hands-free phone API provides a hands-free phone client-side functional interface.

No API name		describe
1.	tls_bt_status_t tls_bt_hf_client_init(tls_bthf_client_callback_t callback)	Register and enable the hfp client function
2. tls_bt_status_t	tls_bt_hf_client_connect(tls_bt_addr_t *bd_addr)	Create the same audio gateway side (Audio Gateway) link connection
3	tls_bt_status_t tls_bt_hf_client_disconnect(tls_bt_addr_t *bd_addr)	Disconnect the link with the audio gateway side
4	tls_bt_status_t tls_bt_hf_client_connect_audio(tls_bt_addr_t *bd_addr)	Create the same audio gateway side (Audio Gateway) audio connection
5	tls_bt_status_t tls_bt_hf_client_disconnect_audio(tls_bt_addr_t *bd_addr)	Disconnect the same audio gateway side (Audio Gateway) audio connection
6	tls_bt_status_t tls_bt_hf_client_start_voice_recognition(void)	start speech recognition
7	tls_bt_status_t tls_bt_hf_client_stop_voice_recognition(void)	stop speech recognition

8	tls_bt_status_t tls_bt_hf_client_volume_control(tls_bthf_client_volume_type_t type, int volume)	Control MIC or speaker sound quantity
9	tls_bt_status_t tls_bt_hf_client_dial(const char *number)	dial
10	tls_bt_status_t tls_bt_hf_client_dial_memory(int location)	Dial the number at the specified location
11	tls_bt_status_t tls_bt_hf_client_handle_call_action(tls_bthf_client_call_action_t action, int idx)	Handle specific in-call responses
12	tls_bt_status_t tls_bt_hf_client_query_current_calls(void)	Query call list
13	tls_bt_status_t tls_bt_hf_client_query_current_operator_name(void)	Query the current operator name
14	tls_bt_status_t tls_bt_hf_client_retrieve_subscriber_info(void)	Query subscription number information
15	tls_bt_status_t tls_bt_hf_client_send_dtmf(char code)	send the dtmf value of the number key
16	tls_bt_status_t tls_bt_hf_client_request_last_voice_tag_number(void)	The number recognized by the request audio gateway Number
17	tls_bt_status_t tls_bt_hf_client_send_at_cmd(int cmd, int val1, int val2, const char *arg)	Send special to AudioGateway specified AT command
18	tls_bt_status_t tls_bt_hf_client_send_audio(tls_bt_addr_t *bd_addr, uint8_t *p_data, uint16_t length)	Send voice to AutoGateway Data (specific interface implementation depends on audio frequency module implementation)

19	tls_bt_status_t tls_bt_hf_client_deinit(void)	Log out of the speakerphone client function

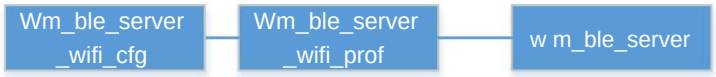
3.4.6 SPP

SPP is based on RFcomm's serial port transparent transmission service API.

No API	name	describe
1.	tls_bt_status_t tls_bt_spp_init(tls_bt_spp_callback_t callback)	Register spp callback interface function
2.	tls_bt_status_t tls_bt_spp_deinit()	Unregister the SPP callback interface
3	tls_bt_status_t tls_bt_spp_enable(void)	Enable SPP function
4	tls_bt_status_t tls_bt_spp_disable(void)	Stop SPP function
5	tls_bt_spp_start_discovery(tls_bt_addr_t * bd_addr, tls_bt_uuid_t *uuid)	Query based on MAC address and uuid traditional equipment
6	tls_bt_status_t tls_bt_spp_connect(wm_spp_sec_t sec_mask, tls_spp_role_t role, uint8_t remote_scn, tls_bt_addr_t * bd_addr)	Create SPP connection, by cient side initiated.
7	tls_bt_status_t tls_bt_spp_disconnect(uint32_t handle)	disconnect the connection with the Server side
8	tls_bt_status_t tls_bt_spp_start_server(wm_spp_sec_t sec_mask,tls_spp_role_t role,uint8_t local_scn,const char * name)	Start the SPP server

9	tls_bt_status_t tls_bt_spp_write(uint32_t handle, uint8_t *p_data, uint16_t length)	data sending function

3.5 Bluetooth assisted WiFi distribution network API



BLE wifi fast distribution network application relationship diagram

BLE fast network configuration , As a specific application of BLE server , Its relationship diagram is as above ;

Wm_ble_server_wifi_prof undertakes the service function , That is, data transmission processing ;

wm_ble_server_wifi_cfg handles specific communication protocol processing. Such a hierarchical structure separates application processing from specific

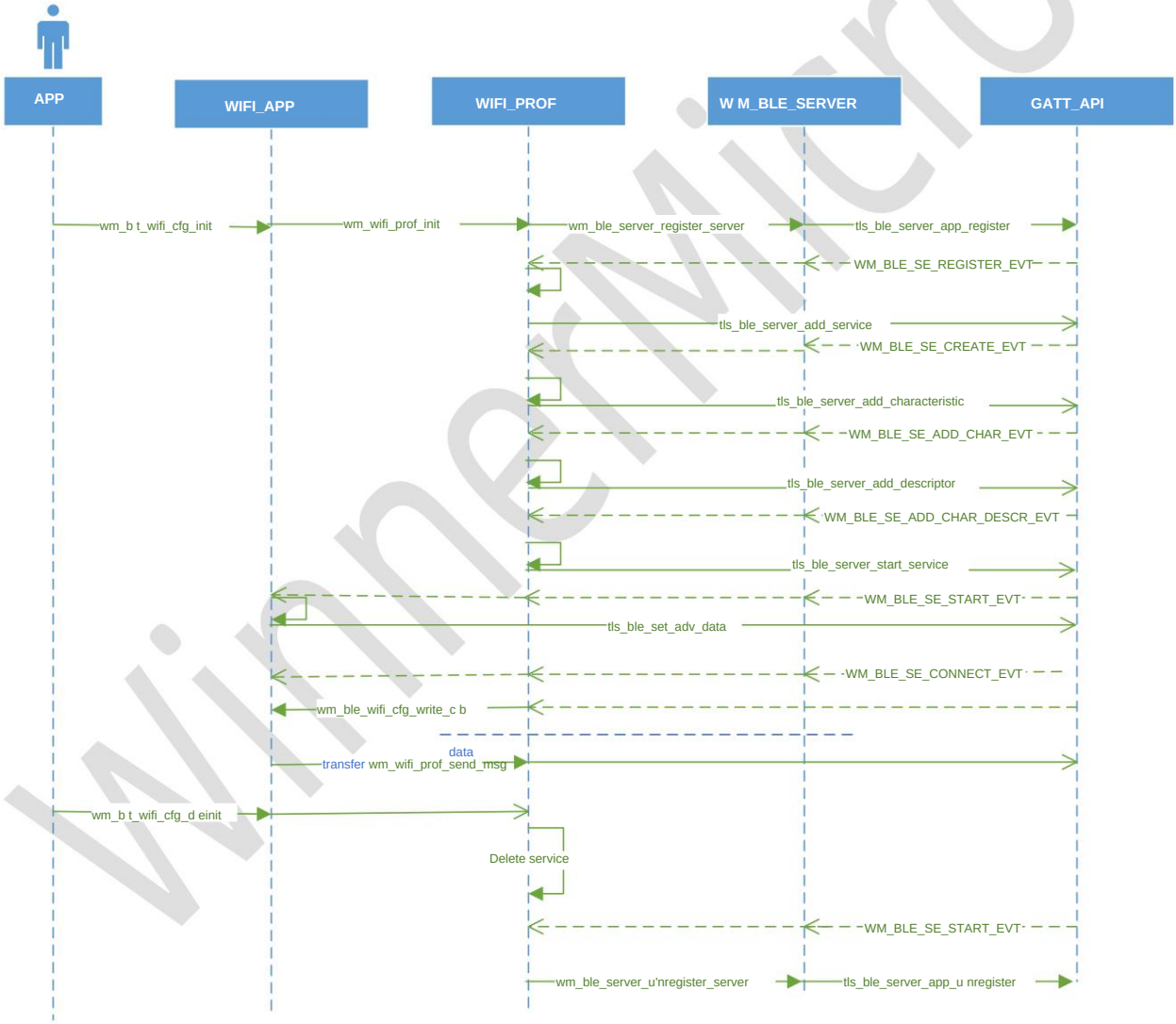
The input layer is independent, and the logical layer call is clearer.

This part of the API is relatively simple, as follows:

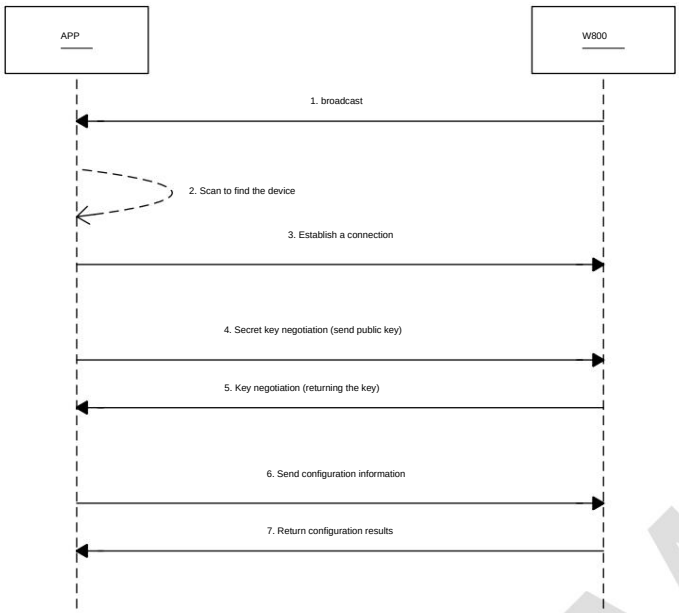
No API	name	describe
1	tls_bt_status_t tls_bt_enable(tls_bt_host_callback_t *scb, tls_bt_hci_if_t *uart, tls_bt_log_level_t log_level)	To run the Bluetooth system, this function will depend on Secondary enable host protocol and controller protocol stack.
2	tls_wifi_set_oneshot_flag(flag) flag 0: closed oneshot 1: UDPÿbroadcast+multicastÿ 2: AP+socket	Start/stop distribution network Note: 1. After the network configuration is successful, the BLE The distribution network service will automatically withdraw out, the radio is turned off. If you need to configure again

	3: AP+WEBSERVER 4: BT	Please call this API again. 2. If the network distribution fails, the user can secondary configuration
3 See 3.1 Bluetooth system logout description		

3.5.1 Software module calling relationship



3.5.2 Example of application process



3.5.3 Auxiliary Distribution Network Service

Service definition:

Service uuidÿ 0x1824

Feature value uuid: 0x2ABC Write & Indication

Feature value description uuid: 2902

Writeÿ BleWiFiÿÿÿ APP -> W800ÿCharacteristic UUIDÿ0x2ABC

Indication: BleWiFi (W800 -> mobile APP) Characteristic UUID: 0x2ABC

3.6 Users realize their own distribution network service

Refer to the example wm_ble_server_demo_prof.c to add a custom service.

4 API usage examples

After the W800 Bluetooth function is powered on, it is disabled by default. If the user wants to use Bluetooth by default, please refer to the following instructions.

4.1 Enable the Bluetooth system (exit)

Step 1, call in the tls_bt_entry() function to turn on the Bluetooth function, and turn off the Bluetooth system call demo_bt_destroy;

```
void tls_bt_entry()
{
    demo_bt_enable(); //turn on bluetooth system;
}

void tls_bt_exit()
{
    demo_bt_destroy(); //turn off bluetooth system;
}
```

Step 2, after the Bluetooth function is successfully turned on, the following callback function will be called, and the user can add his own application;

```
void app_adapter_state_changed_callback(tls_bt_state_t status)
{
    tls_bt_property_t btp;
    tls_bt_host_msg_t msg;
    msg.adapter_state_change.status = status;
    TLS_BT_APPL_TRACE_DEBUG("adapter status = %s\r\n", status==WM_BT_STATE_ON?"bt_state_on":"bt_state_off");

    bt_adapter_state = status;

    #if (TLS_CONFIG_BLE == CFG_ON)

    if(status == WM_BT_STATE_ON)
    {
        TLS_BT_APPL_TRACE_VERBOSE("init base application\r\n");
        /* those funtions should be called basicly*/
        wm_ble_dm_init();
        wm_ble_client_init();
        wm_ble_server_init();

        //at here , user run their own applications;

        //application_run();
    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");
        wm_ble_dm_deinit();
        wm_ble_client_deinit();
        wm_ble_server_deinit();

        //here, user may free their application;

        //application_stop();
    }

    #endif
```

4.2 Start up and run (exit) the sample server

At the position marked in step 2 in Section 4.1, call wm_ble_server_api_demo_init();

At the position marked in step 2 in Section 4.1, call wm_ble_server_api_demo_deinit(); the application

The exit function of the program will be released automatically when the Bluetooth system exits. Of course, when the Bluetooth system is running, the user can also

quit your own application.

4.3 Start up and run (exit) the example client

At the position marked in step 2 in Section 4.1, call wm_ble_client_api_demo_init();

At the position marked in step 2 in section 4.1, call wm_ble_client_api_demo_deinit(); application

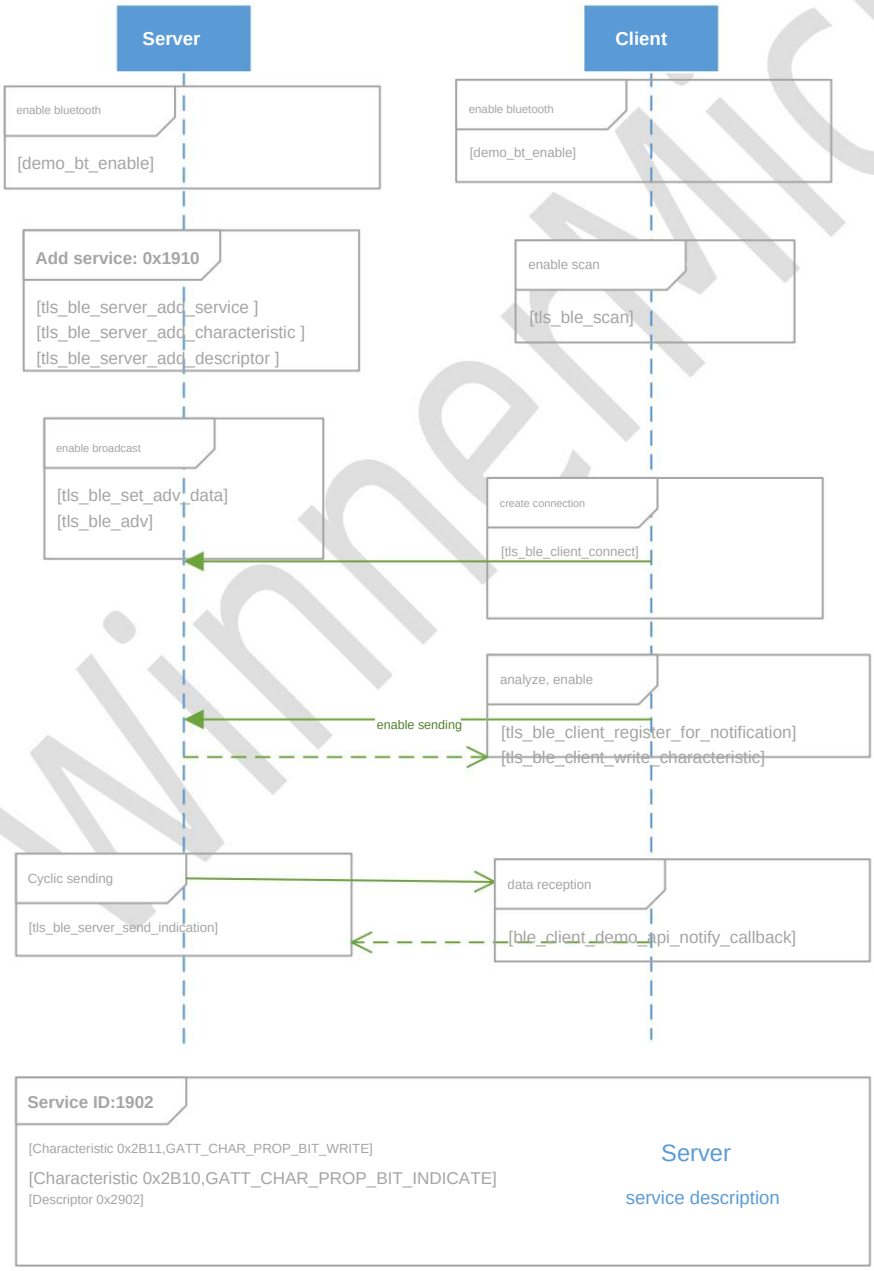
The exit function of the Bluetooth system will be released automatically when the Bluetooth system exits. Of course, when the Bluetooth system is running, users can also

Quit your application.

4.4 Data exchange function

Use two demo boards to run 4.2 server demo and 4.3 client demo respectively. At this point, the Server side

It will continuously send data to the client, and the timing diagram is as follows:



4.5 Multi-connection function

The W800 Bluetooth system acts as a central device and supports connection of up to 7 peripheral devices. An example configuration for this feature is as follows:

1. Run 7 BLE server devices separately, see 4.2 for the configuration mode

2. Run a BLE client that supports multi-connection function, see 4.4 for the configuration mode

At this point, the client will initiate scanning and connection functions one by one until it successfully connects to 7 BLE servers.

Note: Limited to the performance of the controller side, when the client initiates a connection, the connection parameters must use the following intervals:

4.6 UART transparent transmission function

Based on the data exchange between BLE server and BLE client, the transparent transmission function of UART is realized. Example of the function

The configuration is as follows:

1, Server side, using UART1, default attribute (115200-8-N-1) transparent transmission: called at the mark of chapter 4.1

tls_ble_uart_init(BLE_UART_SERVER_MODE, 0x01, NULL)ÿ

2, Client side, using UART1, default attribute (115200-8-N-1) transparent transmission: called at the mark of chapter 4.1

tls_ble_uart_init(BLE_UART_CLIENT_MODE, 0x01, NULL)ÿ

After startup, the server starts broadcasting, and the client scans and connects to the server. After the connection channel is established, the user can for data transmission via UART1.

4.7 Turn on the broadcast

Step 1, call in the tls_bt_entry() function to turn on the Bluetooth function, and turn off the Bluetooth system call demo_bt_destroy;

```
void tls_bt_entry()
{
    demo_bt_enable(); //turn on bluetooth system;
}

void tls_bt_exit()
{
    demo_bt_destroy(); //turn off bluetooth system;
}
```

Step 2, after the Bluetooth function is successfully turned on, The following callback function will be called, and the user calls the broadcast function

demo_ble_adv(1);//Connectable broadcast

```
void app_adapter_state_changed_callback(tls_bt_state_t status)
{
    tls_bt_host_msg_t msg;
    msg.adapter_state_change.status = status;
    TLS_BT_APPL_TRACE_DEBUG("adapter status = %s\r\n", status==WM_BT_STATE_ON?"bt_state_on":"bt_state_off");

    bt_adapter_state = status;

    #if (TLS_CONFIG_BLE == CFG_ON)

    if(status == WM_BT_STATE_ON)
    {
        TLS_BT_APPL_TRACE_VERBOSE("init base application\r\n");
        /* those funtions should be called basiclly*/
        wm_ble_dm_init();
        wm_ble_client_init();
        wm_ble_server_init();

        //at here , user run their own applications;
        //application_run();
        demo_ble_adv(1);
    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");
        wm_ble_dm_deinit();
        wm_ble_client_deinit();
        wm_ble_server_deinit();

        //here, user may free their application;
        //application_stop();
        demo_ble_adv(0);
    }

    #endif
    #if (TLS_CONFIG_BR_EDR == CFG_ON)
    /*class bluetooth application will be enabled by user*/
    #endif

    /*Notify at level application, if registered*/
    if(tls_bt_host_callback_at_ptr)
    {
        tls_bt_host_callback_at_ptr(WM_BT_ADAPTER_STATE_CHG_EVT, &msg);
    }

} ? end app_adapter_state_changed_callback ?
```

4.7.1 Default broadcast data configuration

```
static void ble_server_adv_enable_cb(uint8_t triger_id)
{
    tls_ble_adv(true);
}
static void ble_server_cfg_and_enable_adv()
{
    tls_ble_dm_adv_data_t data;
    uint8_t adv_data[] = {0x0C, 0x07, 0x00, 0x10};

    memset(&data, 0, sizeof(data));
    data.set_scan_rsp = false;
    data.include_name = true;
    data.manufacturer_len = 4;
    memcpy(data.manufacturer_data, adv_data, 4);

    /*configure the user specific data, 0xFF field*/
    tls_ble_set_adv_data(&data);

    /*enable advertisement*/
    tls_dm_evt_triger(0, ble_server_adv_enable_cb);

}
```

4.7.2 User-defined broadcast data settings

```
static void ble_server_adv_enable_cb(uint8_t triger_id)
{
    tls_ble_adv(true);
}
static void ble_server_cfg_and_enable_adv()
{
    tls_ble_dm_adv_data_t data;

    uint8_t adv_data[] = {0x03, 0x09, ' A' , 'B'};  //用户自己填充广播数据内容
    memset(&data, 0, sizeof(data));
    data.set_scan_rsp = false;
    data.pure_data = true;      //标记用户自己填充广播数据；数据内容为 adv_data
    data.manufacturer_len = 4;  //用户自己填充广播数据的长度；
    memcpy(data.manufacturer_data, adv_data, 4);

    /*configure the user specific data*/
    tls_ble_set_adv_data(&data);

    /*enable advertisement*/
    tls_dm_evt_triger(0, ble_server_adv_enable_cb);

}
```

4.8 Turn on the scan

Step 1, call in the tls_bt_entry() function to turn on the Bluetooth function, and turn off the Bluetooth system call demo_bt_destroy;

```
void tls_bt_entry()
{
    demo_bt_enable(); //turn on bluetooth system;
}

void tls_bt_exit()
{
    demo_bt_destroy(); //turn off bluetooth system;
}
```

Step 2. After the Bluetooth function is successfully turned on, the following callback function will be called, and the user calls the scan function

```
void app_adapter_state_changed_callback(tls_bt_state_t status)
{
    tls_bt_host_msg_t msg;
    msg.adapter_state_change.status = status;
    TLS_BT_APPL_TRACE_DEBUG("adapter status = %s\r\n", status==WM_BT_STATE_ON?"bt_state_on":"bt_state_off");

    bt_adapter_state = status;

    #if (TLS_CONFIG_BLE == CFG_ON)

    if(status == WM_BT_STATE_ON)
    {
        TLS_BT_APPL_TRACE_VERBOSE("init base application\r\n");
        /* those funtions should be called basiclly*/
        wm_ble_dm_init();
        wm_ble_client_init();
        wm_ble_server_init();

        //at here , user run their own applications;
        //application_run();
        demo_ble_scan(1);
    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");
        wm_ble_dm_deinit();
        wm_ble_client_deinit();
        wm_ble_server_deinit();

        //here, user may free their application;
        //application_stop();
        demo_ble_scan(0);
    }

    #endif
    #if (TLS_CONFIG_BR_EDR == CFG_ON)
    /*class bluetooth application will be enabled by user*/
    #endif

    /*Notify at level application, if registered*/
    if(tls_bt_host_callback_at_ptr)
    {
        tls_bt_host_callback_at_ptr(WM_BT_ADAPTER_STATE_CHG_EVT, &msg);
    }

} ? end app_adapter_state_changed_callback ?
```

```c
static void demo_ble_scan_report_cb(tls_ble_dm_evt_t event, tls_ble_dm_msg_t *p_data)
{
    if((event != WM_BLE_DM_SCAN_RES_EVT) && (event != WM_BLE_DM_SCAN_RES_CMPL_EVT)) return;

#define BLE_SCAN_RESULT_LEN 256
    int len = 0, i= 0;
    char *buf;

    buf = tls_mem_alloc(BLE_SCAN_RESULT_LEN);
    if (!buf)
    {
        return;
    }

    switch(event)
    {
        case WM_BLE_DM_SCAN_RES_EVT:
            {
                tls_ble_dm_scan_res_msg_t *msg = (tls_ble_dm_scan_res_msg_t *)&p_data->dm_scan_result;
                u8 valid_len;
                u8 device_name[64] = {0};
                memset(buf, 0, BLE_SCAN_RESULT_LEN);
                memset(device_name, 0, sizeof(device_name));
                valid_len = get_valid_adv_length_and_name(msg->value, device_name);
                if(valid_len > 62)
                {
                    //printf("###warning(%d)###\r\n", valid_len);
                    valid_len = 62;
                }
                len = sprintf(buf, "%02X%02X%02X%02X%02X%02X,%d,",
                              msg->address[0], msg->address[1], msg->address[2],
                              msg->address[3], msg->address[4], msg->address[5], msg->rssi);
                if(device_name[0] != 0x00)
                {
                    len += sprintf(buf +len, "\"%s\",", device_name);
                }else
                {
                    len += sprintf(buf+len, "\"\",");
                }

                for (i = 0; i < valid_len; i++)
                {
                    len += sprintf(buf + len, "%02X", msg->value[i]);
                }

                len += sprintf(buf + len, "\r\n");
                buf[len++] = '\0';
                printf("%s\r\n", buf);
            }
            break;
        case WM_BLE_DM_SCAN_RES_CMPL_EVT:
            {
                tls_ble_dm_scan_res_cmpl_msg_t *msg = (tls_ble_dm_scan_res_cmpl_msg_t *)&p_data->dm_scan_result_cmpl;
                printf("scan ended, ret=%d\r\n", msg->num_responses);
                bt_adapter_scaning = 0;
            }
            break;
        default:
            break;
    } ? end switch event ?
    if (buf)
        tls_mem_free(buf);

} ? end demo_ble_scan_report_cb ?

tls_bt_status_t demo_ble_scan(bool start)
{
    tls_bt_status_t ret;

    if(start)
    {
        ret = tls_ble_scan(TRUE);

        if(ret == TLS_BT_STATUS_SUCCESS)
        {
            bt_adapter_scaning = 1;
            ret = wm_ble_register_report_evt(WM_BLE_DM_SCAN_RES_EVT|WM_BLE_DM_SCAN_RES_CMPL_EVT, demo_ble_scan_report_cb);
        }
    }
    else
    {
        ret = tls_ble_scan(FALSE);

        if(ret == TLS_BT_STATUS_SUCCESS)
        {
            //wait scan stop;
            while(bt_adapter_scaning)
            {
                tls_os_time_delay(500);
            }
            //unregister the callback
            ret = wm_ble_deregister_report_evt(WM_BLE_DM_SCAN_RES_EVT|WM_BLE_DM_SCAN_RES_CMPL_EVT, demo_ble_scan_report_cb);
        }

    }

    return ret;
} ? end demo_ble_scan ?
```

4.9 Start broadcasting/scanning in connected state

Step 1, call in the tls_bt_entry() function to turn on the Bluetooth function, and turn off the Bluetooth system call demo_bt_destroy;

```
void tls_bt_entry()
{
    demo_bt_enable(); //turn on bluetooth system;
}

void tls_bt_exit()
{
    demo_bt_destroy(); //turn off bluetooth system;
}
```

The connection state is divided into Slave mode and Master mode. The following two situations are described respectively. ;

4.9.1 Connection state in Slave mode

Step 2, in Slave mode, see section 4.2. Run the demo example of Ble server, after running, the mobile phone

The terminal initiates scanning and connection operations. After the connection is successful, the device side is in Slave mode at this time, and the mobile phone side is in Master mode.

Mode. See wm_ble_server_api.c:

```
static void ble_server_connection_cb(int conn_id, int server_if, int connected, tls_bt_addr_t *bda)
{
    g_conn_id = conn_id;
    memcpy(&g_addr, bda, sizeof(tls_bt_addr_t));

    TLS_BT_APPL_TRACE_API("%s , connected=%d\r\n", __FUNCTION__, connected);

    if(connected)
    {
        /*Update connection parameter 5s timeout, if you need */
        //tls_ble_conn_parameter_update(bda, 16, 32, 0, 300);
        //connected with smart phone already;
    }
    else
    {
        g_conn_id = -1;
    }
}
```

4.9.1.1 Enable Broadcasting

Step 3, [Note] At this time, the device side only supports non-connectable broadcasts.

Call demo_ble_adv(2); // Unconnectable broadcast type is 2

```
int demo_ble_adv(uint8_t type)
{
    TLS_BT_APPL_TRACE_VERBOSE("demo_ble_adv=%d\r\n",type);
    if(type)
    {
        tls_ble_dm_adv_data_t data;

        uint8_t adv_data[] = {
               0x05,0x09, 'A', 'F', '1', '5',
               0x02,0x01,0x05,
               0x03,0x19,0xc1, 0x03};
        memset(&data, 0, sizeof(data));
        data.set_scan_rsp = false;
        data.pure_data = true;          //only manufacture data is includced in the advertisement payload
        data.manufacturer_len = 13;    //configure payload length;
        memcpy(data.manufacturer_data, adv_data, 13);//copy payload ;
        tls_ble_set_adv_data(&data); //configure advertisement data;

        uint8_t scan_resp_data[] = {0x05,0x09,'A','B','C','D'};
        memset(&data, 0, sizeof(data));
        data.set_scan_rsp = true;
        data.pure_data = true;          //only manufacture data is includced in the scan response payload
        data.manufacturer_len = 6;    //configure payload length;
        memcpy(data.manufacturer_data, scan_resp_data, 6);//copy payload ;

        tls_ble_set_adv_data(&data); //configure advertisement data;

        tls_ble_dm_adv_param_t adv_param;
        if(type == 1)
        {
            adv_param.adv_int_min = 0x40; //interval min;
            adv_param.adv_int_max = 0x60; //interval max;

        }else
        {
            adv_param.adv_int_min = 0xA8; //for nonconnectable advertisement, interval min is 0xA0;
            adv_param.adv_int_max = 0xA8; //interval max;
        }
        adv_param.dir_addr = NULL;     //directed address NULL;
        tls_ble_set_adv_param(&adv_param); //configure advertisement parameters;

        /*enable advertisement*/
        tls_dm_evt_triger(type, ble_adv_enable_cb);
    } ? end if type ? else
    {
        tls_ble_adv(0);
    }

    return TLS_BT_STATUS_SUCCESS;
} ? end demo_ble_adv ?
```

4.9.1.2 Start scanning

Step 4 Refer to 4.4, just call the scanning API directly.

demo_ble_scan(1);

4.9.2 Connection state in Master mode

Refer to 4.3 Start up and run the demo client function, after the client establishes a connection with the server:

1) Scanable operation;

2) Unconnectable broadcast operations can be sent

5 Bluetooth AT command

5.1 Brief description of Bluetooth AT commands

The Bluetooth system can be controlled through the Bluetooth AT command, and the Bluetooth AT command is divided into 4 categories. The host and controller part are used to configure

Set the host protocol stack and controller protocol stack, the application layer part is used to configure the Bluetooth application program, and the test part is used to configure the Bluetooth

Authentication function (this section partially contains the application layer).

The meaning of the abbreviation in the Bluetooth AT command is:

abbreviation	meaning
CTRL	CONTROLLER
BLESC	BLE SERVICE
BLESV	BLE SERVER
FLASH	BLE CLIENT
POW	POWER
STS	STATUS
OF THE	DESTORY
PRM	PARAM
FLT	FILTER
CT	CREATE
CH	CHARACTERISTIC
STT	START
STP	STOP
OF	DELETE

DIS	DISCONNECT
SND	SEND
IN	INDICATION
CONN	CONNECT
NTY	NOTIFICATION
ACC	ACCESS
TEST	TESTMODE
IN	ENABLE
GS	GETSTATUS
TPS	TXPOWERSET
TPG	TXPOWERGET

5.2 Bluetooth system AT command

5.2.1.1 AT+BTEN

Function:

Enable the Bluetooth system.

Format (ASCII):

AT+BTEN=<uart_no>,<log_level><CR> +OK=<status>,<adapter_status><CR><LF><CR><LF>

parameter:

uart_no: serial port index number, defined as follows:

value	meaning

1	uart1 The current version only supports UART1

Log_level: log output level, defined as follows:

value	meaning
0	Turn off log output
1	Output error level log
2	Output warn level log
3	Output api level log
4	Output event level log
5	Output debug level log
6	Output verbose level log

return:

status: command response result

value	meaning
0	success
Others>1 failed	

adapter_status: command response result

value	meaning
1	controller running
0 Controller stopped	

5.2.1.2 AT+BTDES

Function:

Stop and log off the Bluetooth system.

Format (ASCII):

AT+BTDES<CR> +OK=<status>,<adapter_status><CR><LF><CR><LF>

parameter:

See BTEN parameter description

5.3 Bluetooth host protocol stack AT command

5.3.1.1 AT+BTCFGHOST

Function:

Initialize and start or stop and unregister the host stack. Note that before starting the host protocol stack, the controller must be started first

protocol stack.

Format (ASCII):

AT+BTCFGHOST=[cmd]<CR> +OK<CR><LF><CR><LF>

parameter:

cmd: host protocol stack control command, defined as follows:

value	meaning
0	Stop and unregister the host stack
1	Initialize and start the host protocol stack

5.4 Bluetooth controller protocol stack AT command

5.4.1.1 AT+BTCTRLEN

Function:

Initialize and start the controller stack.

Format (ASCII):

AT+BTCTRLEN=<uart_no>,<log_level><CR> +OK<CR><LF><CR><LF>

parameter:

uart_no: serial port index number, defined as follows:

value	meaning
1	uart1, the current version only supports UART1

Log_level: log output level, defined as follows:

value	meaning
0	Turn off log output
1	Output error level log

5.4.1.2 AT+BTCTRLDES

Function:

Stop and unregister the controller stack.

Format (ASCII):

AT+BTCTRLDES<CR> +OK<CR><LF><CR><LF>

parameter:

none.

5.4.1.3 AT+BTCTRLGS

Function:

Get control status.

Format (ASCII):

AT+BTCTRLGS<CR> +OK=<status><CR><LF><CR><LF>

parameter:

status: control status, the return format is defined as follows:

TLS_BT_CTRL_IDLE = (1<<0),

TLS_BT_CTRL_ENABLED = (1<<1),

TLS_BT_CTRL_SLEEPING = (1<<2),

TLS_BT_CTRL_BLE_ROLE_MASTER = (1<<3),

TLS_BT_CTRL_BLE_ROLE_SLAVE = (1<<4),

TLS_BT_CTRL_BLE_ROLE_END = (1<<5),

TLS_BT_CTRL_BLE_STATE_IDLE = (1<<6),

TLS_BT_CTRL_BLE_STATE_ADVERTISING = (1<<7),

TLS_BT_CTRL_BLE_STATE_SCANNING = (1<<8),

TLS_BT_CTRL_BLE_STATE_INITIATING = (1<<9),

TLS_BT_CTRL_BLE_STATE_STOPPING = (1<<10),

TLS_BT_CTRL_BLE_STATE_TESTING = (1<<11),

5.4.1.4 AT+BTSLEEP

Function:

Set the sleep mode when the controller is idle. The current version does not support

Format (ASCII):

AT+BTSLEEP=<cmd><CR> +OK<CR><LF><CR><LF>

parameter:

cmd: control command, defined as follows:

value	meaning
0	Prevent the controller from entering sleep
1	Allow the controller to go to sleep

5.4.1.5 AT+BLETPS

Function:

Configure the transmit power for a specific type of BLE. The current version only supports the default power setting

Format (ASCII):

AT+BLETPS=<type>,<level><CR>

```
+OK<CR><LF><CR><LF>
```

parameter:

type: ble type, defined as follows:

value	meaning
0	specific connection handle
1	specific connection handle
2	specific connection handle
3	specific connection handle
4	specific connection handle
5	specific connection handle
6	specific connection handle
7	specific connection handle
8	specific connection handle
9	broadcast
10	scanning
11	default power

level: power index value.

value	Meaning dBm
1	1
2	4
3	7
4	10
5	13

5.4.1.6 AT+BLETPG

Function:

Get BLE specific type. The current version only supports default power gain

Format (ASCII):

AT+BLETPG=?<type><CR> +OK=<level><CR><LF><CR><LF>

parameter:

type: ble type, defined as follows:

value	meaning
0	specific connection handle
1	specific connection handle
2	specific connection handle
3	specific connection handle
4	specific connection handle
5	specific connection handle
6	specific connection handle
7	specific connection handle
8	specific connection handle
9	broadcast
10	scanning
11	default power

level: power index value. See 4.4.1.5

5.4.1.7 AT+BTTXPOW

Function:

Set/query transmit power index.

Format (ASCII):

AT+BTTXPOW=[?]<min>,<max><CR> +OK=<min>,<max><CR><LF><CR><LF>

parameter:

min: The minimum value of the power, the minimum value is 1, which means the minimum power is 1dBm, and the step size of each increase is 3db.

max: The maximum value of the power, the maximum value is 5, which means the maximum power is 13dBm, and the step size of each decrease is 3db.

5.4.1.8 AT+BTSCOPATH

Function:

Specifies the sco link output path. The current version does not support

Format (ASCII):

AT+BTSCOPATH=[?]<path><CR> +OK<CR><LF><CR><LF>

parameter:

path: output path, defined as follows:

value	meaning
0	PCM over HCM

1	Internal Interface

5.4.1.9 AT+BTTEST

Function:

Set the bluetooth test mode.

Format (ASCII):

AT+BTTEST=<mode><CR> +OK<CR><LF><CR><LF>

parameter:

mode: test mode, defined as follows:

value	meaning
0	Exit Bluetooth test mode
1	Enter Bluetooth test mode

5.5 Bluetooth application layer AT command

The Bluetooth application layer is divided into three parts: device management, BLE server and BLE client.

5.5.1 AT commands for device management

5.5.1.1 AT+BLEADV

Function:

Control BLE broadcast sending and stopping.

Format (ASCII):

AT+BLEADV=<mode><CR> +OK<CR><LF><CR><LF>

parameter:

mode: control mode, defined as follows:

value	meaning
0	Stop BLE broadcasting
1	Start BLE broadcast

5.5.1.2 AT+BLEADATA

Function: configure BLE broadcast content.

Format (ASCII):

AT+BLEADATA=<data><CR> +OK<CR><LF><CR><LF>

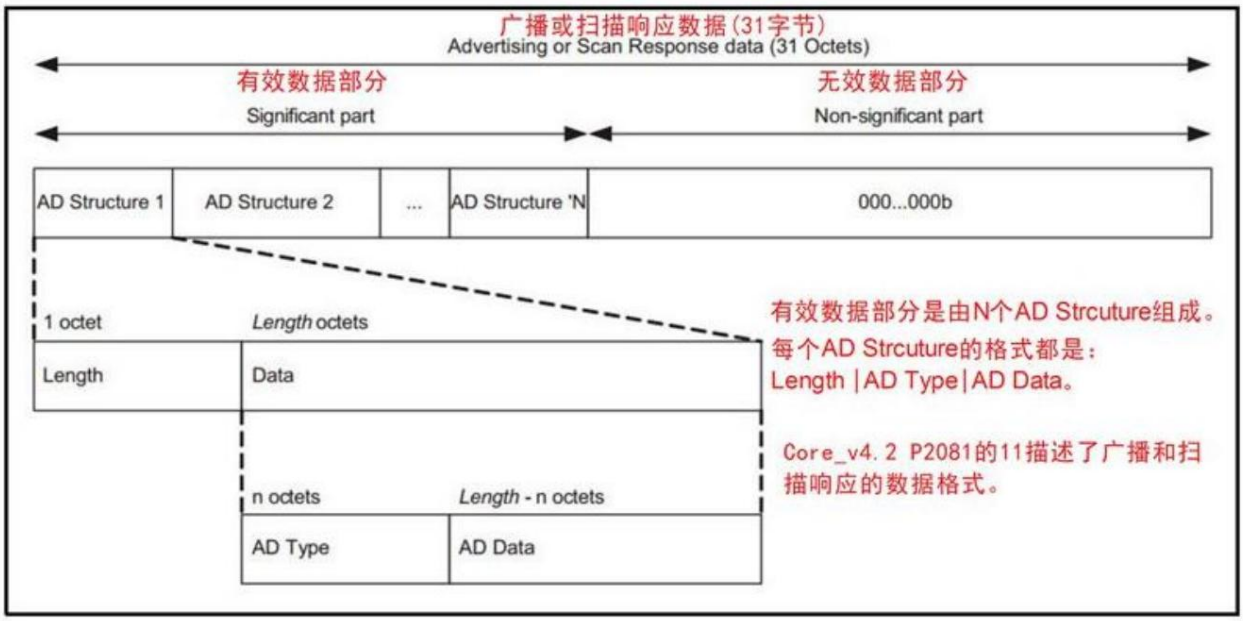
parameter:

data: Broadcast content, in HEX format. The maximum length is 62 characters, equivalent to 31 bytes in hexadecimal.

For example, set broadcast data as 0x02 0x01 0x06 0x03 0x09 0x31 0x32,

Then the setting command is: AT+BLEADATA=02010603093132

For the specific definition of the broadcast data format, see the description of the response core specification.



5.5.1.3 AT+BLEAPRM

Function: Configure BLE broadcast parameters.

Format (ASCII):

AT+BLEAPRM=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<ch annel_map>,[adv_filter_policy],[peer_addr_type],[peer_addr]<CR> +OK=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_m ap>,<adv_filter_policy>,<peer_addr_type>,<peer_addr><CR><LF><CR><LF>

parameter:

adv_int_min: Minimum broadcast interval, value range: 0x0020 ~ 0x4000. Note that when the broadcast type value is greater than

When equal to 3, the value range: 0Xa0~0x4000

adv_int_max: maximum broadcast interval, value range: 0x0020 ~ 0x4000. Note that when the broadcast type value is greater than

When equal to 3, the value range: 0Xa0~0x4000

adv_int_min and adv_int_max fill in the hexadecimal format, such as 10, FF, etc.

adv_type: broadcast type, defined as follows:

value	meaning
1	ADV_TYPE_IND Scannable Connectable Undirected Advertisement
2	ADV_TYPE_DIRECT_IND_HIGH connectable fast directional broadcast
3	ADV_TYPE_SCAN_IND Scannable Unconnectable Undirected Advertisements
4	ADV_TYPE_NONCONN_IND non-connectable non-scannable non-directed broadcast
5	ADV_TYPE_DIRECT_IND_LOW connectable slow directional broadcast

own_addr_type: BLE address type, defined as follows: (This value is automatically added by the protocol stack according to the value of the privacy attribute

Fill, the AT command can be filled with 0 by default)

value	meaning
0	BLE_ADDR_TYPE_PUBLIC
1	BLE_ADDR_TYPE_RANDOM

channel_map: broadcast channel, defined as follows:

value	meaning
1	ADV_CHNL_37
2	ADV_CHNL_38
4	ADV_CHNL_39
7	ADV_CHNL_ALL

adv_filter_policy: filter, defined as follows:

value	meaning
0	ADV_FILTER_ALLOW_SCAN_ANY_CON_ANY
1	ADV_FILTER_ALLOW_SCAN_WLST_CON_ANY
2	ADV_FILTER_ALLOW_SCAN_ANY_CON_WLST
3	ADV_FILTER_ALLOW_SCAN_WLST_CON_WLST

peer_addr_type: peer BLE address type, defined as follows:

value	meaning
0	PUBLIC
1	RANDOM

peer_addr: peer BLE address.

5.5.1.4 AT+BLESCPRM

Function:

Configure BLE scanning parameters.

Format (ASCII):

AT+BLESCPRM=<window>,<interval>,<scan_mode><CR>
+OK<CR><LF><CR><LF>

parameter:

windows: scan windows. [0x0004, 0x4000], fill in the hexadecimal format, such as 10, FF, etc.

interval: scan interval. [0x0004, 0x4000]

scan_mode: scan mode. [0,1] passive scan, active scan

The value of interval should be greater than or equal to windows. When the interval is equal to windows, it means that the controller is always in the

In the scanning state, that is, the scanning window is always open.

5.5.1.5 AT+BLESCFLT

Function:

Configure scan filtering parameters.

Format (ASCII):

AT+BLESCFLT=<filter><CR> +OK<CR><LF><CR><LF>

parameter:

filter: filter parameters, the usage is temporarily ominous,

Note: This command is currently not supported.

5.5.1.6 AT+BLESCAN

Function:

Start or stop scanning.

Format (ASCII):

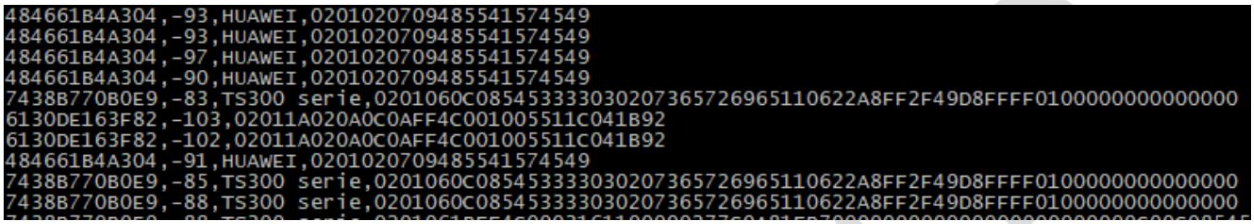
AT+BLESCAN=<mode><CR> +OK<CR><LF><CR><LF>

parameter:

mode: operation mode, defined as follows:

value	meaning
0	stop scanning
1	start scan

The scanning result is shown in the figure below:



5.5.1.7 AT+&BTNAME

Function:

Set the bluetooth name.

Format (ASCII):

AT+&BTNAME=[!]<name><CR>
+OK,<name><CR><LF><CR><LF> return when saving to flash
+OK,<CR><LF><CR><LF> Return when flash is not saved

parameter:

Name Bluetooth name, ASCII string. The maximum length is 16 bytes.

5.5.1.8 AT+&BTNAME

Function:

Get the bluetooth name.

Format (ASCII):

AT+&BTNAME

+OK,<name><CR><LF><CR><LF>

parameter:

Name Bluetooth name, ASCII string. The maximum length is 18 bytes.

5.5.1.9 AT+ BLESSCM

Function:

Specifies BLE to scan on a specific channel.

Format (ASCII):

AT+ BLESSCM=CH +OK

parameter:

CH is defined as:

value	meaning
1	Specify 37 channels to scan
2	Specify 38 channels to scan
4	Specify 39 channels to scan
7	Frequency hopping, scan at 37, 38, 39 in sequence (default)

5.5.1.10 AT+BTSCM

Function:

Configure the connectable discovery status of traditional Bluetooth.

Format (ASCII):

AT+ BTSCM=MODE +OK

parameter:

CH is defined as:

value	meaning
0	Invisible
1	connectable not discoverable
2	connectable, discoverable
7	Frequency hopping, scan at 37, 38, 39 in sequence (default)

5.5.2 BLE server AT command

This chapter describes how to use AT commands to create a BLE server step by step, and also provides an AT command to create a demo

The function of server is AT+BLEDS=1/0, 1 is used to create, 0 is used to logout. Note the server and 4.2 created at this time

The demo server described in the chapter is the same.

5.5.2.1 AT+BLECTSV

Function:

Create a server.

Format (ASCII):

AT+BLECTSV=<uuid><CR> +OK=<status><server_if><CR><LF><CR><LF>

parameter:

uuid: unique id, double byte.

status: command execution result, 0 is successful. Other, wrong.

server_if: server interface index number.

Note: w800 supports up to 7 gatt apps. These 7 include server and client. The current distribution is:

The server supports 3, and the client supports 4.

uuid definition: https://www.bluetooth.com/specifications/assigned-numbers/

5.5.2.2 AT+BLEADDSC

Function:

Add a service to server.

Format (ASCII):

AT+BLEADDSC=<server_if>,<inst_id>,<uuid>,<num_handles><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

inst_id: The default value is 1.

uuid: The uuid of this service.

num_handles: The default value is 5.

service_handle: The handle of the service value.

Note: w800 supports up to 8 services. Pay attention to TDS (uuid is 0x1824 for wifi distribution network). user

This uuid cannot be used.

For the definition of handles: the user creates a service and assigns a handle value. Each time a user adds a

The character is assigned 2 handles, and a handle is assigned for each description added by the user.

5.5.2.3 AT+BLEADDCH

Function:

Add a characteristic value to the service.

Format (ASCII):

AT+BLEADDCH=<server_if>,<service_handle>,<uuid>,<prop>,<perm><CR>
+OK=<status><server_if><service_handle><char_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

service_handle: Add the handle returned by the service.

uuid: unique id.

properties: encryption authorization description, in hexadecimal format, see 5.5.9.3 for specific definition values.

permissions: Read and write attributes, see 5.5.9.3 for specific definition values in hexadecimal format.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.4 AT+BLEADESC

Function:

Add a description value to the service.

Format (ASCII):

AT+BLEADESC=<server_if>,<service_handle>,<uuid>,<perm><CR>
+OK=<status><server_if><service_handle><desc_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

service_handle: Add the handle returned by the service.

uuid: The uuid of this description service.

permissions: Read and write attributes, see 4.5.5.3 for specific definition values in hexadecimal format.

status: command execution result, 0 is successful. Other, wrong.

desc_handle: The handle of the description service.

5.5.2.5 AT+BLESTTSC

Function:

Start the service.

Format (ASCII):

AT+BLESTTSC=<server_if>,<service_handle>,<tran_type><CR>
+OK=<status><server_if><service_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

service_handle: Add the handle returned by the service.

tran_type: BLE transmission type, the default value is 2.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.6 AT+BLESTPSC

Function:

Out of service.

Format (ASCII):

AT+BLESTPSC=<server_if>,<service_handle><CR>

+OK=<status><server_if><service_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

service_handle: Add the handle returned by the service.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.7 AT+BLEDELSC

Function:

Delete service.

Format (ASCII):

AT+BLEDELSC=<server_if>,<service_handle><CR> +OK=<status><server_if><service_handle><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

service_handle: Add the handle returned by the service.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.8 AT+BLEDESSV

Function:

Log out of the demo server.

Format (ASCII):

AT+BLEDESSV=<server_if><CR>

```
+OK=<status><server_if><CR><LF><CR><LF>
```

parameter:

server_if: Return value when created.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.9 AT+BLESCONN

Function:

Connect client. This feature is not currently supported

Format (ASCII):

```
AT+BLESCONN=[!?]<server_if>,<addr><CR>

+OK=<conn_id><CR><LF><CR><LF>
```

parameter:

server_if: Create the interface number returned by the server.

addr: Bluetooth mac address of the client.

conn_id: connection id.

5.5.2.10 AT+BLESVDIS

Function:

Disconnected client.

Format (ASCII):

```
AT+BLESVDIS=<server_if>,<addr>,<conn_id><CR>

+OK=<status><server_if><conn_indication><CR><LF><CR><LF>
```

parameter:

server_if: Create the interface number returned by the server.

addr: Bluetooth mac address of the client.

conn_indication: 1, connected, 0 disconnected

status: command execution result, 0 is successful. Other, wrong.

5.5.2.11 AT+BLESSING

Function:

Send indication.

Format (ASCII):

AT+BLESIND=<server_if><conn_id><attr_handle><data><CR>
+OK=<status><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

conn_id: The id number when creating the connection.

attr_handle: the return value when creating a feature value

data: the string entered by the user.

status: command execution result, 0 is successful. Other, wrong.

5.5.2.12 AT+BLESRSP

Function:

Read and write operations return values.

Format (ASCII):

AT+BLESRSP=<server_if><conn_id><attr_handle><data><CR>
+OK=<status><CR><LF><CR><LF>

parameter:

server_if: Create the interface number returned by the server.

conn_id: The id number when creating the connection.

attr_handle: the return value when creating a feature value

data: the string entered by the user.

status: command execution result, 0 is successful. Other, wrong.

5.5.3 BLE client AT command

This chapter describes how to use AT commands to create a BLE client step by step, and also provides an AT command to create a demo

The function of client is AT+BLEDC=1/0, 1 is used to create, 0 is used to logout. Note the client and 4.3 created at this time

The demo client described in the chapter is the same.

5.5.3.1 AT+BLECCT

Function:

Create a client with the specified uuid.

Format (ASCII):

AT+BLECCT=<uuid><CR> +OK=<status><client_if><CR><LF><CR><LF>

parameter:

uuid: unique id.

client_if: Create the interface number returned by the client.

status: command execution result, 0 is successful. Other, wrong.

Note: w800 supports up to 7 gatt apps. These 7 include server and client. The current distribution is:

The server supports 3, and the client supports 4.

5.5.3.2 AT+BLECCONN

Function:

Connect to the server.

Format (ASCII):

AT+BLECCONN=<client_if>,<addr><CR> +OK=<status><client_if><conn_id><CR><LF><CR><LF>

parameter:

client_if: Create the interface number returned by the client.

addr: Bluetooth mac address of the server.

conn_id: connection id.

status: command execution result, 0 is successful. Other, wrong.

5.5.3.3 AT+BLECSSC

Function:

Scan server's service list.

Format (ASCII):

AT+BLECSSC=<conn_id><CR> +OK=<status><CR><LF><CR><LF>

parameter:

conn_id: The id returned when connecting to the client.

status: command execution result, 0 is successful. Other, wrong.

5.5.3.4 AT+BLECGDB

Function:

Return the list of services.

Format (ASCII):

AT+BLECGDB=<conn_id><CR> +OK=<list><CR><LF><CR><LF>

parameter:

conn_id: The id returned when connecting to the client.

list: service list:

```
+OK,0,4,20
0x1801,T=0x00,HDL=0,PROP=0x00
        0x2a05,T=0x03,HDL=3,PROP=0x20
0x1800,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=22,PROP=0x02
        0x2a01,T=0x03,HDL=24,PROP=0x02
        0x2aa6,T=0x03,HDL=26,PROP=0x02
0xfe35,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=42,PROP=0x0a
        0x2a01,T=0x03,HDL=44,PROP=0x30
        0x2902,T=0x04,HDL=45,PROP=0x00
        0x2a02,T=0x03,HDL=47,PROP=0x08
        0x2a03,T=0x03,HDL=49,PROP=0x30
        0x2902,T=0x04,HDL=50,PROP=0x00
0x046a,T=0x00,HDL=0,PROP=0x00
        0x046c,T=0x03,HDL=53,PROP=0x0a
0x1821,T=0x00,HDL=0,PROP=0x00
        0x2a6f,T=0x03,HDL=56,PROP=0x0a
        0x2901,T=0x04,HDL=57,PROP=0x00
        0x2abc,T=0x03,HDL=59,PROP=0x1a
        0x2902,T=0x04,HDL=60,PROP=0x00
```

5.5.3.5 AT+BLECRNTY

Function:

Register for events that respond to server notifications.

Format (ASCII):

AT+BLECRNTY=<client_if>,<addr>,<attr_handle>,<conn_id><CR>
+OK=<status><conn_id><attr_handle><register_or_not><CR><LF><CR><LF>

parameter:

client_if: Create the interface number returned by the client.

addr: mac address.

attr_handle: The characteristic handle value of notification in the service list.

conn_id: return value when creating a connection

rcegister_or_not: indicates registration or cancellation.

5.5.3.6 AT+BLECDNTY

Function:

Unregisters registered notification response events.

Format (ASCII):

AT+BLECDNTY=<client_if>,<addr>,<attr_handle>,<conn_id><CR>
+OK=<status><conn_id><attr_handle><register_or_not><CR><LF><CR><LF>

parameter:

client_if: Create the interface number returned by the client.

addr: mac address.

attr_handle: The characteristic handle value of notification in the service list.

conn_id: return value when creating a connection

register_or_not: Indicates registration or cancellation.

5.5.3.7 AT+BLECACH

Function:

Read and write characteristicisc.

Format (ASCII):

AT+BLECACH=<mode>,<conn_id>,<handle>,<auth_req>,[data]<CR>
Write operation+OK=<status><conn_id><CR><LF><CR><LF>
Read operation+OK=<status><conn_id><length><data><CR><LF><CR><LF>

parameter:

mode: operation mode, defined as follows:

value	meaning
0	write operation

1	read operation

conn_id: The id returned when connecting to the client.

handle: handle for reading and writing characteristic values.

auth_req: Default is 0.

data: The data to be written, in string format, only valid for write operations.

5.5.3.8 AT+BLECDIS

Function:

Disconnect.

Format (ASCII):

AT+BLECDIS=<client_if>,<addr>,<conn_id><CR>
+OK=<status><client_if><conn_id><reason><CR><LF><CR><LF>

parameter:

client_if: Create the interface number returned by the client.

addr: mac address.

conn_id: The id returned when connecting to the client.

reason: If this command is initiated by 800, the value of reason is always 0;

If initiated by the APP side, see reason code definition, 4.5.5.2

5.5.3.9 AT+BLECDES

Function:

logout client,.

Format (ASCII):

AT+BLECDES=<client_if><CR> +OK=<status><client_if><CR><LF><CR><LF>

parameter:

client_if: The interface value assigned at creation time.

5.5.4 Example of server client communication based on AT command

Creating BLE server and BLE client based on AT commands requires two demo boards, one of which is responsible for server role, a demo board assumes the client role. After starting the Bluetooth function, run AT+BLEDS=1, AT+BLEDC=1. You can see that the server is constantly sending data to the client. See 4.4 for specific servcie description Data exchange function.

5.5.5 BLE assisted WiFi distribution network AT command

5.5.5.1 AT+ONESHOT

Function:

Start or stop the distribution network service.

Format (ASCII):

AT+ONESHOT=<mode><CR> +OK=<mode><CR><LF><CR><LF>

parameter:

mode: operation mode, defined as follows:

value	meaning
0	Stop distribution network
1	Start UDP distribution network
2	Start SoftAP+Socket distribution network
3	Start SoftAP+WebServer network configuration
4	Start Bluetooth distribution network

Notice:

After starting the Bluetooth distribution network, the user can use the mobile phone APP to configure the WiFi information. After the network distribution is successful, the network distribution service will automatically

Logout, Bluetooth turns off broadcasting. If you need to configure the network again, please start the Bluetooth distribution network again.

5.5.6 Traditional Bluetooth audio AT commands

Function:

Set to enable or disable the AUDIO sink function.

Format (ASCII):

AT+ BTAVS=<state><CR> +OK<CR><LF><CR><LF>

parameter:

state: Enable the AV SINK logout function, defined as follows:

value	meaning
0	Unregister the sink function
1	Enable the sink function

5.5.7 Traditional Bluetooth hands-free phone AT commands

Function:

Set to enable or cancel the HAND FREE function.

Format (ASCII):

AT+ BTHFP=<state><CR> +OK<CR><LF><CR><LF>

parameter:

state: enable the logout HAND FREE function, defined as follows:

value	meaning
0	Cancel the HAND FREE function
1	Enable the HAND FREE function

5.5.8 SPP AT command

Function:

Set to enable or disable the SPP server/client function.

Format (ASCII):

AT+ BTSPPS/ BTSPPC=<state><CR> +OK<CR><LF><CR><LF>

parameter:

state: enable logout of SPP server/client function, defined as follows:

value	meaning
0	Logout of the SPP server/client function
1	Enabling the SPP server/client function

5.5.9 Status code definition:

5.5.9.1 GATT Status Definition

BTA_GATT_OK	0x0000
BTA_GATT_INVALID_HANDLE	0x0001
BTA_GATT_READ_NOT_PERMIT	0x0002
BTA_GATT_WRITE_NOT_PERMIT	0x0003
BTA_GATT_INVALID_PDU	0x0004
BTA_GATT_INSUF_AUTHENTICATION	0x0005
BTA_GATT_REQ_NOT_SUPPORTED	0x0006
BTA_GATT_INVALID_OFFSET	0x0007
BTA_GATT_INSUF_AUTHORIZATION	0x0008
BTA_GATT_PREPARE_Q_FULL	0x0009
BTA_GATT_NOT_FOUND	0x000A
BTA_GATT_NOT_LONG	0x000B
BTA_GATT_INSUF_KEY_SIZE	0x000C
BTA_GATT_INVALID_ATTR_LEN	0x000D
BTA_GATT_ERR_UNLIKELY	0x000E
BTA_GATT_INSUF_ENCRYPTION	0x000F
BTA_GATT_UNSUPPORT_GRP_TYPE	0x0010
BTA_GATT_INSUF_RESOURCE	0x0011
BTA_GATT_NO_RESOURCES	0x80

BTA_GATT_INTERNAL_ERROR	0x81
BTA_GATT_WRONG_STATE	0x82
BTA_GATT_DB_FULL	0x83
BTA_GATT_BUSY	0x84
BTA_GATT_ERROR	0x85
BTA_GATT_CMD_STARTED	0x86
BTA_GATT_ILLEGAL_PARAMETER	0x87
BTA_GATT_PENDING	0x88
BTA_GATT_AUTH_FAIL	0x89
BTA_GATT_MORE	0x8a
BTA_GATT_INVALID_CFG	0x8b
BTA_GATT_SERVICE_STARTED	0x8c
BTA_GATT_ENCRYPED_NO_MITM	0x8d
BTA_GATT_NOT_ENCRYPTED	0x8e
BTA_GATT_CONGESTED	0x8f
BTA_GATT_DUP_REG	0x90
BTA_GATT_ALREADY_OPEN	0x91
BTA_GATT_CANCEL	0x92
BTA_GATT_CCC_CFG_ERR	0xFD
BTA_GATT_PRC_IN_PROGRESS	0xFE
BTA_GATT_OUT_OF_RANGE	0xFF

5.5.9.2 Reason code definition:

Success	0x00
Unknown HCI Command	0x01
Unknown Connection Identifier	0x02
Hardware Failure	0x03
Page Timeout	0x04
Authentication Failure	0x05
PIN or Key Missing	0x06
Memory Capacity Exceeded	0x07
Connection Timeout	0x08
Connection Limit Exceeded	0x09
Synchronous Connection Limit To A Device Exceeded	0x0a
ACL Connection Already Exists	0x0b
Command Disallowed	0x0c
Connection Rejected due to Limited Resources 0x0d	
Connection Rejected Due To Security Reasons	0x0e
Connection Rejected due to Unacceptable BD_ADDR	0x0f
Connection Accept Timeout Exceeded	0x10
Unsupported Feature or Parameter Value	0x11

Invalid HCI Command Parameters	0x12
Remote User Terminated Connection	0x13
Remote Device Terminated Connection due to Low Resources	0x14
Remote Device Terminated Connection due to Power Off	0x15
Connection Terminated By Local Host	0x16
Repeated Attempts	0x17
Pairing Not Allowed	0x18
Unknown LMP PDU	0x19
Unsupported Remote Feature / Unsupported LMP Feature	0x1a
SCO Offset Rejected	0x1b
SCO Interval Rejected	0x1c
SCO Air Mode Rejected	0x1d
Invalid LMP Parameters / Invalid LL Parameters 0x1e	
Unspecified Error	0x1f
Unsupported LMP Parameter Value / Unsupported LL Parameter Value	0x20
Role Change Not Allowed	0x21
LMP Response Timeout / LL Response Timeout 0x22	
LMP Error Transaction Collision	0x23

LMP PDU Not Allowed	0x24
Encryption Mode Not Acceptable	0x25
Link Key cannot be Changed	0x26
Requested QoS Not Supported	0x27
Instant Passed	0x28
Pairing With Unit Key Not Supported	0x29
Different Transaction Collision	0x2a
Reserved	0x2b
QoS Unacceptable Parameter	0x2c
QoS Rejected	0x2d
Channel Classification Not Supported	0x2e
Insufficient Security	0x2f
Parameter Out Of Mandatory Range	0x30
Reserved	0x31
Role Switch Pending	0x32
Reserved	0x33
Reserved Slot Violation	0x34
Role Switch Failed	0x35
Extended Inquiry Response Too Large	0x36
Secure Simple Pairing Not Supported By Host	0x37
Host Busy – Pairing	0x38
Connection Rejected due to No Suitable Channel 0x39	

Found	
Controller Busy	0x3a
Unacceptable Connection Parameters	0x3b
Directed Advertising Timeout	0x3c
Connection Terminated due to MIC Failure	0x3d
Connection Failed to be Established	0x3e
MAC Connection Failed	0x3f

5.5.9.3 Definition of Permissions and properties

```
/** Attribute permissions */
#define WM_GATT_PERM_READ                    (1 << 0) /**< bit 0 - 0x0001 */
#define WM_GATT_PERM_READ_ENCRYPTED (1 << 1) /**< bit 1 - 0x0002 */
#define WM_GATT_PERM_READ_ENC_MITM           (1 << 2) /**< bit 2 - 0x0004 */
#define WM_GATT_PERM_WRITE                   (1 << 4) /**< bit 4 - 0x0010 */
#define WM_GATT_PERM_WRITE_ENCRYPTED (1 << 5) /**< bit 5 - 0x0020 */
#define WM_GATT_PERM_WRITE_ENC_MITM (1 << 6) /**< bit 6 - 0x0040 */
#define WM_GATT_PERM_WRITE_SIGNED            (1 << 7) /**< bit 7 - 0x0080 */
#define WM_GATT_PERM_WRITE_SIGNED_MITM (1 << 8) /**< bit 8 - 0x0100 */


/** definition of characteristic properties */
#define WM_GATT_CHAR_PROP_BIT_BROADCAST (1 << 0) /**< 0x01 */
#define WM_GATT_CHAR_PROP_BIT_READ           (1 << 1) /**< 0x02 */
#define WM_GATT_CHAR_PROP_BIT_WRITE_NR       (1 << 2) /**< 0x04 */
```

```
#define WM_GATT_CHAR_PROP_BIT_WRITE                    (1 << 3) /**< 0x08 */
#define WM_GATT_CHAR_PROP_BIT_NOTIFY                   (1 << 4) /**< 0x10 */
#define WM_GATT_CHAR_PROP_BIT_INDICATE                 (1 << 5) /**< 0x20 */
#define WM_GATT_CHAR_PROP_BIT_AUTH                     (1 << 6) /**< 0x40 */
#define WM_GATT_CHAR_PROP_BIT_EXT_PROP                 (1 << 7) /**< 0x80 */
```

6 Example of Bluetooth AT command operation

This chapter combines specific examples to give the specific operation specifications of Bluetooth AT commands. The black screenshot is the response to the AT command.

6.1 Enable and exit the Bluetooth system

6.1.1 Enable Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.1.2 Exit the Bluetooth system

AT+BTDES

```
+OK=0,0
```

6.2 Enable auxiliary WiFi distribution network service

6.2.1 Turn on the Bluetooth function and enable the network distribution service

```
AT+BTEN=1.0                    //Enable the bluetooth system
AT+ONESHOT=4 //Enable distribution network service
```

At this time, you can use the APP to perform network configuration operations; note that after the network configuration is successful, the system will automatically log out of the network distribution service.

```
+OK=0,1

+OK
```

6.2.2 Exit the auxiliary WiFi distribution network service and log off the Bluetooth system

AT+ONESHOT=0 //Exit distribution network service

AT+BTDES //Exit the bluetooth system

6.3 BLE server operation example

This chapter describes how to create a BLE server step by step through AT commands and communicate with the Nrf connect APP on the mobile phone. interoperability.

6.3.1 Enable Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.3.2 Create server

AT+BLECTSV=9999 //Create a server with uuid 9999

```
+OK=0,4
```

6.3.3 Add service

AT+BLEADDSC=4,1,1826,5 //Add service with uuid 1826

//1824uuid, dedicated for Bluetooth distribution network.

```
+OK=0,4,40
```

6.3.4 Adding eigenvalues

AT+BLEADDCH=4,40,2abc,28,11 //Add uuid as the characteristic value of 2abc

```
+OK=0,4,40,42
```

6.3.5 Add feature value description

AT+BLEADESC=4,40,2902,11 //Add a feature value description with uuid 2902

```
+OK=0,4,40,43
```

6.3.6 Start the service

AT+BLESTTSC=4,40,2 //Start the service

```
+OK=0,4,40
```

6.3.7 Configure broadcast data

AT+BLEADVDATA=0201060309574D2D30363A30313A3335

//Set broadcast content, broadcast type and name field

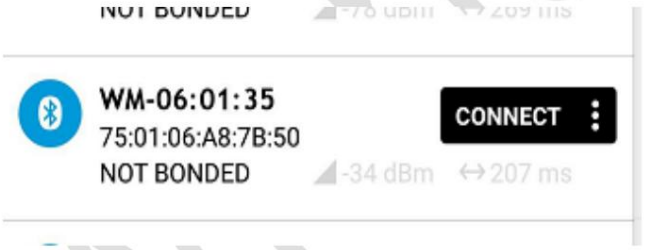
6.3.8 Start broadcasting

AT+BLEADV=1 //enable broadcast

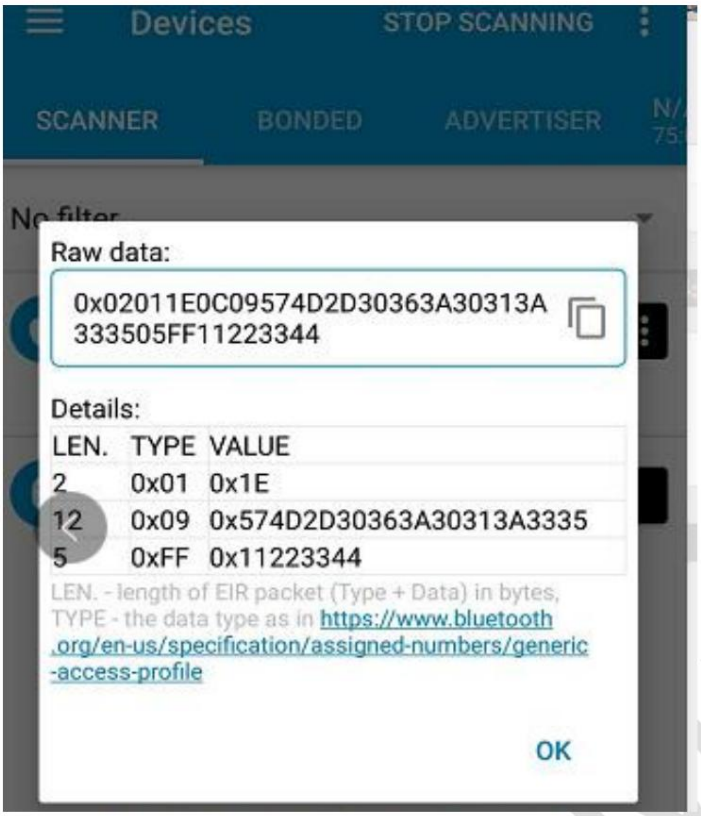
```
+OK
```

6.3.9 The mobile phone starts scanning

Nrf connect scan results:



The broadcast data on the mobile phone shows:



6.3.10 Initiate a connection on the mobile phone side

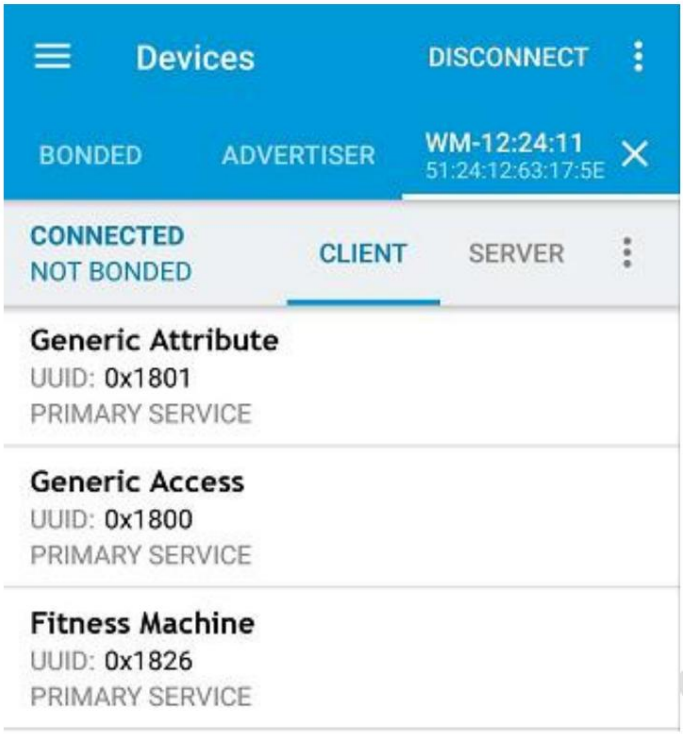
Click the CONNECT button, then w800 will display:

```
+OK=0,4,4,1,60C25D5A4CC1
```

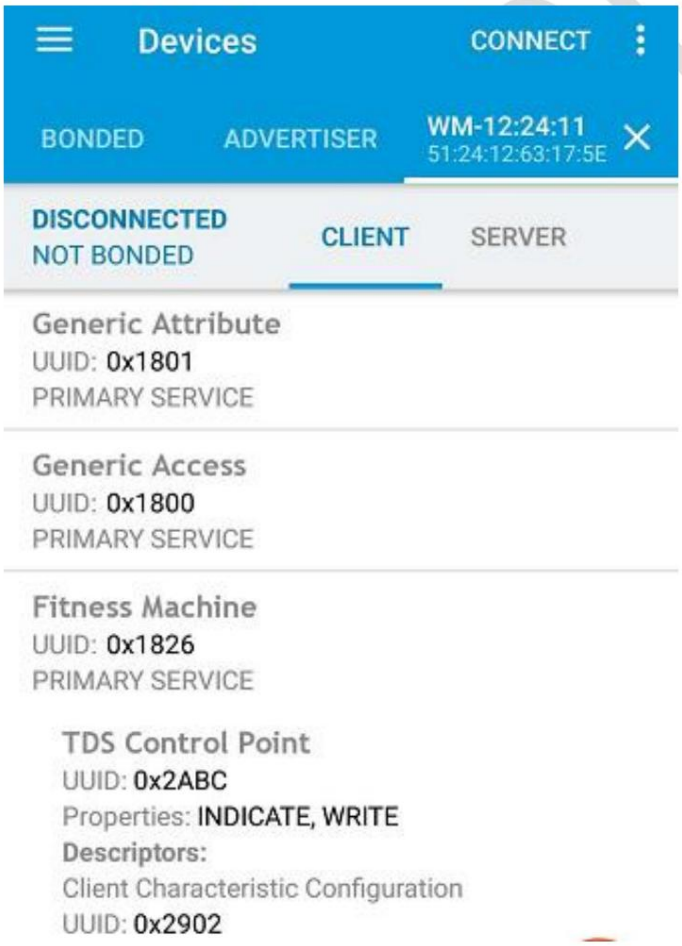
That is, the MAC address of the mobile phone is 60C25D5A4CC1 and the connection is successful. After the connection is successful, you can see us on the phone side

Created service description.

In the figure below, UUID:0x1826 is the service we created.



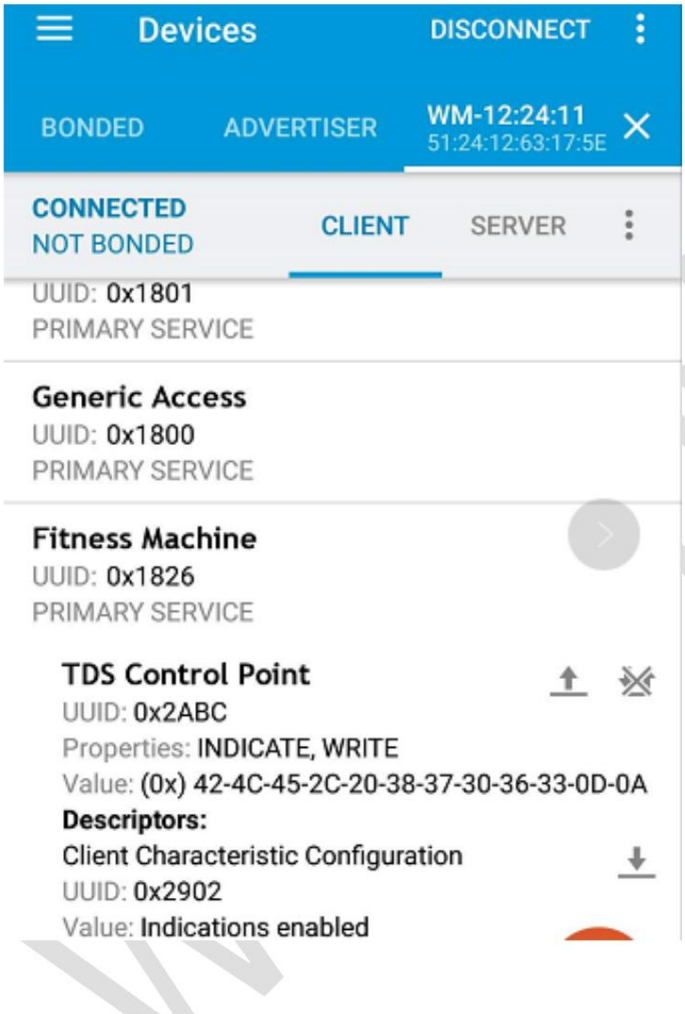
Click Transport Discovery to see the property values and descriptions we created



6.3.11 Enable the Indication function on the mobile phone side

Click the up and down arrows on the right side of 0x2abc to enable the indicate operation. After that, W800 will

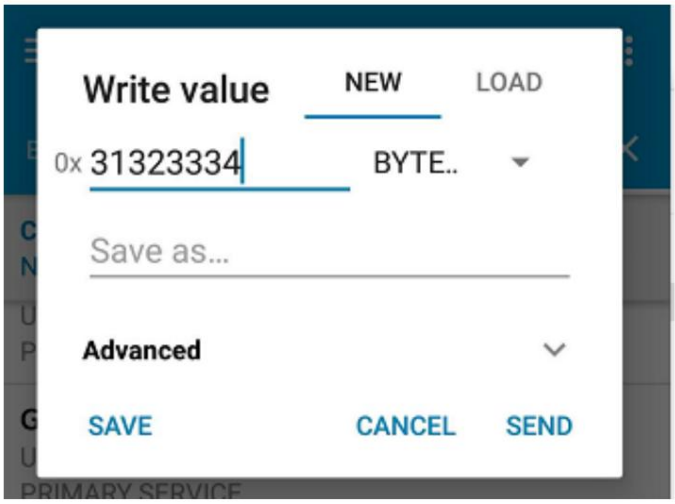
Send a string to the mobile phone, the display is as follows: BLE, the current system time, the HEX format shown below



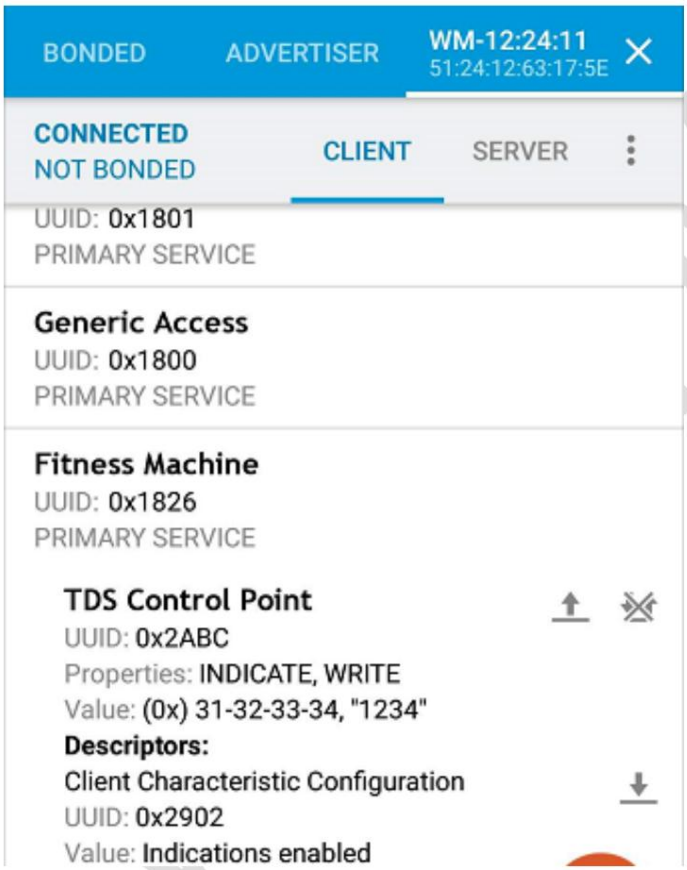
Click the up and down arrows on the right side of 0X2ABC again, the shape is now, which means stop sending indication.

6.3.12 Obtaining characteristic value data by mobile phone profile

Click the upward arrow on the right side of 0X2ABC, which represents the characteristic value write operation, and W800 will return the received content.



The APP displays the received return value:



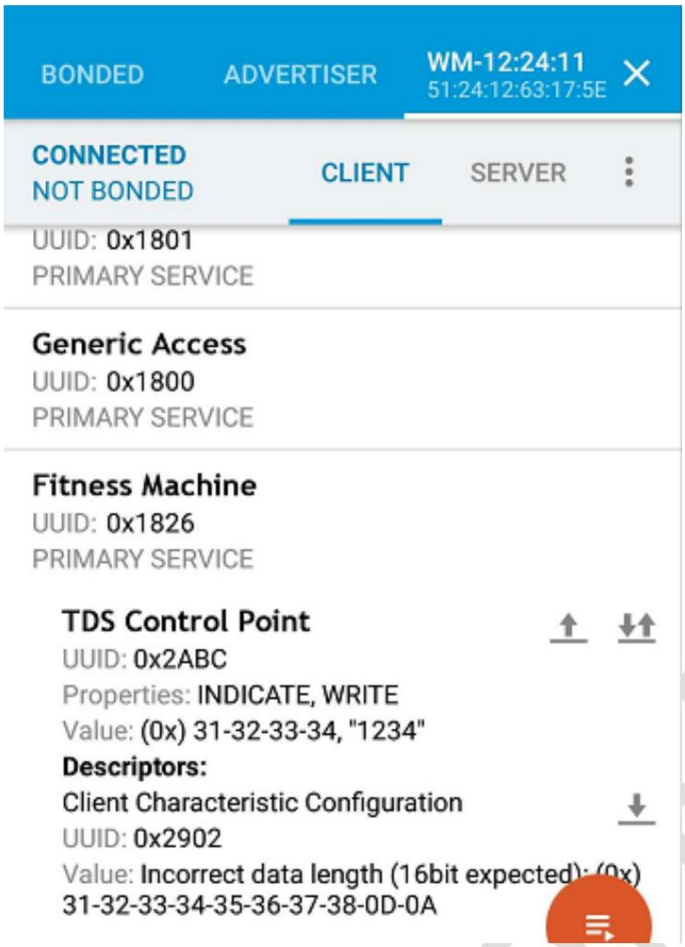
At this time, the received content will be displayed on the W800 side:

```
+OK=0,4,42,1,4,31323334
+OK=0
```

6.3.13 Read the descriptor on the mobile phone side

Click the read operation on the right side of the descriptor, the down arrow means to read the description content, W800 returns 12345678 words

String:



6.3.14 Disconnect from mobile phone

AT+BLESVDIS=4,047EB5A65FCB,4 //Disconnect from the server,

//client_if 4, address 047EB5A65FCB, ID 4

```
+OK=0,4,0
```

6.3.15 Stop service

AT+BLESTPSC=4,40 //Stop the service whose handle is 40

```
+OK=0,4,40
```

6.3.16 Delete service

AT+BLEDELSC=4,40

```
+OK=0,4,40
```

6.3.17 Logout server

AT+BLEDESSV=4 //logout the server whose client_if is 4

```
+OK=0,4
```

6.3.18 Log out of Bluetooth service

AT+BTDES

6.4 BLE client operation example

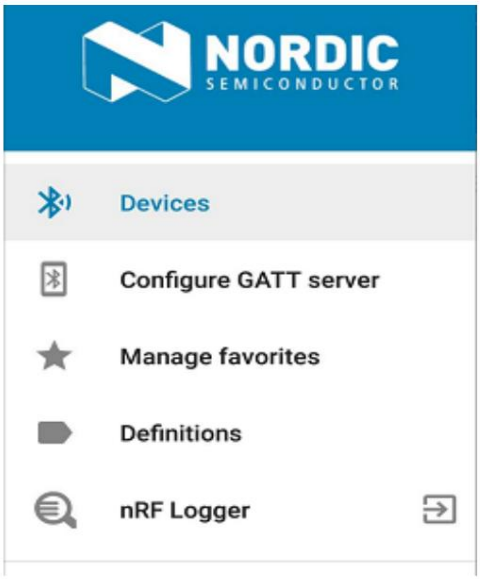
This chapter introduces the step-by-step creation of the BLE Server on the mobile phone, the configuration of the characteristic value is the description, and the start of the broadcast. W800

Perform scanning, connection, and read characteristic value operations. The mobile terminal still uses the Nrf connect APP.

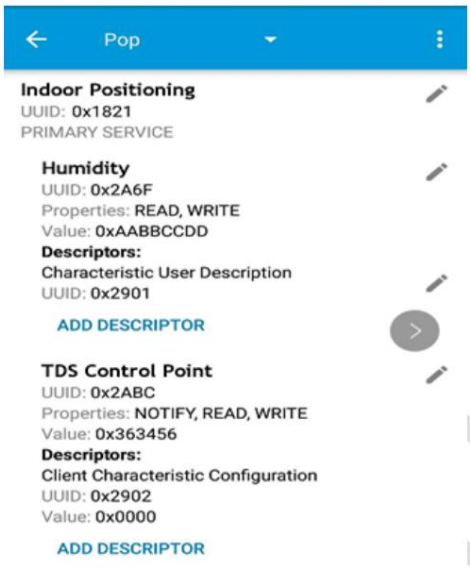
6.4.1 Create a server on the mobile phone

The process of configuring Nrf connect GATT server is shown in the figure below. It is necessary to add a service, configure the characteristic value unique write attribute and attribute

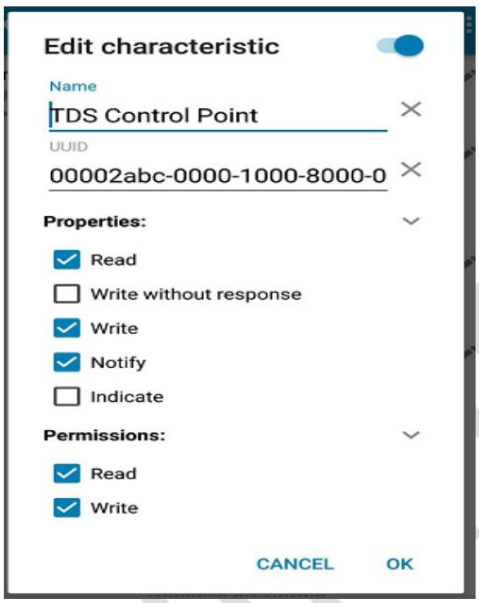
property value and then enable the broadcast function:



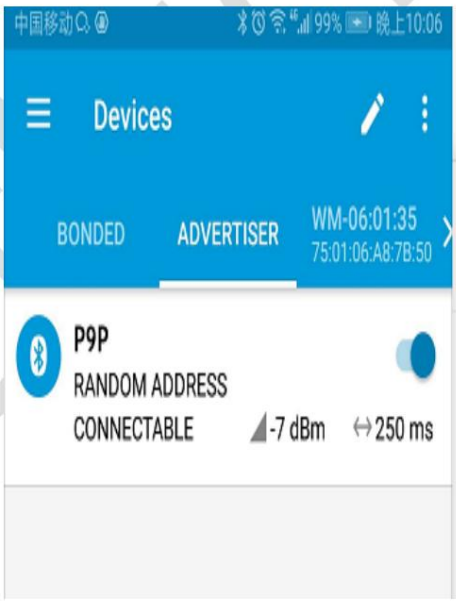
step one



step two



step three



step four

6.4.2 Enable Bluetooth on W800

AT+BTEN=1.0 //Enable the bluetooth system

```
+OK=0,1
```

6.4.3 W800 create client

AT+BLECCT=8888 //Create a client with uuid 8888

```
+OK=0,4
```

6.4.4 W800 start scanning

AT+BLESCAN=1 // start scanning

```
7438B770B0E9,-89,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
7438B770B0E9,-83,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
71C1608D025C,-93,HUAWEI,020102070948554157 4549
71C1608D025C,-87,HUAWEI,020102070948554157 4549
71C1608D025C,-86,HUAWEI,020102070948554157 4549
```

At this point, you can see the device name huawei, which is the broadcast content sent by the mobile phone.

6.4.5 W800 stops scanning

AT+BLESCAN=0 //stop scanning

6.4.6 W800 connect to server

AT+BLECCONN=4,71C1608D025C //connect to mobile phone

Example of a connection error, at which point the connection operation can be repeated.

```
+OK=133,4,0
```

Example of a successful connection:

```
+OK=0,4,4
```

6.4.7 W800 Scanning Service List

AT+BLECSSC=4 //Scan the server's service list

```
+OK=0,4, CMPLT
```

6.4.8 W800 read service list

AT+BLECGDB=4 // read service list

```
+OK=0,4,20
0x1801,T=0x00,HDL=0,PROP=0x00
        0x2a05,T=0x03,HDL=3,PROP=0x20
0x1800,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=22,PROP=0x02
        0x2a01,T=0x03,HDL=24,PROP=0x02
        0x2aa6,T=0x03,HDL=26,PROP=0x02
0xfe35,T=0x00,HDL=0,PROP=0x00
        0x2a00,T=0x03,HDL=42,PROP=0x0a
        0x2a01,T=0x03,HDL=44,PROP=0x30
        0x2902,T=0x04,HDL=45,PROP=0x00
        0x2a02,T=0x03,HDL=47,PROP=0x08
        0x2a03,T=0x03,HDL=49,PROP=0x30
        0x2902,T=0x04,HDL=50,PROP=0x00
0x046a,T=0x00,HDL=0,PROP=0x00
        0x046c,T=0x03,HDL=53,PROP=0x0a
0x1821,T=0x00,HDL=0,PROP=0x00
        0x2a6f,T=0x03,HDL=56,PROP=0x0a
        0x2901,T=0x04,HDL=57,PROP=0x00
        0x2abc,T=0x03,HDL=59,PROP=0x1a
        0x2902,T=0x04,HDL=60,PROP=0x00
```

6.4.9 W800 read characteristic value

AT+BLECACH=1,4,59,0 // Read the handle value of interest. This example reads 59. At this time, the return value is: the length is 3, and the content is a hexadecimal string 363456

```
+OK=0,4,3,363456
```

6.4.10 W800 Disconnect

AT+BLECDIS=4,63573EA5A2F7,4

```
+OK=0,4,4,0
```

6.4.11 W800 logout client

AT+BLECDES=4

```
+OK=0,4
```

6.4.12 W800 log out of Bluetooth service

AT+BTDES

6.5 Switch example server

6.5.1 Enable Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.5.2 Enable demo server

AT+BLEDS=1

6.5.3 Stop demo server

AT+BLEDS=0

6.5.4 Exit the Bluetooth system

AT+BTDES

6.6 switch example client

6.6.1 Enable Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.6.2 Enable example client

AT+BLEDC=1

6.6.3 Stop the example client

AT+BLEDC=0

6.6.4 Exit the Bluetooth system

AT+BTDES

6.7 Switch multi-connection example client

6.7.1 Enable the Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.7.2 Enable multi-connection demo client

AT+BLEDCMC=1

6.7.3 Stop demo client

AT+BLEDCMC=0

6.7.4 Exit the Bluetooth system

AT+BTDES

6.8 Switch UART transparent transmission

6.8.1 Enable the Bluetooth system

AT+BTEN=1.0

```
+OK=0,1
```

6.8.2 Enable UART transparent transmission Server/Client

AT+BLEUM=1,1 //Enable the server side of UART transparent transmission, use UART1 transparent transmission

AT+BLEUM=2,1 //Enable the client end of UART transparent transmission, use UART1 transparent transmission

6.8.3 Stop UART transparent transmission

AT+BLEUM=0,1 //Close UART transparent transmission mode on server side

AT+BLEUM=0,2 //Close UART transparent transmission mode on client side

6.8.4 Exit the Bluetooth system

AT+BTDES

6.9 Example of Traditional Bluetooth Audio Operation

After enabling the Bluetooth function, you can directly operate the AT command. See AT command chapter

6.10 Operation example of traditional Bluetooth hands-free phone

After enabling the Bluetooth function, you can directly operate the AT command. See AT command chapter

6.11 SPP operation example

After enabling the Bluetooth function, you can directly operate the AT command. See AT command chapter

For the SPP server mode, after enabling it, the user can use the Bluetooth serial port and other test tools on the mobile phone to send

The current device can perform data read and write operations.

6.12W800 Test Mode

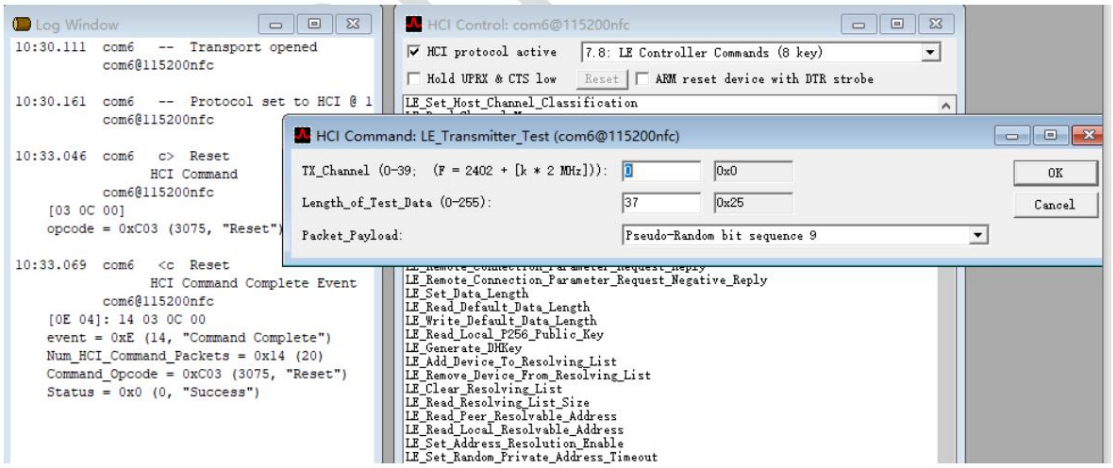
W800 supports real-time access to test mode, which can be used by customers to test RF performance and controller function test and certification

certification test.

6.12.1 W800 enters test mode

AT+BTTEST=1 //Enter the bluetooth test, at this time you can use the test tool directly through the configured uart port

Operate the controller.



6.12.2 W800 Exit Signaling Test

AT+BTTEST=0 //Exit the test mode, at this time the host protocol stack controls the controller.