WM_W800_Bluetooth system architecture and API description

V1.1

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	
V1.0	2021/3/4 [C] Create document	Pengxg	
V1.1	2021/05/17 1. Delete traditional Bluetooth related content 2ÿ AT commands set broadcast content and scan response content. 3ÿ AT command to set broadcast parameters	Pengxg	

Table of contents

2 Introduction

2.1 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.

2.2 Intended audience

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

2.3 Definition of terms

Ordinal term/abbreviation		Description/Definition
1	BT	BlueTooth
2	BECAME	Bluetooth Low Energy
3	HCI	Host Controller Interface
4	GAP	General Access Profile
5	IFS	Inter Frame Space

2.4 References

"W800 Chip Product Specifications"

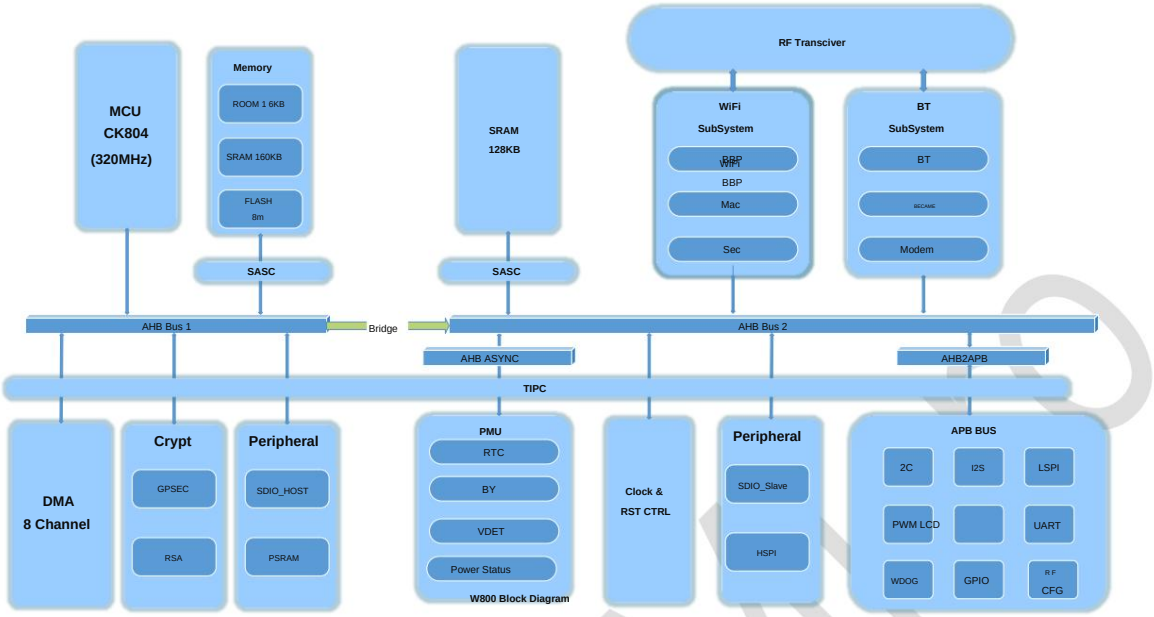
ÿBluetooth Core spec4.0 and 4.2ÿ

"WM_W800_Bluetooth System Architecture and API Description_V1.0"

"Bluetooth controller spec"

3 W800 Bluetooth System

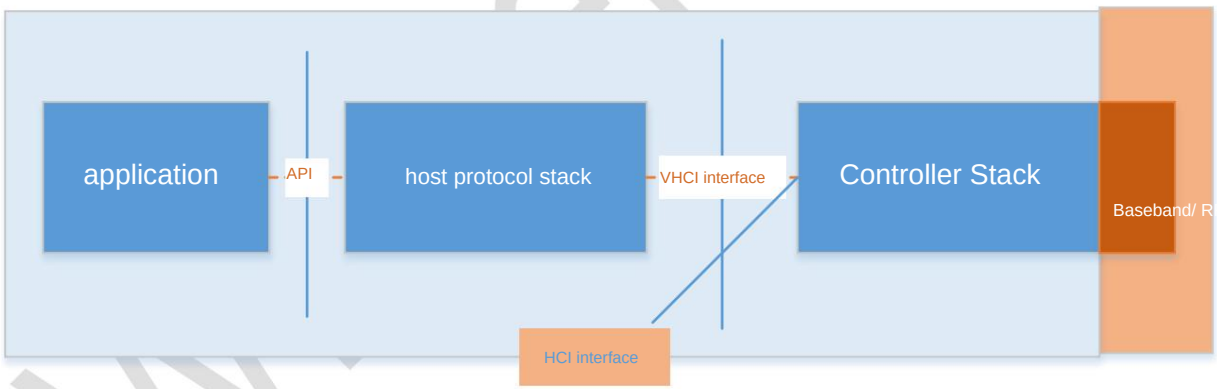
3.1 Chip bluetooth design block diagram



3.2 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.

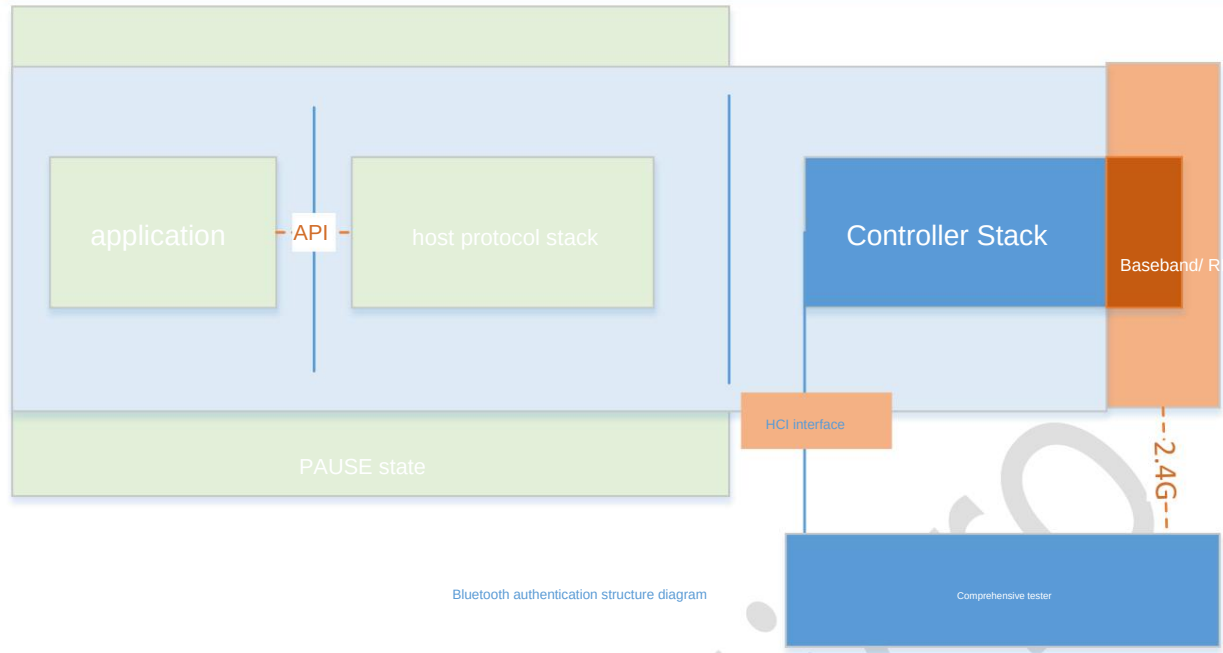
The radio frequency part of Bluetooth is shared with the WiFi system.



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 directly controls the control through the UART port controller. At this time, the host protocol stack is in the freeze state.

3.3 Introduction to NimBLE

3.3.1 NimBLE

NimBLE is an open source Bluetooth 5.0 protocol stack under the Apache Foundation, with complete Host and Controller layers. Occupies less resources, supports Bluetooth 5.0 features, and also supports Mesh and other functions. Based on FreeRTOS and our Controller, the Host layer is transplanted.

3.3.2 NimBLE directory structure

The entire nimble protocol stack contains 4 directories: /docs

folder contains some documentation of the nimble protocol stack, suffixed with .rst /ext folder contains the

encryption library used by the nimble protocol stack/nimble folder contains the entire nimble protocol The stack

code implementation/porting folder contains the related implementation of the W800 platform

3.4 Application Layer Protocol Description

Based on our Controller, the functions supported by the NimBLE protocol stack are as follows: ÿ

Privacy 1.2 (LE Privacy 1.2) ÿ Security Management (SM), support for traditional pairing (LE Legacy Pairing), secure connection (LE Secure Connections), specific key distribution (Transport Specific Key Distribution) ÿ Link layer PDU data length extension (LE Data Length Extension) ÿ Multi-role concurrency (master (central)/slave (peripheral), server/client) ÿ Simultaneous broadcast and scan ÿ Low-speed directional broadcast (Low Duty Cycle Directed Advertising) ÿ Connection parameters request procedure ÿ LE Ping ÿ Complete GATT client, server, and sub-functions ÿ Abstract HCI interface layer

3.4.1 GAP GAP

defines a series of concepts such as roles, modes, and processes. Users need to understand these concepts first, and then configure and use BLE according to the GAP specification according to their own development needs, so as to realize the broadcast of BLE devices. For example, if the user needs to develop an application program for sending and receiving BLE broadcasts, then it is necessary to set the relevant modes defined by GAP to achieve the broadcasting effect. The role description is as follows:

application role	Application Features
Broadcaster	is used to send non-connectable broadcasts and respond to scan requests sent by Observer, and cannot communicate with Observer establishes a connection
Observer	receives the broadcast sent by the Broadcaster, and can choose to send a scan request to the Broadcaster, and receive the scan response
Peripheral	is used to send connectable broadcasts and establish a connection with Central according to the connection request received
Central	Receive a connectable broadcast, send a connection request to Peripheral, and establish a connection

3.4.2 ATT

Connected BLE devices use ATT / GATT specification for application data exchange.

ATT defines the concepts of roles and attributes, which are used to store data

ATT role

ATT role	Application Features
ATT server	The server can define a series of properties for clients to access
ATT client	Clients can use the ATT protocol to discover, read, and write server-defined attributes

The

attribute attribute logic results are as follows

property handle	property type	attribute value	attribute permissions
0x0000- 0xFFFF	UUID	0-N bytes	Read/Write/Indication/Notification

in:

1) The attribute handle is allocated by the attribute server;

2) The attribute type is defined by the user or specified by a higher-level specification;

3) The attribute value is defined by the user or specified by a higher-level specification, and is used to save application data;

4) Attribute permissions are defined by the user or specified by a higher-
level specification. The attribute access method-ATT protocol frame
attribute access method is also the ATT protocol frame, which is called ATT PDU (protocol data unit) in the Bluetooth specification.

ATT PDU is used by ATT client to discover, read and write attributes, or used by ATT server to send notification and indication of attributes.

There are 6 types of ATT PDU as follows:

ATT PDU type	describe
Commands	ATT PDU sent by the client to the server, the server will not send a response
Requests are	ATT PDUs sent by the client to the attribute server, and the server will send a response as a response
Response	The server sends the client as a response to the request
Notification is sent from	the server to the client, and the client will not send confirmation as a response
Indication	Sent by the server to the client, the client needs to send confirmation as a response
Confirmation	Sent by the client to the server as a response to Indication

3.4.3 GATT

GATT is for applications or other configuration files so that ATT clients can communicate with ATT servers.

GATT defines the framework for using the ATT protocol PDU. This framework defines the data exchange process and also defines the application data exchange format: service (service) and characteristics (characteristics). Through GATT we can discover services, and read/write or configure the characteristics of peer devices.

GATT roles

are the same as ATT, and GATT also has two roles:

GATT Role	Role Description

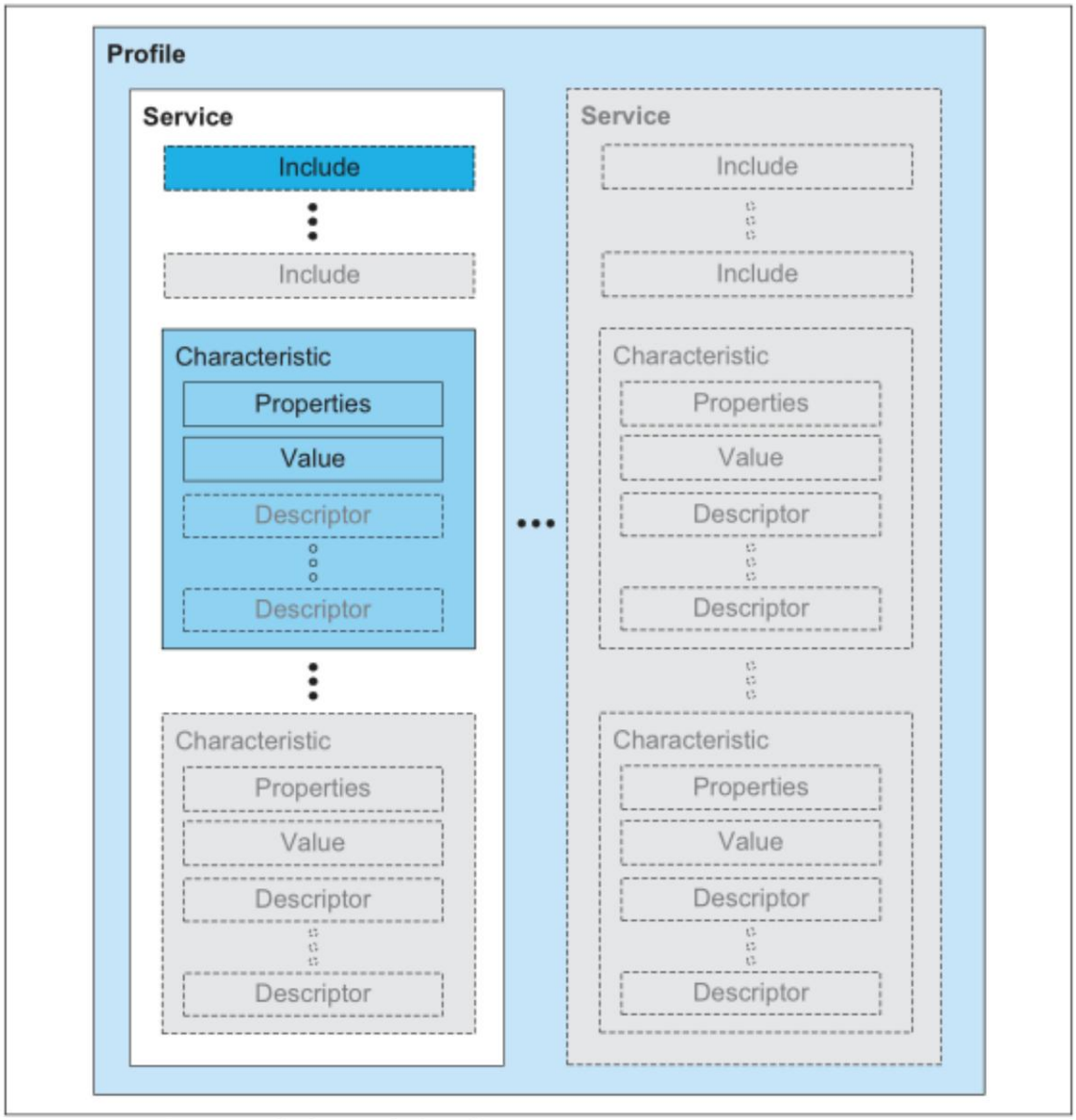
GATT server defines services and characteristics of BLE devices	
GATT client sends data requests to access services and characteristics of BLE devices	

The GATT role is not fixed, only when the corresponding process is started, the GATT role is determined, and the GATT role is released when the process ends. Among them: GATT client sends commands and requests to server, and can receive response, indications and notifications from server; GATT server receives commands and requests from client and sends response, indication and notification to client.

GATT Data Structure

The GATT configuration file specifies the structure of the data exchange. This structure defines the basic elements: service (service) and characteristics (characteristics). All services and characteristics are contained in attributes, which are containers for GATT data.

The GATT data structure is shown in the following figure:

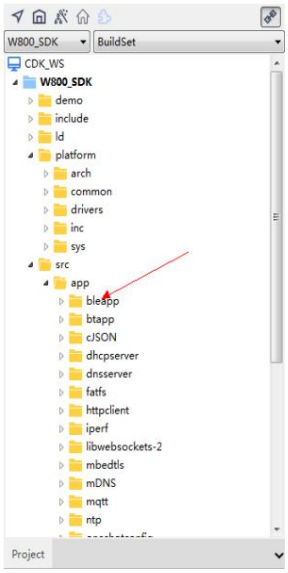


Description of GATT data structure:

1. The top layer is a profile, which can be understood as an application, which consists of one or more service composition;
2. Each service is composed of characteristic definition and service reference;
3. A feature contains a feature value and other information related to the feature value;
4. Both services and characteristics are stored by the GATT server in the form of attributes.

3.5 Sample Code Framework Description

3.5.1 Bluetooth system software code location



The bleapp directory is the bluetooth sample code, users can refer to or make 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_gap.c	GAP implementation and reporting processing of related events
4	wm_ble_server_wifi_prof.c	BLE auxiliary distribution network service communication module, responsible for the implementation of the transport layer
5	wm_ble_server_wifi_app.c	BLE auxiliary distribution network application protocol processing module, responsible for the realization of the application layer protocol Realize api
6	wm_ble_client_api_demo.c	to create demo server function Realize api to create demo client function
7	wm_ble_server_api_demo.c	
8	wm_ble_client_api_multi_conn_demo.c implement api to create demo client, which can support connection Connect 7 demo servers.	
9	wm_ble_uart_if.c	Example of implementing BLE-based UART transparent transmission

4 API Description

4.1 Bluetooth system API

No API name		describe
1	int tls_bt_init(uint8_t uart_idx)	Running the Bluetooth system, this function will enable the host protocol in turn and controller protocol stack.

2	int tls_bt_deinit(void)	Stop the Bluetooth system, and modify the function to cancel the host protocol stack and controller protocol stack in turn.

4.2 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, allocate memory and create tasks, etc.
2	tls_bt_status_t tls_bt_ctrl_disable(void); logout controller protocol 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_ctrl_status_t tls_bt_controller_get_status(void);	Read the current state of the controller,
6	bool wm_bt_vuart_host_check_send_available(void);	Used to determine whether the host can send instructions to the controller make
7	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
8	tls_bt_status_t tls_bt_ctrl_if_register (const tls_bt_host_if_t *p_host_if);	Register the controller data sending interface, that is, the host protocol stack receiving data interface
9	tls_bt_status_t tls_bt_ctrl_sleep (bool enable);	Whether to run the controller to enter on idle sleep mode
10 bool	tls_bt_ctrl_is_sleep (void);	Reads whether the controller is in sleep mode
11 tls_bt_status_t tls_bt_ctrl_wakeup(void) exit sleep mode		
12 tls_bt_status_t	enable_bt_test_mode(tls_bt_hci_if_t *p_hci_if)	Enter Bluetooth test mode
13 tls_bt_status_t exit_bt_test_mode()		Exit Bluetooth test mode

4.3 Application layer protocol API

4.3.1 GAP

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

4.3.1.1 GAP API description

No AP	name	describe
1.	int tls_ble_gap_init(void);	Initialize the default broadcast and scan parameters; set the device name. Note: This function is called automatically when the Bluetooth system is running.
2.	int tls_ble_gap_deinit(void);	Release resources. Note: This function is automatically called when the Bluetooth system logs out
3.	int tls_ble_gap_set_adv_param(uint8_t adv_type, uint32_t min, uint32_t max, uint8_t chn_map, uint8_t filter_policy,uint8_t *dir_mac,uint8_t dir_mac_type)	Set broadcast parameters
4.	int tls_nimble_gap_adv(wm_ble_adv_type_t type, int duration);	start, stop broadcasting
5.	int tls_ble_gap_scan(wm_ble_scan_type_t type, bool filter_duplicate);	Start and stop scanning
6.	int tls_ble_gap_set_scan_param(uint32_t intv, uint32_t window, uint8_t filter_policy, bool limited, bool passive, bool filter_duplicate);	Set scan parameters
6	int tls_ble_gap_set_name(const char *dev_name,uint8_t update_flash);	set device name Note: If the device is broadcasting and the broadcast parameter struct ble_hs_adv_fields Specify name_is_complete. After setting the name, the wide After broadcasting needs to be stopped and enabled again, it will take effect.
7	int tls_ble_gap_get_name(char *dev_name);	Read device name. Note: This function first reads the device name saved in Flash name, if not present, read the device name in ram
8	int tls_ble_gap_set_data(wm_ble_gap_data_t type, uint8_t *data, int data_len);	Used to set custom broadcast data or scan response Allow

9	int tls_ble_register_gap_evt(uint32_t evt_type, app_gap_evt_cback_t *evt_cback);	Reporting function for registering GAP events
10	int tls_ble_deregister_gap_evt(uint32_t evt_type, app_gap_evt_cback_t *evt_cback);	Reporting function for unregistering GAP events

4.3.2 BLE server

BLE server assumes the role of GATT server. The wm_ble_server_api_demo module provides an example of user program development. The example function is described as: 1. Create the following service list function and start broadcasting;

```
#define WM_GATT_SVC_UUID       0xFFF0
#define WM_GATT_INDICATE_UUID  0xFFF1
#define WM_GATT_WRITE_UUID     0xFFF2


static const struct ble_gatt_svc_def gatt_demo_svr_svcs[] = {
    {
        /* Service: uart */
        .type = BLE_GATT_SVC_TYPE_PRIMARY,
        .uuid = BLE_UUID16_DECLARE(WM_GATT_SVC_UUID),
        .characteristics = (struct ble_gatt_chr_def[]) { {
                .uuid = BLE_UUID16_DECLARE(WM_GATT_WRITE_UUID),
                .val_handle = &g_ble_demo_attr_write_handle,
                .access_cb = gatt_svr_chr_demo_access_func,
                .flags = BLE_GATT_CHR_F_WRITE,
            },{
                .uuid = BLE_UUID16_DECLARE(WM_GATT_INDICATE_UUID),
                .val_handle = &g_ble_demo_attr_indicate_handle,
                .access_cb = gatt_svr_chr_demo_access_func,
                .flags = BLE_GATT_CHR_F_INDICATE,
            },{
              0, /* No more characteristics in this service */
            }
         },
    },

    {
        0, /* No more services */
    },
};
```

2. After receiving the other party's connection, update the ATT layer MTU

function; 3. After receiving the other party's connection, if the other party's indication function is received, continue to send specific data to the other party.

This module provides two external APIs for initialization and logout respectively. The specific codes are as follows:

```
int tls_ble_server_demo_api_init(tls_ble_output_func_ptr output_func_ptr)
{
    int rc = BLE_HS_EAPP;

    if(bt_adapter_state == WM_BT_STATE_OFF)
    {
        TLS_BT_APPL_TRACE_ERROR("%s failed rc=%s\r\n", __FUNCTION__, tls_bt_rc_2_str(BLE_HS_EDISABLED));
        return BLE_HS_EDISABLED;
    }

    TLS_BT_APPL_TRACE_DEBUG("%s, state=%d\r\n", __FUNCTION__, g_ble_server_state);

    if(g_ble_server_state == BLE_SERVER_MODE_IDLE)
    {
        g_ble_demo_prof_connected = 0;

        //step 0: reset other services. Note
        rc = ble_gatts_reset();
        if(rc != 0)
        {
            TLS_BT_APPL_TRACE_ERROR("tls_ble_server_demo_api_init failed rc=%d\r\n", rc);
            return rc;
        }

        //step 1: config/adding the services
        rc = wm_ble_server_demo_gatt_svr_init();

        if(rc == 0)
        {
            tls_ble_register_gap_evt(WM_BLE_GAP_EVENT_CONNECT|WM_BLE_GAP_EVENT_DISCONNECT|WM_BLE_GAP_EVENT_NOTIFY_T>
            TLS_BT_APPL_TRACE_DEBUG("### wm_ble_server_api_demo_init \r\n");

            g_ble_uart_output_fptr = output_func_ptr;
            /*step 2: start the service*/
            rc = ble_gatts_start();
            assert(rc == 0);

            /*step 3: start advertisement*/
            rc = wm_ble_server_api_demo_adv(true);

            if(rc == 0)
            {
                g_ble_server_state = BLE_SERVER_MODE_ADVERTISING;
            }
        }else
        {
            TLS_BT_APPL_TRACE_ERROR("### wm_ble_server_api_demo_init failed(rc=%d)\r\n", rc);
        }
    } ? end if g_ble_server_state==B... ?
    else
    {
        TLS_BT_APPL_TRACE_WARNING("wm_ble_server_api_demo_init registered\r\n");
        rc = BLE_HS_EALREADY;
    }
```

```
int tls_ble_server_demo_api_deinit()
{
   int rc = BLE_HS_EAPP;

    if(bt_adapter_state == WM_BT_STATE_OFF)
    {
        TLS_BT_APPL_TRACE_ERROR("%s failed rc=%s\r\n", __FUNCTION__, tls_bt_rc_2_str(BLE_HS_EDISABLED));
        return BLE_HS_EDISABLED;
    }

   TLS_BT_APPL_TRACE_DEBUG("%s, state=%d\r\n", __FUNCTION__, g_ble_server_state);

   if(g_ble_server_state == BLE_SERVER_MODE_CONNECTED || g_ble_server_state == BLE_SERVER_MODE_INDICATING)
   {
        g_ble_demo_indicate_enable = 0;

        rc = ble_gap_terminate(g_ble_demo_conn_handle, BLE_ERR_REM_USER_CONN_TERM);
        if(rc == 0)
        {
           g_ble_server_state = BLE_SERVER_MODE_EXITING;
        }
   }else if(g_ble_server_state == BLE_SERVER_MODE_ADVERTISING)
   {
        rc = tls_nimble_gap_adv(WM_BLE_ADV_STOP, 0);
        if(rc == 0)
        {
            if(g_ble_uart_output_fptr)
            {
                g_ble_uart_output_fptr = NULL;
            }
            g_send_pending = 0;
            g_ble_server_state = BLE_SERVER_MODE_IDLE;
        }
   }else if(g_ble_server_state == BLE_SERVER_MODE_IDLE)
   {
        rc = 0;
   }else
   {
        rc = BLE_HS_EALREADY;
   }

   return rc;
} ? end tls_ble_server_demo_api_deinit ?
```

4.3.2.1 BLE server API Description

The NimBLE protocol stack does not support the function of dynamically adding or canceling the service when the GATT service is running. Therefore, the GATT service must

The service function can only be enabled after the configuration is completed.

No	API name	describe
1	int ble_gatts_reset(void)ÿ	Reset the GATT service list and release resources.
2	int ble_gatts_count_cfg(const struct ble_gatt_svc_def *defs)ÿ	Configure the GATT service
3	int ble_gatts_add_svcs(const struct ble_gatt_svc_def *svcs)	Add GATT service
4	int ble_gatts_start(void)	Start the GATT server
5	int ble_gattc_indicate_custom(uint16_t conn_handle, uint16_t chr_val_handle, struct os_mbuf *int)	To a certain conn_handle through the specified attr_handle Send indication data
6	int ble_gattc_notify_custom(uint16_t conn_handle, uint16_t chr_val_handle, struct os_mbuf *int)	To a certain conn_handle through the specified attr_handle Send notification data

4.3.3 BLE client

BLE client assumes the role of GATT client, that is, actively initiates scanning, connection, communication and other applications.

The wm_ble_client_api_demo module provides the following sample functions: 1. Initiate a scan; 2. Initiate a connection

according to whether the broadcast data contains the service field of FFFO;

3. After the connection is established, read the service list of

the other party; 4. Analyze the service list, determine whether the characteristics contain the FFF1 field, and enable the indication,

Print after receiving indication data

5. Separate the servcie list, judge whether the characteristics contains the FFF2 field, and send 0Xaa, 0xbb characters

Section to each

other. With reference to this module implementation, users can develop their own applications.

4.3.3.1 BLE client API Description

No	API name	describe

1.	int ble_gap_connect(uint8_t own_addr_type, const ble_addr_t *peer_addr, int32_t duration_ms, const struct ble_gap_conn_params *conn_params, ble_gap_event_fn *cb, void *cb_arg)	Used to establish a BLE connection with the other device
2.	int ble_gattc_disc_all_svcs(uint16_t conn_handle, ble_gatt_disc_svc_fn *cb, void *cb_arg)	After the connection is established, read the server side service list
3	int ble_gattc_exchange_person(uint16_t conn_handle, ble_gatt_mtu_fn *cb, void *cb_arg)	After the connection is established, it is used to communicate with the other party Mutual ATT layer MTU function
4	int ble_gattc_write_flat(uint16_t conn_handle, uint16_t attr_handle, const void *data, uint16_t data_len, ble_gatt_attr_fn *cb, void *cb_arg)	Used to send data to the specified conn_handle and attr_handle
5	int ble_gattc_read(uint16_t conn_handle, uint16_t attr_handle, ble_gatt_attr_fn *cb, void *cb_arg)	Used to initiate a read operation to the specified conn_handle and attr_handle

4.4 Bluetooth assisted WiFi distribution network API

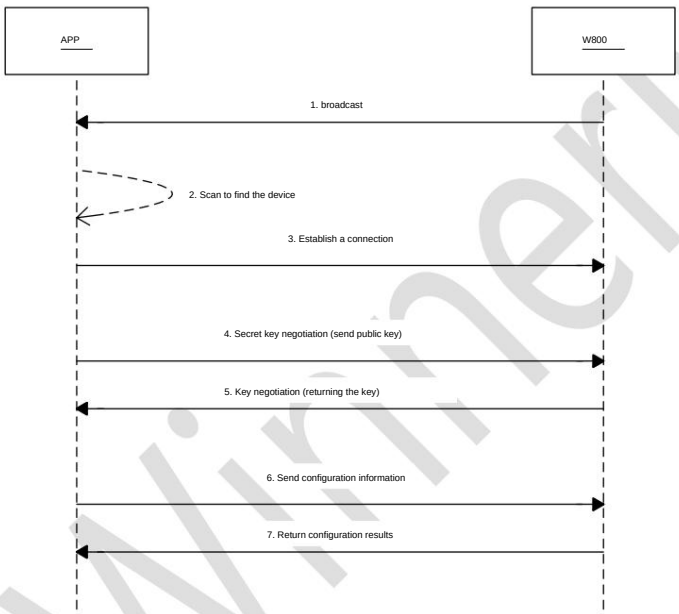
BLE assisted WiFi distribution network, as a specific application of BLE server. wm_ble_server_wifi_prof implements the function of BLE profile, responsible for data transmission and processing, and wm_ble_server_wifi_cfg handles specific communication protocol processing. Such a hierarchical structure makes the application process independent of the specific transport layer, and the logic level call is clearer.

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

No API name	describe

1	tls_wifi_set_oneshot_flag(flag) flag 0: closed oneshot 1: UDPÿbroadcast+multicastÿ 2: AP+socket 3: AP+WEBSERVER 4: BT	When the flag is set to 4, it starts/stops the BLE assisted WiFi distribution network (the Bluetooth system needs to be enabled before using the module). Note: 1. After the network distribution is successful, the BLE distribution network service will automatically exit and the broadcast will be turned off. If you need to configure the network again, please call this API again. 2. If the network distribution fails, the user can secondary configuration

4.4.1 Example of application process



4.4.2 Auxiliary WiFi distribution network Service

definition 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

4.5 Users realize their own distribution network service

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

5 API usage examples

The W800 Bluetooth function is disabled by default after the device is reset. If the user wants to use Bluetooth by default, please refer to the following instructions.

5.1 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;

```
/*This function is called at wm_main.c*/
void tls_bt_entry()
{
  //tls_bt_init(0x01);    //enable it if you want to turn on bluetooth after system booting
}

void tls_bt_exit()
{
  //tls_bt_deinit();      //enable it if you want to turn off bluetooth when system reseting;
}
```

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 here;

```
static void app_adapter_state_changed_callback(tls_bt_state_t 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");

        //at here , user run their own applications;
        #if 1
        //tls_ble_wifi_cfg_init();
        //tls_ble_server_demo_api_init(NULL);
        //tls_ble_client_demo_api_init(NULL);
        //tls_ble_client_multi_conn_demo_api_init();
        #endif

    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");

        //here, user may free their application;
        #if 1
        tls_ble_wifi_cfg_deinit(2);
        tls_ble_server_demo_api_deinit();
        tls_ble_client_demo_api_deinit();
        tls_ble_client_multi_conn_demo_api_deinit();
        #endif
    }

    #endif

} ? end app_adapter_state_changed_callback ?
```

5.2 Start up and run (exit) the sample server

At the position marked in step 2 in section 4.1, call wm_ble_server_demo_api_init(); at the position marked in step 2 in section 4.1, call wm_ble_server_demo_api_deinit(); the exit function of the application will be released automatically when the Bluetooth system exits. Of course, when the Bluetooth system is running, the user can also exit his own application program at any time.

5.3 Start up and run (exit) the sample client

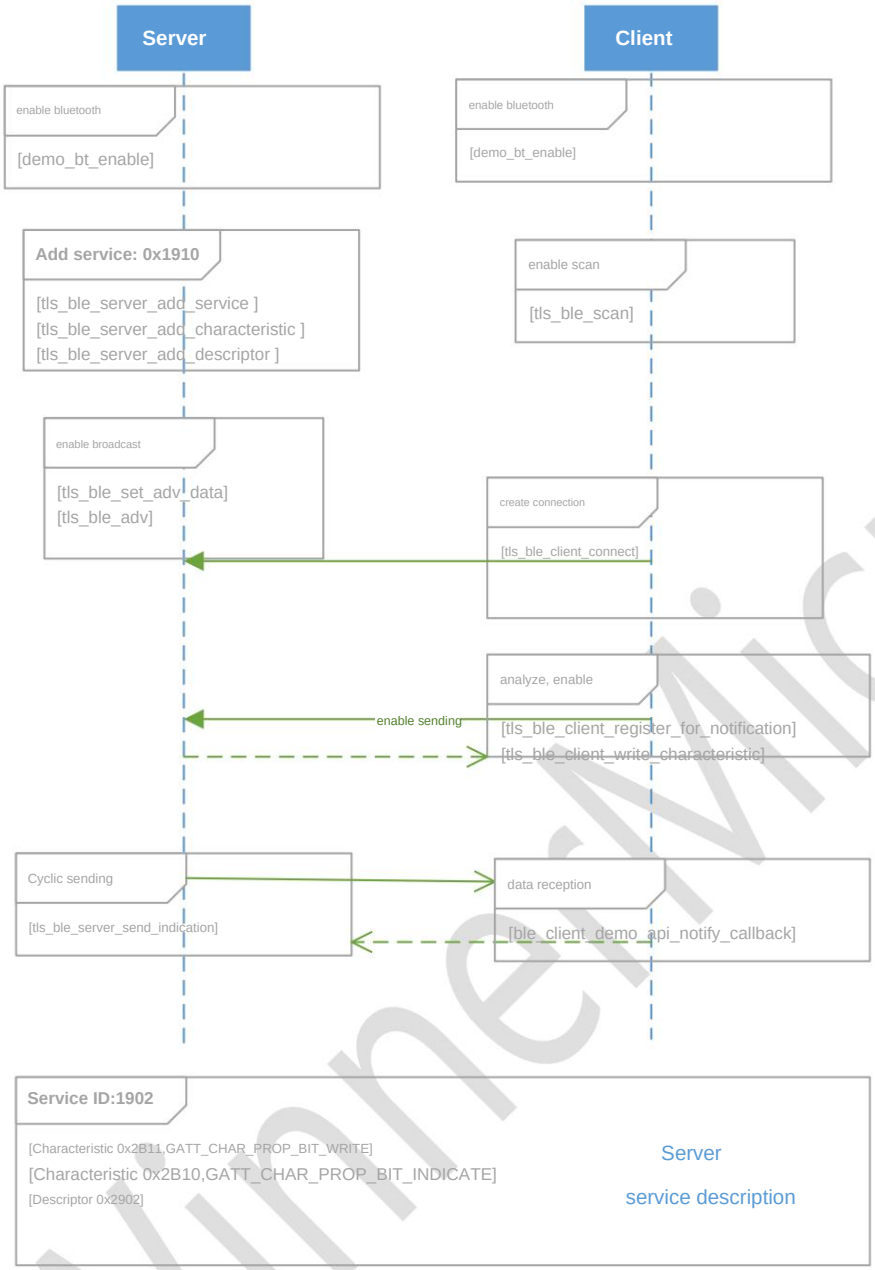
At the position marked in step 2 in section 4.1, call wm_ble_client_demo_api_init(); at the position marked in step 2 in section 4.1, call wm_ble_client_demo_api_deinit(); the exit function of the application will be released automatically when the Bluetooth system exits. Of course, when the Bluetooth system is running, the user can also exit his own application program at any time.

5.4 Run multi-connection (exit) example client on startup

At the position marked in step 2 in section 4.1, call wm_ble_client_multi_conn_demo_api_init(); at the position marked in step 2 in section 4.1, call wm_ble_client_multi_conn_demo_api_deinit();
The exit function of the application will be released automatically when the Bluetooth system exits. Of course, when the Bluetooth system is running, the user can also exit his own application program at any time.

5.5 Data exchange function

Use two demo boards to run 4.2 server demo and 4.3 client demo respectively. For specific demo functions, refer to See descriptions in 3.3.2 and 3.3.3. After the connection is successful, the server will continuously send data to the client in the form of indication, as shown in the sequence diagram as shown below:



5.6 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 respectively. Refer to 5.2 for configuration mode. 2. Run 1 BLE client that supports multi-connection function. Refer to 5.4 for configuration mode.

At this point, the client will initiate scanning and connection functions in sequence until the connection to 7 BLE servers is successful.

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

```
static void wm_ble_update_conn_params(struct ble_gap_conn_params *conn_params)
{
    int i = 0;

    for(i = 0; i<MAX_CONN_DEVCIE_COUNT; i++)
    {
        if(conn_devices[i].conn_state == DEV_DISCONNCTED)
        {
            conn_params->itvl_min = 0x20 + i*16;
            conn_params->itvl_max = 0x22 + i*16;
            return;
        }
    }
}
```

5.7 UART transparent transmission function

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

The example 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); 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); After startup, the server starts broadcasting.

After the client scans the broadcast, it connects to the server and analyzes the server

end service list, and after matching, the BLE channel is established. Users can transmit data through UART1.

5.8 Turn on the broadcast

Step 1, call to open the Bluetooth function in the tls_bt_entry() function;

```
/*This function is called at wm_main.c*/
void tls_bt_entry()
{
  //tls_bt_init(0x01);   //enable it if you want to turn on bluetooth after system booting
}

void tls_bt_exit()
{
  //tls_bt_deinit();     //enable it if you want to turn off bluetooth when system reseting;
}
```

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

tls_ble_demo_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 basicly*/
        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 ?
```

5.8.1 Default broadcast data configuration

```
int tls_ble_wifi_adv(bool enable)
{
    int rc;

    if(enable)
    {
        uint8_t own_addr_type;
        struct ble_gap_adv_params adv_params;
        struct ble_hs_adv_fields fields;
        const char *name;
        uint8_t adv_ff_data[] = {0x0C, 0x07, 0x00, 0x10};
        /**
     * Set the advertisement data included in our advertisements:
     *    o Flags (indicates advertisement type and other general info).
     *    o Device name.
     *    o user specific field (winner micro).
     */

        memset(&fields, 0, sizeof fields);

        /* Advertise two flags:
     *    o Discoverability in forthcoming advertisement (general)
     *    o BLE-only (BR/EDR unsupported).
     */
        fields.flags = BLE_HS_ADV_F_DISC_GEN |
                       BLE_HS_ADV_F_BREDR_UNSUP;

        name = ble_svc_gap_device_name();
        fields.name = (uint8_t *)name;
        fields.name_len = strlen(name);
        fields.name_is_complete = 1;

        fields.mfg_data = adv_ff_data;
        fields.mfg_data_len = 4;

        rc = ble_gap_adv_set_fields(&fields);
        if (rc != 0) {
            MODLOG_DFLT(INFO,"error setting advertisement data; rc=%d\r\n", rc);
            return rc;
        }


        MODLOG_DFLT(INFO, "Starting advertising\r\n");

        /* As own address type we use hard-coded value, because we generate
        NRPA and by definition it's random */
        rc = tls_ble_gap_adv(WM_BLE_ADV_IND);
        assert(rc == 0);
    } ? end if enable ? else
    {
        MODLOG_DFLT(INFO, "Stop advertising\r\n");
        rc = ble_gap_adv_stop();
    }
    return rc;
} ? end tls_ble_wifi_adv ?
```

5.8.2 User-defined broadcast data settings

```
int tls_ble_demo_adv(uint8_t type)
{
    int rc = 0;
    TLS_BT_APPL_TRACE_DEBUG("### %s type=%d\r\n", __FUNCTION__, type);

    if(bt_adapter_state == WM_BT_STATE_OFF)
    {
        TLS_BT_APPL_TRACE_ERROR("%s failed rc=%s\r\n", __FUNCTION__, tls_bt_rc_2_str(BLE_HS_EDISABLED));
        return BLE_HS_EDISABLED;
    }
    if(type)
    {
        uint8_t bt_mac[6] = {0};
        uint8_t adv_data[] = {
               0x0C,0x09, 'W', 'M', '-', '0', '0', '0', '0', '0','0','0', '0',
               0x02,0x01,0x05,
               0x03,0x19,0xc1, 0x03};
        extern int tls_get_bt_mac_addr(uint8_t *mac);

        tls_get_bt_mac_addr(bt_mac);
        sprintf(adv_data+5,"%02X:%02X:%02X",bt_mac[3], bt_mac[4], bt_mac[5]);
        adv_data[13] = 0x02;   //byte 13 was overwritten to zero by sprintf; recover it;
        rc = tls_ble_gap_set_data(WM_BLE_ADV_DATA, adv_data, 20);
        switch(type)
        {
            case 1:
                rc = tls_ble_gap_adv(WM_BLE_ADV_IND);
                break;
            case 2:
                rc = tls_ble_gap_adv(WM_BLE_ADV_NONCONN_IND);
                break;
            default:
                /*AT/ DEMO cmd only support adv_ind and adv_nonconn_ind mode*/
                return BLE_HS_EINVAL;
        }

    } ? end if type ? else
    {
        rc = tls_ble_gap_adv(WM_BLE_ADV_STOP);
    }

    return rc;
} ? end tls_ble_demo_adv ?
```

5.9 Turn on the scan

Step 1, call to open the Bluetooth function in the tls_bt_entry() function;

```
/*This function is called at wm_main.c*/
void tls_bt_entry()
{
  //tls_bt_init(0x01);    //enable it if you want to turn on bluetooth after system booting
}

void tls_bt_exit()
{
  //tls_bt_deinit();      //enable it if you want to turn off bluetooth when system reseting;
}
```

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

```
static void app_adapter_state_changed_callback(tls_bt_state_t 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");

        //at here , user run their own applications;
        #if 1
        //tls_ble_wifi_cfg_init();
        //tls_ble_server_demo_api_init(NULL);
        //tls_ble_client_demo_api_init(NULL);
        //tls_ble_client_multi_conn_demo_api_init();
        tls_ble_demo_scan(1);
        #endif

    }else
    {
        TLS_BT_APPL_TRACE_VERBOSE("deinit base application\r\n");

        //here, user may free their application;
        #if 1
        tls_ble_wifi_cfg_deinit(2);
        tls_ble_server_demo_api_deinit();
        tls_ble_client_demo_api_deinit();
        tls_ble_client_multi_conn_demo_api_deinit();
        #endif
    }

    #endif

} ? end app_adapter_state_changed_callback ?
```

```
static int
ble_gap_evt_cb(struct ble_gap_event *event, void *arg)
{
    struct ble_gap_conn_desc desc;
    struct ble_hs_adv_fields fields;
    int rc = 0;

    switch (event->type) {
    case BLE_GAP_EVENT_DISC:
        rc = ble_hs_adv_parse_fields(&fields, event->disc.data,
                                     event->disc.length_data);
        if (rc != 0) {
            return 0;
        }
        /* An advertisment report was received during GAP discovery. */
        print_adv_fields(&fields);
        return 0;
    case BLE_GAP_EVENT_DISC_COMPLETE:
        break;
    default:
        break;
    }

    return rc;
} ? end ble_gap_evt_cb ?


/**
 * Called                1) AT cmd; 2)demo show;
 *
 * @param type           0: scan stop; 1: scan start, default passive;
 *
 *
 * @return               0 on success; nonzero on failure.
 */
int tls_ble_demo_scan(uint8_t type)
{
    int rc;

    TLS_BT_APPL_TRACE_DEBUG("### %s type=%d\r\n", __FUNCTION__, type);

    if(bt_adapter_state == WM_BT_STATE_OFF)
    {
        TLS_BT_APPL_TRACE_ERROR("%s failed rc=%s\r\n", __FUNCTION__, tls_bt_rc_2_str(BLE_HS_EDISABLED));
        return BLE_HS_EDISABLED;
    }
    if(type)
    {
        tls_ble_register_gap_evt(WM_BLE_GAP_EVENT_DISC|WM_BLE_GAP_EVENT_DISC_COMPLETE, ble_gap_evt_cb);
        rc = tls_ble_gap_scan(WM_BLE_SCAN_PASSIVE, false);
    }else
    {
        rc = tls_ble_gap_scan(WM_BLE_SCAN_STOP, false);
        tls_ble_deregister_gap_evt(WM_BLE_GAP_EVENT_DISC|WM_BLE_GAP_EVENT_DISC_COMPLETE,ble_gap_evt_cb );
    }

    return rc;
} ? end tls_ble_demo_scan ?
```

5.10 Open broadcast/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. ; 5.10.1 In the connection state of Slave mode Step 2, in Slave mode, see Section 4.2. Run the demo example of the Ble server. After running, the mobile phone 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.

5.10.1.1 Turn on broadcast Step

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

Call tls_ble_gap_set_adv_param to set the broadcast type to non-connectable broadcast Call tls_nimble_gap_adv to start

broadcasting 5.10.1.2 Start scanning Step 4 Refer to 4.4, just call the scanning API directly.

demo_ble_scan(1); 5.10.2 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) It can scan and operate;

2) Unconnectable broadcast operations can be sent

6 Bluetooth AT commands

6.1 Brief description of Bluetooth AT commands

The Bluetooth system can be controlled by AT commands, and the Bluetooth AT commands are divided into 4 categories. The host and controller are used to configure the main

The machine protocol stack and the 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 part includes 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

6.2 Bluetooth system AT command

6.2.1.1 AT+BTEN

Function:

Enable the Bluetooth system.

Format (ASCII):

AT+BTEN=<uart_no>,<log_level><CR>
+OK=<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	

6.2.1.2 AT+BTDES

Function:

Stop and log off the Bluetooth system.

Format (ASCII):

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

parameter:

See BTEN parameter description

6.3 Bluetooth controller protocol stack AT command

6.3.1.1 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),

6.3.1.2 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

6.3.1.3 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

6.3.1.4 AT+BLETPG

Function:

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

Format (ASCII):

AT+BLETPG=?<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

6.3.1.5 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

6.4 Bluetooth application layer AT command

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

6.4.1 Device management AT commands

6.4.1.1 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

6.4.1.2 AT+BLEADATA

Function:

Configure BLE broadcast content.

Format (ASCII):

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

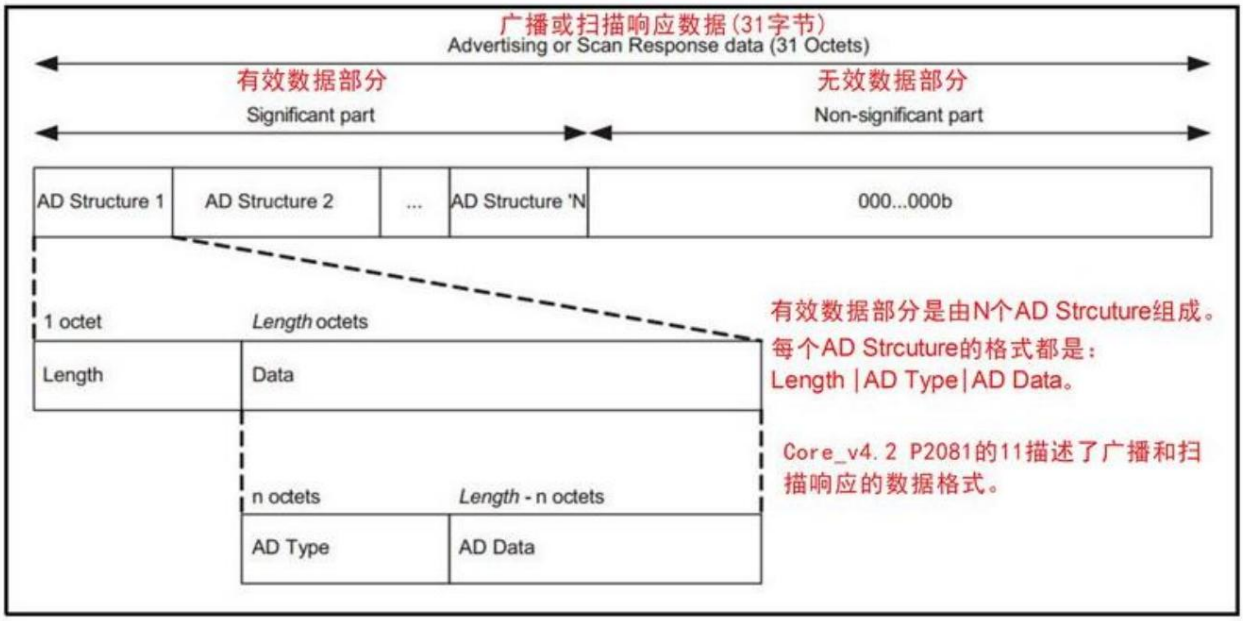
parameter:

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

For example, if the broadcast data is set to 0x02 0x01 0x06 0x03 0x09 0x31 0x32, then the setting command is:

AT+BLEADVDATA=02010603093132. For the specific definition of the broadcast data format, see the description of the

response core specification.



6.4.1.3 AT+BLEAPRM

Function: Configure BLE broadcast parameters.

Format (ASCII):

AT+BLEAPRM=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_m ap>,[adv_filter_policy],[peer_addr_type],[peer_addr]<CR> +OK=<adv_int_min>,<adv_int_max>,<adv_type>,<own_addr_type>,<channel_map>,<ad v_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.

6.4.1.4 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 interval is equal to windows, it means that the controller is always in Scanning status, that is, the scanning window is always open.

6.4.1.5 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:

```
484661B4A304,-93,HUAWEI,020102070948554157 4549
484661B4A304,-93,HUAWEI,020102070948554157 4549
484661B4A304,-97,HUAWEI,020102070948554157 4549
484661B4A304,-90,HUAWEI,020102070948554157 4549
7438B770B0E9,-83,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
6130DE163F82,-103,02011A020A0C0AFF4C001005511C041B92
6130DE163F82,-102,02011A020A0C0AFF4C001005511C041B92
484661B4A304,-91,HUAWEI,020102070948554157 4549
7438B770B0E9,-85,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
7438B770B0E9,-88,TS300 serie,0201060C0854533330302073657269651106 22A8FF2F49D8FFFF0100000000000000
```

6.4.1.6 AT+&BTNAME

Function:

Set/read bluetooth name. Format

(ASCII):

Set AT+&BTNAME=[!]<name><CR> Read AT+&BTNAME Set return: +OK,<CR><LF><CR><LF> Read return: +OK=NAME,<CR>< LF><CR><LF>

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

6.4.1.7 AT+&BTMAC

Function:

Set/read Bluetooth MAC address. Format

(ASCII):

Set AT+&BTMAC=<MAC><CR> Read AT+&BTMAC setting return: +OK,<CR><LF><CR><LF> Read return: +OK=MAC,<CR><LF><CR><LF>

Parameters: Example of MAC

address setting: AT+&BTMAC=c00d308a0b08

6.4.1.8 AT+ BLESSCM

Function:

Specify 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)

6.4.2 BLE assisted WiFi distribution network AT command

6.4.2.1 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 log out, and the blue

Teeth turn off the radio. If you need to configure the network again, please start the Bluetooth distribution

network again. 6.4.3 Status code definition:

6.4.3.1 HCI 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 Found	0x39
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

7 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.

7.1 Enable and exit the Bluetooth system

7.1.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.1.2 Exit the Bluetooth system

AT+BTDES

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

7.2 Switch example broadcast

7.2.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.2.2 Open connectable broadcast example

AT+BLEDMADV=1

```
[WM_I] <0:20:53.986> ### tls_ble_demo_adv type=1
Starting advertising
GAP procedure initiated: advertise; disc_mode=2 adv_channel_map=0
 own_addr_type=0 adv_filter_policy=0 adv_itvl_min=64 adv_itvl_max
=64
+OK
```

7.2.3 Example of stopping broadcasting

AT+BLEDMADV=0

```
[WM_I] <0:23:33.818> ### tls_ble_demo_adv type=0
Stop advertising
GAP procedure initiated: stop advertising.
+OK
```

7.2.4 Exit the Bluetooth system

AT+BTDES

7.3 Switch example scan

7.3.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.3.2 Open scan example

AT+BLEDMSCAN=1

```
[WM_I] <1:41:40.442> ### tls_ble_demo_scan type=1
GAP procedure initiated: discovery; own_addr_type=0 filter_policy=0 passive=1 limited=0 filter_duplicates=0 duration=foreve
r
+OK

[WM_I] <1:41:40.484> gap_event, [BLE_GAP_EVENT_DISC]
    mfg_data=0x06:0x00:0x01:0x09:0x20:0x02:0x6e:0x9d:0xa9:0xc2:0xf7:0xd9:0xbf:0x39:0x6c:0x9c:0x15:0x40:0xf5:0x8d:0x0e:0x16:
0x5d:0x4a:0x0a:0x43:0x9d:0x06:0xce
[WM_I] <1:41:40.506> gap_event, [BLE_GAP_EVENT_DISC]
    flags=0x1a
    tx_pwr_lvl=7
    mfg_data=0x4c:0x00:0x10:0x06:0x6e:0x1d:0x67:0x68:0x89:0x80
[WM_I] <1:41:40.520> gap_event, [BLE_GAP_EVENT_DISC]
    flags=0x1a
    tx_pwr_lvl=12
    mfg_data=0x4c:0x00:0x10:0x07:0x1a:0x1f:0xe6:0x14:0xbf:0x73:0x18
[WM_I] <1:41:40.536> gap_event, [BLE_GAP_EVENT_DISC]
    flags=0x06
    mfg_data=0x4c:0x00:0x10:0x05:0x59:0x1c:0x60:0x55:0x47
[WM_I] <1:41:40.548> gap_event, [BLE_GAP_EVENT_DISC]
    mfg_data=0x06:0x00:0x01:0x09:0x20:0x02:0x90:0xa9:0x5d:0xcf:0x5a:0x59:0x92:0x39:0x3a:0xd9:0x63:0xda:0x9f:0x76:0x10:0x7a:
```

7.3.3 Example of stop scanning

AT+BLEDMSCAN=0

7.3.4 Exit the Bluetooth system

AT+BTDES

7.4 Switch example server

7.4.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.4.2 Enable demo server

AT+BLEDS=1

7.4.3 Stop demo server

AT+BLEDS=0

7.4.4 Exit the Bluetooth system

AT+BTDES

7.5 switch example client

7.5.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.5.2 Enable example client

AT+BLEDC=1

7.5.3 Stop the sample client

AT+BLEDC=0

7.5.4 Exit the Bluetooth system

AT+BTDES

7.6 Switch multi-connection example client

7.6.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.6.2 Enable multi-connection demo client

AT+BLEDCMC=1

7.6.3 Stop demo client

AT+BLEDCMC=0

7.6.4 Exit the Bluetooth system

AT+BTDES

7.7 Switch UART transparent transmission

7.7.1 Enable Bluetooth system

AT+BTEN=1.0

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

7.7.2 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

7.7.3 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

7.7.4 Exit the Bluetooth system

AT+BTDES

7.8 Enable auxiliary WiFi distribution network service

7.8.1 Turn on the Bluetooth function, enable the network distribution

AT+BTEN=1,0 service//enable the Bluetooth system

AT+ONESHOT=4 //Enable the distribution network service At this time, you

can use the APP to perform network distribution operations; note that after the network distribution is successful, the system will automatically cancel the distribution network service.

```
+OK=0,1

+OK
```

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

AT+ONESHOT=0 //Exit distribution network service//Exit Bluetooth

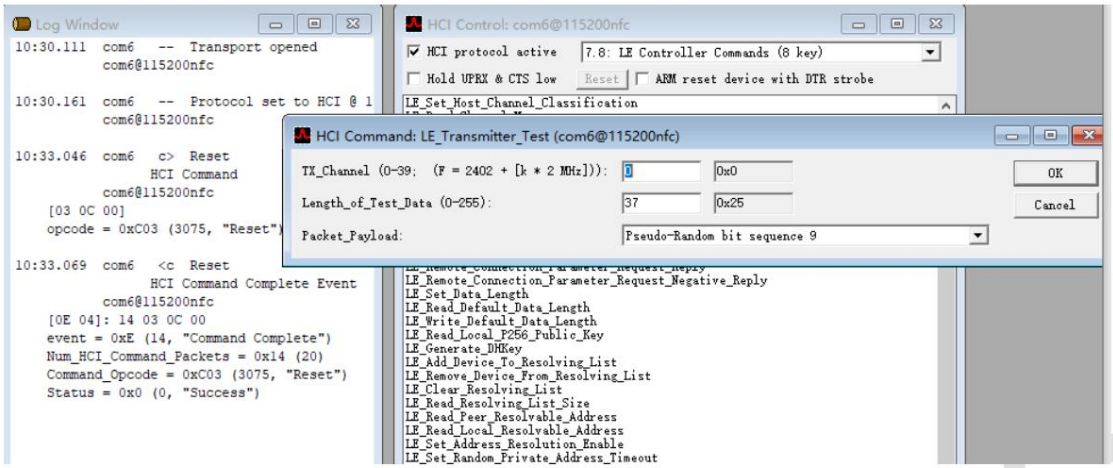
AT+BTDES system

7.9 W800 Test Mode

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

7.9.1 W800 enters test mode

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



7.9.2 W800 Exit Signaling Test

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