WLAN Framework Application Notes

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 14th December, 2018

Table of contents

1 Purpose and structure of this paper

1.1 Purpose and Background of this Paper

With the rapid development of the Internet of Things, more and more embedded devices are equipped with WIFI wireless network devices. In order to manage WIFI network devices, RT-Thread introduced the WLAN device management framework. This framework has many functions for controlling and managing WIFI, providing many conveniences for developers to use WIFI devices.

This article will help developers learn how to use this framework to control and manage WIFI. It will introduce the WLAN framework from multiple aspects such as concepts and examples. It will introduce relevant knowledge and demonstrate WIFI related functions through Shell and code, such as controlling WIFI scanning, connection, disconnection, etc.

1.2 Structure of this paper

- Introduction to WLAN framework
- WLAN framework configuration
- WLAN framework usage

2 Problem Statement

This application note mainly introduces the WLAN framework around the following issues:

- What is the WLAN framework and what functions does it have?
- How to configure the WLAN framework?
- How to use WLAN framework?

To solve the above problems, we need to understand the composition principle of the WLAN framework and learn how to use the various functions in the WLAN framework. Gradually begin to introduce the composition of the WLAN framework and the use of related functions.

3. Problem Solving

3.1 Introduction to WLAN Framework

The WLAN framework is a set of middleware developed by RT-Thread for managing WIFI. It connects to specific WIFI drivers and controls WIFI disconnection, scanning and other operations. It carries different applications and provides WIFI control, events, data diversion and other operations for applications, and provides a unified WIFI control interface for upper-level applications. The WLAN framework is mainly composed of three parts. The DEV driver interface layer provides a unified calling interface for the WLAN framework. The Manage management layer provides users with specific functions such as WIFI scanning, connection, disconnection and reconnection. The Protocol protocol is responsible for processing the data stream generated on WIFI, and different communication protocols can be mounted according to different usage scenarios, such as LWIP. It has the characteristics of simple use, complete functions, convenient docking, and strong compatibility.

3.2 WLAN Framework Composition

The following figure is the structural framework of the WIFI framework

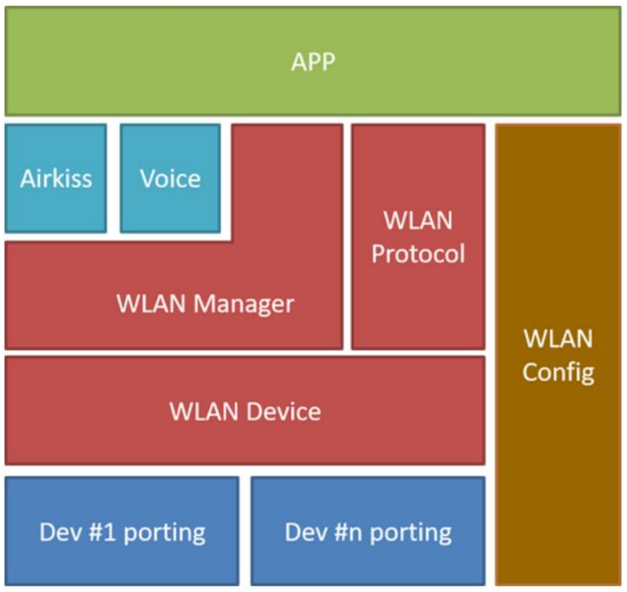


Figure **1:** *WIFI* frame

The first part, app, is the application layer, which is a specific application based on the WLAN framework, such as WiFi-related Shell commands.

The second part, airkiss and voice, is the network configuration layer, which provides functions such as wireless network configuration and sound wave network configuration.

The third part, WLAN manager, is the WLAN management layer. It can control and manage WLAN devices. It has WLAN control-related functions such as setting mode, connecting to hotspots, disconnecting hotspots, starting hotspots, scanning hotspots, etc. It also provides management functions such as disconnection reconnection and automatic hotspot switching.

The fourth part, WLAN protocol, is the protocol layer. The data stream is submitted to a specific protocol for parsing, and users can specify different protocols for communication.

The fifth part, WLAN config, is the parameter management layer. It manages the hotspot information and passwords that are successfully connected, and writes them to non-volatile storage media.

middle.

The sixth part, WLAN dev, is the driver interface layer, which connects to specific WLAN hardware and provides a unified calling interface for the management layer.

3.3 WLAN Framework Functions

- Automatic connection When the automatic connection function is turned on, as long as the WIFI is disconnected, it will automatically read the hotspot information that was successfully connected before and connect to the hotspot. If a hotspot connection fails, it will switch to the next hotspot information to connect until the connection is successful. The hotspot information used for automatic connection will be tried in the order of successful connection time, and the hotspot information that was successfully connected most recently will be used first. After the connection is successful, the hotspot information will be cached at the front and will be used first the next time the connection is disconnected.

- Parameter storage stores the WIFI parameters of a successful connection. A copy of the WIFI parameters will be cached in memory. If an external non-volatile storage interface is configured, a copy will be stored in an external storage medium. Users can implement the **struct** rt_wlan_cfg_ops structure according to their actual situation and save the parameters anywhere. The cached parameters mainly provide hotspot information for automatic connection. When the wifi is not connected, the cached parameters will be read and a connection attempt will be made.

• WIFI control provides complete WIFI control interface, scanning, connection, hotspot, etc. Provides WIFI related status callback events, disconnection, Connection, connection failure, etc. Provide users with a simple and easy-to-use WIFI management interface.

• Shell command: You can input commands in Msh to control WIFI to perform scanning, connection, disconnection and other actions. Print WIFI status and other debugging information interest.

3.4 WLAN framework configuration and initialization

This article will be based on the Zhengdian Atom STML4 IOT board development board, which has an AP6181 WiFi chip on board and WiFi driver Already implemented. Suitable for learning WLAN management framework.

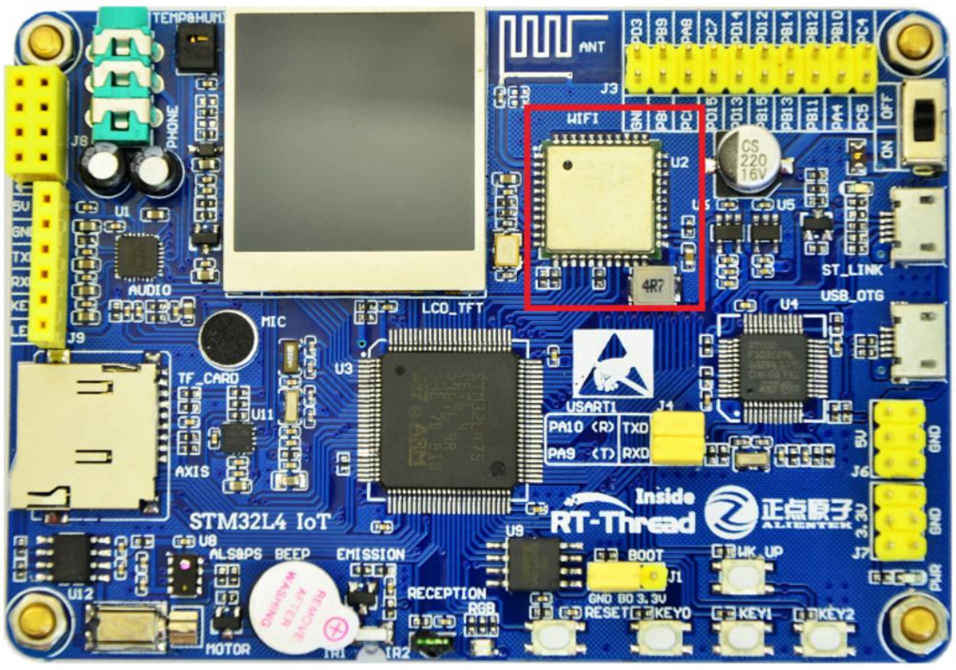


Figure 2: *WIFI* frame

WLAN framework configuration mainly includes the following aspects

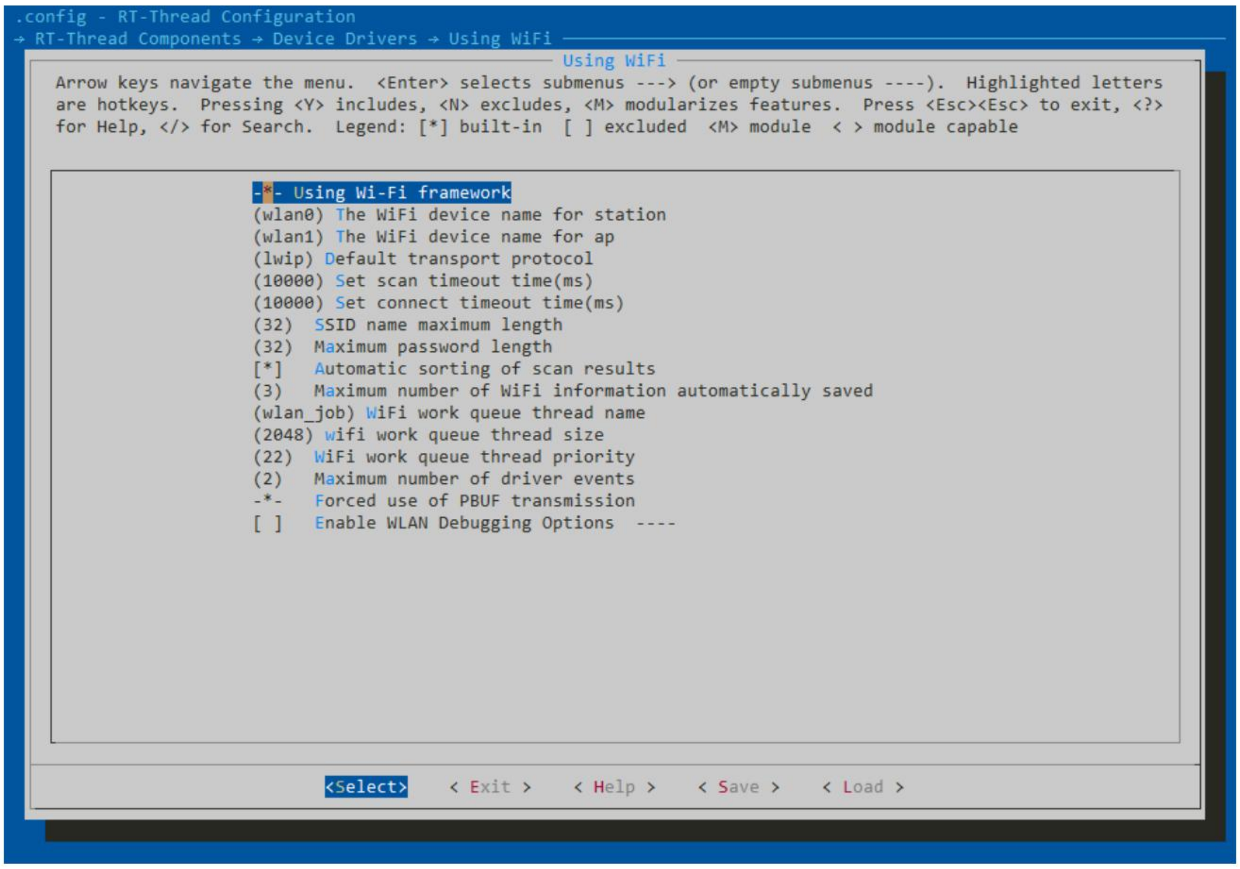
• Enable the WLAN framework and configure

• Initialize the WLAN device and specify the protocol to use

3.4.1. WLAN framework configuration

Here we will introduce the configuration related to the WIFI framework. Use the env tool to enter the IOT board directory and enter in the env command line menuconfig command, open the graphical configuration interface

• In the menuconfig configuration interface, select RT-Thread Components -> Device Drivers -> Using WiFi - > As shown in the figure below



```
.config - RT-Thread Configuration
→ RT-Thread Components → Device Drivers → Using WiFi
                              Using WiFi
  Arrow keys navigate the menu.  <Enter> selects submenus ---> (or empty submenus ----).  Highlighted letters
  are hotkeys.  Pressing <Y> includes, <N> excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
  for Help, </> for Search.  Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

                    -*- Using Wi-Fi framework
                    (wlan0) The WiFi device name for station
                    (wlan1) The WiFi device name for ap
                    (lwip) Default transport protocol
                    (10000) Set scan timeout time(ms)
                    (10000) Set connect timeout time(ms)
                    (32)  SSID name maximum length
                    (32)  Maximum password length
                    [*]   Automatic sorting of scan results
                    (3)   Maximum number of WiFi information automatically saved
                    (wlan_job) WiFi work queue thread name
                    (2048) wifi work queue thread size
                    (22)  WiFi work queue thread priority
                    (2)   Maximum number of driver events
                    -*-   Forced use of PBUF transmission
                    [ ]   Enable WLAN Debugging Options  ----

              <Select>    < Exit >    < Help >    < Save >    < Load >
```

Figure 3: *WLAN* Configuration

These configuration items are described below.

- Using Wi-Fi framework: Using WLAN management framework
- the WiFi device name for station: the default name of the Station device
- the WiFi device name for ap: the default name of the ap device
- Default transport protocol: default protocol
- Scan timeout time: Scan result timeout time
- connect timeout time: Connection timeout time
- SSID name maximum length: SSID maximum length
- Maximum password length: Maximum password length
- Automatic sorting of scan results:
- Maximum number of WiFi information automatically saved: Maximum number of entries automatically saved
- WiFi work queue thread name: WiFi background thread name
- wifi work queue thread size: WIFI background thread stack size
- WiFi work queue thread priority: WiFi background thread priority

• Maximum number of driver events: The maximum number of events registered at the dev layer

• Forced use of PBUF transmission: Forced use of PBUF to exchange data

• Enable WLAN Debugging Options: Enable debugging log

After configuring as shown above, save and exit

3.4.2. WLAN device initialization

The WLAN framework needs to specify a working mode for initializing the WLAN device, which needs to be implemented by writing code, so the following code is needed Initialize. The code is as follows:

```
int wifi_init(void)
{
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);          //Configure WLAN device
        Operating mode
    return 0;
}
```

After writing the above code, be sure to call and execute it once, and then you can use the WLAN framework to manage the device.

3.5 WLAN Usage

Note: Before performing the following operations, you must first perform WLAN device initialization. For details, see the WLAN device initialization section.

3.5.1. Shell operation WiFi

Using shell commands can help us quickly debug WiFi related functions. Just enter the corresponding command in msh and WiFi will be Execute the corresponding action. The following three commands will show how to operate WiFi using shell commands.

The wifi-related shell commands are as follows:

```
Wi-Fi                                : Print Help
wifi help wifi                       : View help
join SSID [PASSWORD] wifi ap         : Connect to wifi, SSDI is empty, use configuration to connect automatically
SSID [PASSWORD]                      ÿ Create hotspot
Wi-Fi scan                           ÿ Scan all hot spots
wifi disc                            : Disconnect
wifi ap_stop                         : Stop hotspot
Wi-Fi status                         ÿ Print wifi status sta + ap
wifi smartconfig                     ÿ Start the network configuration function
```

3.5.1.1. WiFi Scan After executing the WiFi scan command, the surrounding hotspot information will be printed on the terminal. Through the printed hotspot information, You can see multiple attributes such as SSID, MAC address, etc.

• The wifi scan command format is as follows

Machine Translated by Google

```
Wi-Fi scan
```

Command Description

Fields	describe
Wi-Fi	All wifi commands start with wifi
scan	Wi-Fi scan action

Enter this command in msh to perform a wifi command scan. The scan results are shown in the figure below.

```
Wi-Fi scan
SSID                                    MAC              security          rssi chn Mbps
------------------------------- ----------------- -------------- ---- --- ----
rtt_test_ssid_1                 c0:3d:46:00:3e:aa OPEN                 -14      8 300
test_ssid                       3c:f5:91:8e:4c:79 WPA2_AES_PSK -18 ec:88:8f:88:aa:9a      6 72
rtt_test_ssid_2                 WPA2_MIXED_PSK -47 c0:3d:46:00:41:ca                     6 144
rtt_test_ssid_3                 WPA2_MIXED_PSK -48                                       3 300
```

3.5.1.2. WiFi connection After executing the WiFi connection command, if the hotspot exists and the password is correct, the development board will connect to the hotspot and obtain

After the network connection is successful, you can use sockets for network communication.

• The wifi connection command format is as follows

```
wifi join rtt-SSID0 12345678
```

• Command parsing

Fields	describe
Wi-Fi	All wifi commands start with wifi
join	wifi executes the connection action
ssid	Hotspot name
123456789	Hotspot password. If you don't have a password, you don't need to enter this item.

• After the connection is successful, the obtained IP address will be printed on the terminal, as shown below

```
wifi join ssid_test 12345678
[I/WLAN.mgnt] wifi connect success ssid:ssid_test
[I/WLAN.lwip] Got IP address : 192.168.1.110
```

3.5.1.3. WiFi Disconnect After executing the WiFi Disconnect command, the development board will disconnect from the hotspot.

• The wifi disconnect command format is as follows

```
wifi disc
```

• Command parsing

Fields	describe
Wi-Fi	All wifi commands start with wifi
disc	Wi-Fi disconnect action

• After the disconnection is successful, the following information will be printed on the terminal, as shown in the figure below

```
wifi disc
 [I/WLAN.mgnt] disconnect success!
```

3.5.2. WiFi Scanning

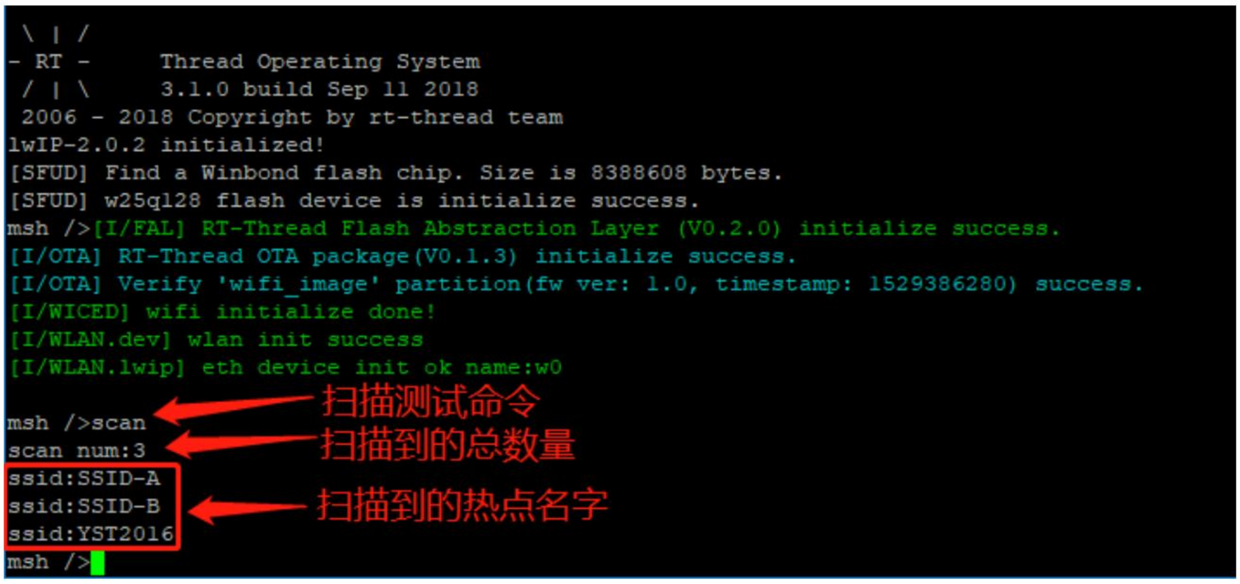
The following code will show WiFi synchronous scanning, and then we will print the results on the terminal. First, you need to perform WiFi initialization, and then execute the WiFi scanning function rt_wlan_scan_sync. This function is synchronous and returns the number and results of the scan. In this example, the name of the scanned hotspot will be printed out.

```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h> #include
<wlan_prot.h> #include
<wlan_cfg.h>

void wifi_scan(void) {

    struct rt_wlan_scan_result *result; int i = 0;


    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* WiFi scan */
    result = rt_wlan_scan_sync();
    /* Print scan results */ rt_kprintf("scan
    num:%d\n", result->num); for (i = 0; i < result->num; i++) {


        rt_kprintf("ssid:%s\n", result->info[i].ssid.val);
```

Machine Translated by Google

```
        }
}

int scan(int argc, char *argv[]) {

        wifi_scan(); return
        0;
}
MSH_CMD_EXPORT(scan, scan test.);
```

The results are as follows:



```
 \ | /
- RT -     Thread Operating System
 / | \     3.1.0 build Sep 11 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] w25q128 flash device is initialize success.
msh />[I/FAL] RT-Thread Flash Abstraction Layer (V0.2.0) initialize success.
[I/OTA] RT-Thread OTA package(V0.1.3) initialize success.
[I/OTA] Verify 'wifi_image' partition(fw ver: 1.0, timestamp: 1529386280) success.
[I/WICED] wifi initialize done!
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w0

msh />scan
scan num:3
ssid:SSID-A
ssid:SSID-B
ssid:YST2016
msh />
```

Figure 4: scanning

3.5.3. WiFi connection and disconnection

The following code will demonstrate a WiFi simultaneous connection.

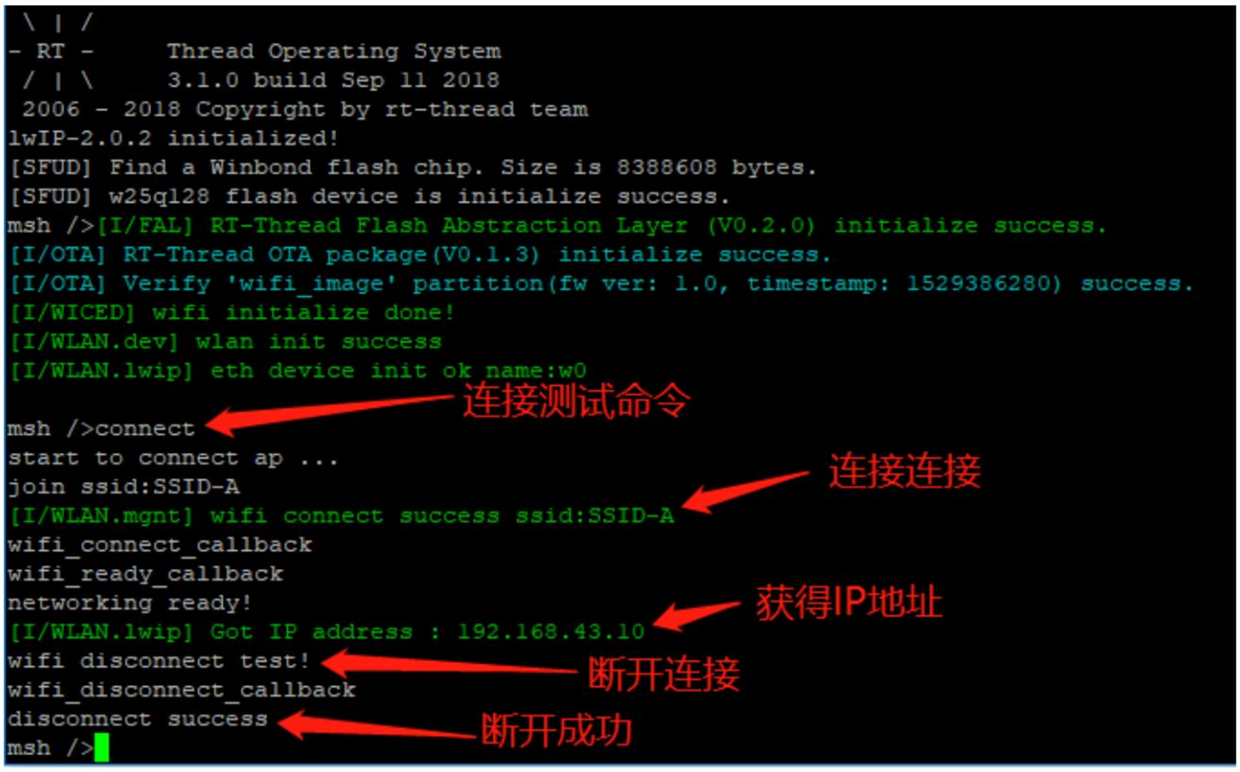
First, you need to perform WIFI initialization, and then create a semaphore to wait for the RT_WLAN_EVT_READY event. Register the callback function of the event you need to pay attention to, execute the rt_wlan_connect wifi connection function, and the function returns whether the connection is successful. However, communication cannot be carried out after the connection is successful, and you need to wait for the network to obtain an IP. Use the semaphore created in advance to wait for the network to be ready. After the network is ready, communication can be normal.

After connecting to WIFI, wait for a while and then execute the rt_wlan_disconnect function to disconnect. The disconnection operation is blocking, and the return value indicates whether the disconnection is successful.

```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h>
#include <wlan_prot.h> #include
<wlan_cfg.h>

#define WLAN_SSID                       "SSID-A"
```

```
#define WLAN_PASSWORD                          "12345678"
#define NET_READY_TIME_OUT                      (rt_tick_from_millisecond(15 * 1000))

static rt_sem_t net_ready = RT_NULL;

static void
wifi_ready_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__);
    rt_sem_release(net_ready);
}

static void
wifi_connect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_disconnect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_connect_fail_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

rt_err_t wifi_connect(void) {

    rt_err_t result = RT_EOK;

    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* station connect */
    rt_kprintf("start to connect ap ...\n"); net_ready =
    rt_sem_create("net_ready", 0, RT_IPC_FLAG_FIFO);
    rt_wlan_register_event_handler(RT_WLAN_EVT_READY,
                wifi_ready_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED,
                wifi_connect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_DISCONNECTED,
                wifi_disconnect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED_FAIL,
                wifi_connect_fail_callback, RT_NULL);

    /* connect wifi */
    result = rt_wlan_connect(WLAN_SSID, WLAN_PASSWORD);

    if (result == RT_EOK) {

        /* waiting for IP to be got successfully */ result =
        rt_sem_take(net_ready, NET_READY_TIME_OUT); if (result == RT_EOK) {


            rt_kprintf("networking ready!\n");
        }
        else
        {
            rt_kprintf("wait ip got timeout!\n");

        } rt_wlan_unregister_event_handler(RT_WLAN_EVT_READY);
        rt_sem_delete(net_ready);

        rt_thread_delay(rt_tick_from_millisecond(5 * 1000)); rt_kprintf("wifi
        disconnect test!\n");
        /* disconnect */
        result = rt_wlan_disconnect(); if (result !=
        RT_EOK) {

            rt_kprintf("disconnect failed\n"); return result;


        } rt_kprintf("disconnect success\n");
    }
    else
    {
        rt_kprintf("connect failed!\n");

    } return result;
}

int connect(int argc, char *argv[]) {
    wifi_connect();
    return 0;
}
MSH_CMD_EXPORT(connect, connect test.);
```

The results are as follows



```
 \ | /
- RT -     Thread Operating System
 / | \     3.1.0 build Sep 11 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] w25q128 flash device is initialize success.
msh />[I/FAL] RT-Thread Flash Abstraction Layer (V0.2.0) initialize success.
[I/OTA] RT-Thread OTA package(V0.1.3) initialize success.
[I/OTA] Verify 'wifi_image' partition(fw ver: 1.0, timestamp: 1529386280) success.
[I/WICED] wifi initialize done!
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w0

msh />connect
start to connect ap ...
join ssid:SSID-A
[I/WLAN.mgnt] wifi connect success ssid:SSID-A
wifi_connect_callback
wifi_ready_callback
networking ready!
[I/WLAN.lwip] Got IP address : 192.168.43.10
wifi disconnect test!
wifi_disconnect_callback
disconnect success
msh />
```

Figure 5: Disconnect

3.5.4. WiFi turns on automatic reconnection

First, enable the automatic reconnection function, use the command line to connect to a hotspot A, and then connect to another hotspot B. After waiting for a few seconds, power off hotspot B, and the system will automatically retry to connect to hotspot B. At this time, hotspot B cannot be connected, and the system automatically switches to hotspot A for connection. After successfully connecting to A, the system stops connecting.

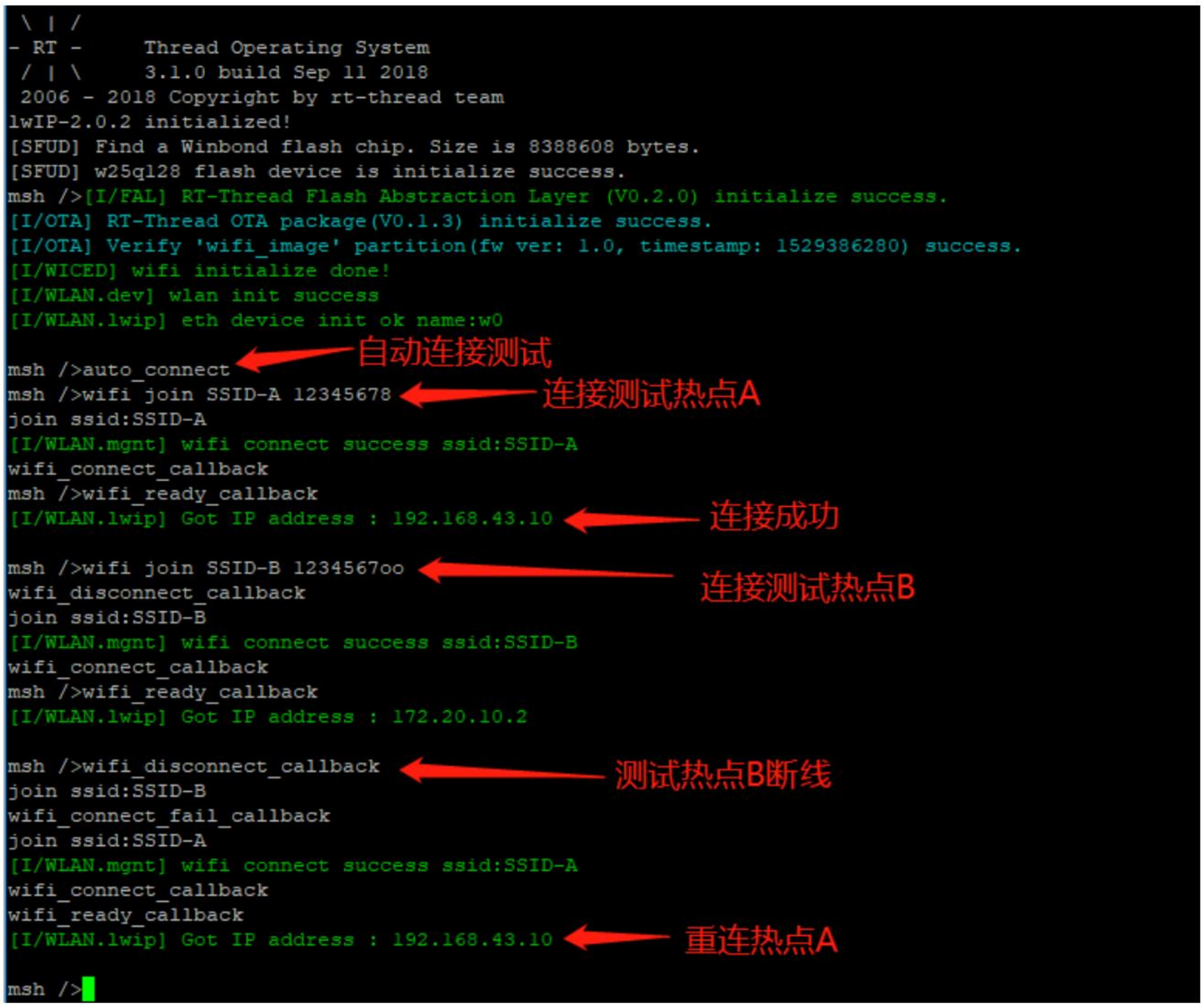
```
#include <rthw.h>
#include <rtthread.h>

#include <wlan_mgnt.h>
#include <wlan_prot.h>
#include <wlan_cfg.h>

static void
wifi_ready_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

        rt_kprintf("%s\n", __FUNCTION__);
}
```

Machine Translated by Google

```
static void
wifi_connect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_disconnect_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

static void
wifi_connect_fail_callback(int event, struct rt_wlan_buff *buff, void *parameter) {

    rt_kprintf("%s\n", __FUNCTION__); if ((buff !=
    RT_NULL) && (buff->len == sizeof(struct rt_wlan_info))) {

        rt_kprintf("ssid : %s \n", ((struct rt_wlan_info *)buff->data)->ssid.val);
    }
}

int wifi_autoconnect(void) {

    /* Configuring WLAN device working mode */
    rt_wlan_set_mode(RT_WLAN_DEVICE_STA_NAME, RT_WLAN_STATION);
    /* Start automatic connection */
    rt_wlan_config_autoreconnect(RT_TRUE); /* register
    event */
    rt_wlan_register_event_handler(RT_WLAN_EVT_READY,
              wifi_ready_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED,
              wifi_connect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_DISCONNECTED,
              wifi_disconnect_callback, RT_NULL);
    rt_wlan_register_event_handler(RT_WLAN_EVT_STA_CONNECTED_FAIL,
              wifi_connect_fail_callback, RT_NULL);
    return 0;
}

int auto_connect(int argc, char *argv[])
```

```
{
        wifi_autoconnect();
        return 0;
}
MSH_CMD_EXPORT(auto_connect, auto connect test.);
```

The results are as follows:



```
 \ | /
- RT -     Thread Operating System
 / | \     3.1.0 build Sep 11 2018
 2006 - 2018 Copyright by rt-thread team
lwIP-2.0.2 initialized!
[SFUD] Find a Winbond flash chip. Size is 8388608 bytes.
[SFUD] w25q128 flash device is initialize success.
msh />[I/FAL] RT-Thread Flash Abstraction Layer (V0.2.0) initialize success.
[I/OTA] RT-Thread OTA package(V0.1.3) initialize success.
[I/OTA] Verify 'wifi_image' partition(fw ver: 1.0, timestamp: 1529386280) success.
[I/WICED] wifi initialize done!
[I/WLAN.dev] wlan init success
[I/WLAN.lwip] eth device init ok name:w0

msh />auto_connect
msh />wifi join SSID-A 12345678
join ssid:SSID-A
[I/WLAN.mgnt] wifi connect success ssid:SSID-A
wifi_connect_callback
msh />wifi_ready_callback
[I/WLAN.lwip] Got IP address : 192.168.43.10

msh />wifi join SSID-B 12345670o
wifi_disconnect_callback
join ssid:SSID-B
[I/WLAN.mgnt] wifi connect success ssid:SSID-B
wifi_connect_callback
msh />wifi_ready_callback
[I/WLAN.lwip] Got IP address : 172.20.10.2

msh />wifi_disconnect_callback
join ssid:SSID-B
wifi_connect_fail_callback
join ssid:SSID-A
[I/WLAN.mgnt] wifi connect success ssid:SSID-A
wifi_connect_callback
wifi_ready_callback
[I/WLAN.lwip] Got IP address : 192.168.43.10

msh />
```

Figure 6: auto connect

4 References

4.1 APIs related to this article

4.2 API List

API	Location
rt_wlan_set_mode	wlan_mgnt.c

API	Location
rt_wlan_prot_attach	wlan_prot.c
rt_wlan_scan_sync	wlan_mgnt.c
rt_wlan_connect	wlan_mgnt.c
rt_wlan_disconnect	wlan_mgnt.c
rt_wlan_config_autoreconnect	wlan_mgnt.c

4.3 Detailed explanation of core API

4.3.1. rt_wlan_set_mode()

4.4 Function

Register the WLAN device to the WLAN device framework

4.5 Function Prototype

```
rt_err_t rt_wlan_set_mode(const char *dev_name, rt_wlan_mode_t mode);
```

4.6 Function Parameters

parameter	describe
dev_name	Wi-Fi Device Name
mode	WLAN device working mode

4.7 Return Value

return value	describe
-RT_EINVAL	Parameter error
-RT_EIO	Device not found
-RT_ERROR	Execution failed
RT_EOK	execution succeed

wlan mode can take one of the following values RT_WLAN_NONE clears the working mode RT_WLAN_STATION works in STATION mode RT_WLAN_AP works in AP mode

4.7.1. rt_wlan_prot_attach()

4.8 Function

Specify the protocol used by the WLAN device

4.9 Function Prototype

```
rt_err_t rt_wlan_prot_attach(const char *dev_name, const char *prot_name);
```

4.10 Function Parameters

parameter	describe
name	Wi-Fi Device Name
prot_name	Protocol Name

4.11 Return Value

return value	describe
-RT_ERROR	Execution failed
RT_EOK	execution succeed

type can be one of the following values RT_WLAN_PROT_LWIP The protocol type is LWIP

4.11.1. rt_wlan_scan_sync()

4.12 Function

Simultaneous scanning of hot spots

4.13 Function prototype

```
struct rt_wlan_scan_result *rt_wlan_scan_sync(void);
```

4.14 Function Parameters

none

4.15 Return Value

return value	describe
rt_wlan_scan_result	Scan Results

The scan result is a structure with the following members:

num : info number info : info pointer

```
struct rt_wlan_scan_result
{
        rt_int32_t num;
        struct rt_wlan_info *info;
};
```

4.15.1. rt_wlan_connect()

4.16 Function

Connect to hotspot simultaneously

4.17 Function prototype

```
rt_err_t rt_wlan_connect(const char *ssid, const char *password);
```

4.18 Function Parameters

parameter	describe
ssid	WIFI NAME
password	WIFI PASSWORD

4.19 Return Value

return value	describe
-RT_EINVAL	Parameter error
-RT_EIO	Unregistered Device
-RT_ERROR	Connection failed
RT_EOK	connection succeeded

return value	describe

4.19.1. rt_wlan_disconnect()

4.20 Function

Synchronize disconnect hotspot

4.21 Function prototype

```
rt_err_t rt_wlan_disconnect(void);
```

4.22 Function Parameters

none

4.23 Return Value

return value	describe
-RT_EIO	Unregistered Device
-RT_ENOMEM	Not enough storage
-RT_ERROR	Disconnection failed
RT_EOK	Disconnect successfully

4.23.1. rt_wlan_config_autoreconnect()

4.24 Function

Configuring automatic reconnection mode

4.25 Function prototype

```
void rt_wlan_config_autoreconnect(rt_bool_t enable);
```

4.26 Function Parameters

parameter	describe
enable	enable/disable automatic reconnection

4.27 Return Value

none