Airoha IoT SDK for RTOS Wi-Fi Developer's Guide

Version: 1.8

Release date: 21 May 2018

Document Revision History

Revision	Date	Description
1.0	7 March 2016	Initial release
1.1	22 April 2016	• Wi-Fi WPS connection support. • Replaced NVRAM with NVDM. • Changed the API naming to register and unregister an event handler, such as replacing the `wifi_connection_register_event_notifier()` with `wifi_connection_register_event_handler()`.
1.2	30 June 2016	• The document is updated to include Wi-Fi initialization APIs and supported library descriptions for the Wi-Fi module. • Instructions on how to configure the Wi-Fi module in repeater mode. • Updated the Scan APIs and Smart Connection APIs. • Deleted the Wi-Fi Profile.
1.3	2 September 2016	• Updated Table 3 to list the functions that can be invoked before task scheduler running. • Instructions on how to configure the Wi-Fi connection with a specific BSSID. • Updated Smart Connection Overview and Figure 3.
1.4	4 November 2016	• Renamed “`libbsp.a`” to “`libwifi.a`”.
1.5	13 January 2017	• Updated the software architecture diagram. • Added the monitor mode for raw Wi-Fi packets.
1.6	5 May 2017	• Updated the overview to add description for MT7686/MT7682/MT5932 chipsets. • Added Table 1 to describe supported features of different chipsets. • Updated supported libraries. • Added detailed description about payload of each event in Wi-Fi event Table 5. • Added `wifi_config_set_country_code()/wifi_config_get_country_code()` APIs in Table 6. • Modified the example code to process the WEP key length when the device is in AP mode. • Renamed “Elian APK” to “SmartConnection APK”. • Modified the folder name for APK and Smart Connection libraries. • Updated the supported version of Smart Connection protocol.
1.7	15 September 2017	• Updated the software architecture in Figure 1. • Updated section 1.1, “Supported features” to include support for MT7682/MT7687 STA WPS and repeater modes. • Updated section 2.7, “WPS”. AP + Enrollee PIN method is only supported in MT7687 and MT7697.
1.8	21 May 2018	• Added lwip_network_init() and lwip_net_start() usage description in

Revision	Date	Description
		section 2. • Updated MT5932 feature description.

Table of Contents

1. Overview 1
1.1. Supported features 1
1.2. Supported libraries 2
1.3. Software architecture of the Wi-Fi module 2
2. Using the Module 3
2.1. Using the Wi-Fi module in STA mode 3
2.2. Using the Wi-Fi module in AP mode 12
2.3. Using the Wi-Fi module in a repeater mode 18
2.4. Scan 20
2.5. Smart Connection 24
2.6. Monitor mode and RX filter 27
2.7. WPS 28

Lists of Tables and Figures

Table 1. Supported features of different chipsets 1
Table 2. Libraries for the Wi-Fi mode configuration 2
Table 3. Configuration functions....3
Table 4. The supported security modes....5
Table 5. Wi-Fi events....9
Table 6. Configuration and connection APIs in the AP mode 12
Table 7. The functions not supported in repeater mode....19
Table 8. Scan mode 20
Table 9. Scan option....21
Table 10. Channels supporting 2.4G 21
Table 11. Channels supporting 5G 21
Table 12. Scan APIs 22
Table 13. Smart Connection APIs....25
Table 14. RX filter definitions....27
Table 15. WPS APIs....29

Figure 1. Software architecture 2
Figure 2. Repeater mode topology 18
Figure 3. Communicating with an IoT device using Smart Connection....25

1. Overview

MT7687, MT7697, MT7686, and MT7682 chipsets are based on Wi-Fi System-on-Chip (SOC) with embedded TCP/IP stack for Internet of Things (IoT) devices that can connect to other smart devices or to cloud applications and services directly. Airoha IoT SDK provides API and example applications for the Wi-Fi module.

This document guides you through the following:

- Initializing the Wi-Fi module in Access Point (AP) and Station (STA) modes.
- Configuring the module to operate in STA mode or AP mode.
- Scanning for the available stations.
- Using smart connection and the RX filter.

1.1. Supported features

The Wi-Fi module on Airoha IoT SDK for RTOS platform supports the following features and modes.

- STA Open or wired equivalent privacy (WEP)/Wi-Fi protected access (WPA)/WPA2 personal.
- STA Wi-Fi powersave mode and delivery traffic indication message (DTIM) configuration. Power save mode and DTIM are essential in Wi-Fi connectivity support.
- STA Wi-Fi Protected Setup (WPS), is a network security standard to create a secure wireless home network.
- Wi-Fi aggregated MAC protocol data unit (A-MPDU) support. It enables to pack multiple MPDUs together to reduce the overheads and average them over multiple frames, thereby increasing the data rate.
- SoftAP Open/WEP-Open/WPA/WPA2 personal, the authentication modes are the same as for STA mode.
- SoftAP WPS.
- High throughput for 20 MHz and 40 MHz bandwidths or both (HT40/HT20/Mix). This module sets the Wi-Fi bandwidth.
- Repeater Mode.

Supported features for each chipset are listed in Table 1.

Table 1. Supported features of different chipsets

Features	MT7687	MT7697	MT7686	MT7682
STA Open/WEP-Open/WPA/WPA2	Y	Y	Y	Y
STA Wi-Fi power saving mode	Y	Y	Y	Y
STA WPS	Y	Y	Y	Y
Wi-Fi (A-MPDU)	Y	Y	Y	Y
SoftAP Open/WEP-Open/WPA/WPA2	Y	Y	Y	Y
HT40/HT20/Mix	Y	Y	Y	Y
Repeater Mode	Y	Y	Y	Y

1.2. Supported libraries

The SDK provides configurable libraries to use in your projects. Simply link the corresponding library to your project described in Table 2.

Table 2. Libraries for the Wi-Fi mode configuration

Wi-Fi Mode	Library			
	MT7687	MT7697	MT7686	MT7682
Station SoftAP Repeater WPS	`libwifi.a` `libminisupp.a`		`libwifi.a` `libwifi_ram.a`	

1.3. Software architecture of the Wi-Fi module

There are three software layers: **Driver**, **Middleware** and **Application**, as shown in Figure 1. The Wi-Fi module is located in **Middleware**. For more details, refer to Airoha IoT SDK for RTOS get started guide.

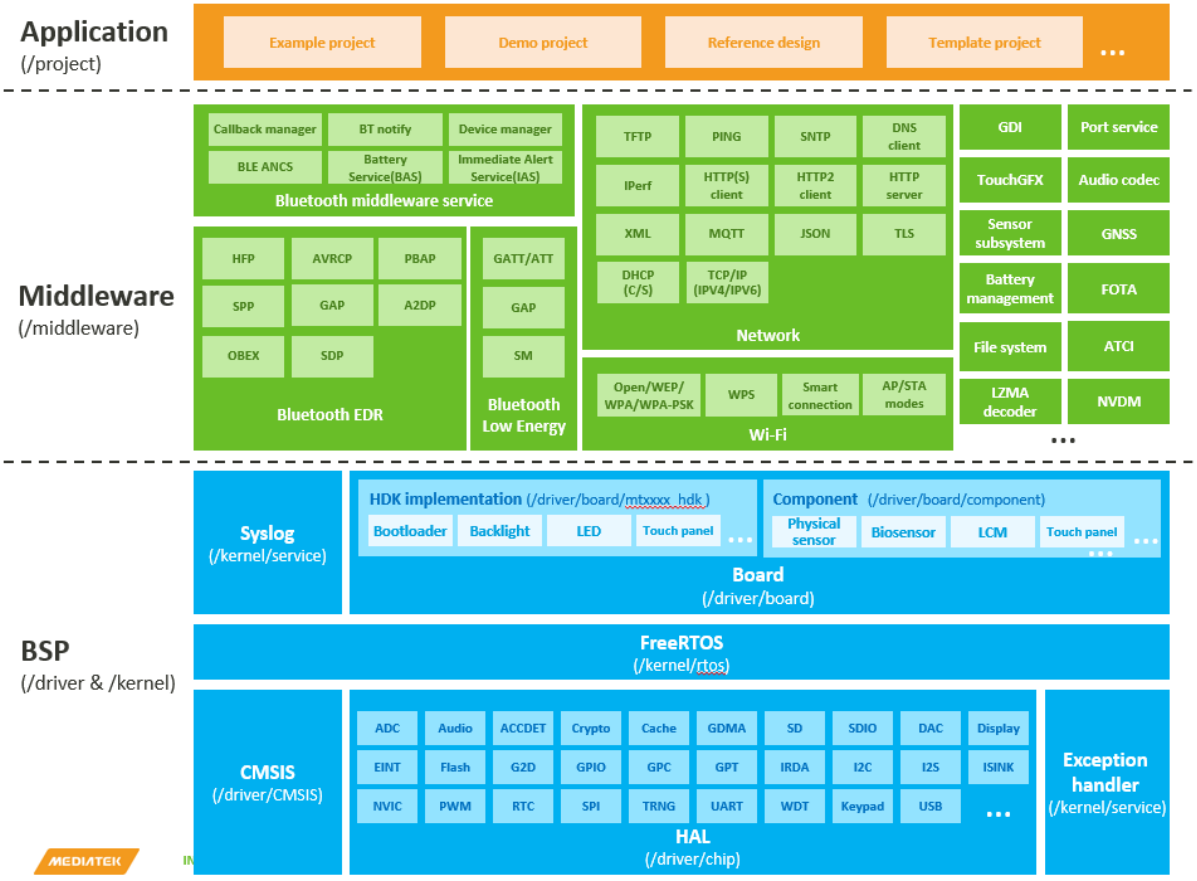

Figure 1. Software architecture

2. Using the Module

The Wi-Fi module can only be applied once the settings are configured at system reboot. Apply your settings in the `wifi_init()` API to initialize the driver in STA or AP mode. To configure the board as a repeater, see section 2.2, “Using the Wi-Fi module in AP mode”.

Starting from the SDK v4, initialize the Wi-Fi settings by calling the `wifi_init()` function. The legacy Wi-Fi profile API is disabled by default. To enable the legacy Wi-Fi profile API, define compile option `MTK_WIFI_PROFILE_ENABLE`.

If tx/rx packets are necessary, lwip_network_init() and lwip_net_start() must also be called. The two functions will do the TCPIP stack and net interface initialization and the DHCP client/ DHCP server process initialization.

Most of the Wi-Fi APIs in wifi_api.h require FreeRTOS to be running. These APIs should be invoked in a task triggered by vTaskStartScheduler(). The APIs in Table 3 are invoked in the main() function as an exception.

Table 3. Configuration functions

API	API
`wifi_init()`	`wifi_connection_register_event_handler()`
`wifi_config_get_mac_address ()`	`wifi_connection_unregister_event_handler ()`

You can register an event handler to complete the initialization and call the Wi-Fi APIs, as shown in the example code below.

```
/* Register wifi initialization event handler in main() function */
wifi_connection_register_event_handler(WIFI_EVENT_IOT_INIT_COMPLETE,
                                       user_wifi_init_callback);

/* User-defined wifi initialization callback function */
bool g_wifi_init_ready = false;
void user_wifi_init_callback() { g_wifi_init_ready = true; }
void user_wifi_init_qurey_status(){
    while(g_wifi_init_ready == false) {
         vTaskDelay(20);
    }
}

/* User-defined task */
user_task()
{
    user_wifi_init_query_status();
    /* Call APIs to connect only after Wi-Fi is initialized. */
    wifi_config_set_ssid(WIFI_PORT_STA, "REMOTE_AP", sizeof("REMOTE_AP"));
    wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678",
sizeof("12345678"));
    wifi_config_reload_setting();
}
```

2.1. Using the Wi-Fi module in STA mode

In most cases, the configuration to initialize the Wi-Fi in STA mode takes effect immediately. However, some settings can only take effect by calling the `wifi_config_reload_setting()` after calling the following APIs:

- `wifi_config_set_pmk()`
- `wifi_config_set_security_mode()`
- `wifi_config_set_ssid()`
- `wifi_config_set_wep_key()`
- `wifi_config_set_wpa_psk_key()`

The configuration APIs use in-band mechanism for Wi-Fi driver and Wi-Fi firmware communication, thus these APIs must be called after the OS task scheduler has started, to make sure an in-band task is running. The function `wifi_config_register_rx_handler()` registers `wifi_rx_handler()` to manage the raw packets. There is a limitation on calling this handler as it's using an in-band mechanism. It's restricted to call any in-band functions like Wi-Fi configuration APIs or Wi-Fi connection APIs inside `wifi_rx_handler()`.

Note 1. There is no need to configure the security mode in STA mode, as the security mode of the STA can automatically match to the security mode of the AP.

Note 2. If AP's security mode is unknown, call the `wifi_config_set_wpa_psk_key()` and `wifi_config_set_wep_key()` APIs to set the password.

To use the module in a STA mode, apply any of the examples described below.

2.1.1. The AP router operates in open mode

1) In this example, the AP router operates in open mode with an SSID of "REMOTE_AP". Initialize the module in STA mode, as shown below.

   a) Define the operation mode (`config.opmode`) and the SSID (`config.sta_config.ssid`) in the `wifi_config_t` structure.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_STA_ONLY;
strcpy((char *)config.sta_config.ssid, "REMOTE_AP");
config.sta_config.ssid_length = strlen((char *)config.sta_config.ssid);
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

You've now successfully initialized the Wi-Fi module in STA mode.

2) To use configuration APIs in STA mode to connect to the AP router:

   a) Call the `wifi_config_set_opmode()` function to set the opmode to WIFI_MODE_STA_ONLY.

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
```

   b) Call the `wifi_config_set_ssid()` function to set the port to WIFI_PORT_STA and the SSID to "REMOTE_AP".

```
wifi_config_set_ssid(WIFI_PORT_STA, "REMOTE_AP", strlen("REMOTE_AP"));
```

   c) Apply the parameters by calling the `wifi_config_reload_setting()` function.

```
wifi_config_reload_setting();
```

You've successfully configured the Wi-Fi settings.

2.1.2. The AP router operates in WPA2PSK authentication mode

1) In this example, the AP router operates in WPA2PSK authentication mode with AES encryption type, the password is "12345678" and the SSID of the AP router is "REMOTE_AP". Initialize the module in STA mode, as shown below.

   a) Define the operation mode (`config.opmode`), password (`config.sta_config.password`) and the SSID (`config.sta_config.ssid`) in the `wifi_config_t` structure.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_STA_ONLY;
strcpy((char *)config.sta_config.ssid, "REMOTE_AP");
config.sta_config.ssid_length = strlen((char *)config.sta_config.ssid);
strcpy((char *)config.sta_config.password, "12345678");
config.sta_config.password_length = strlen((char
*)config.sta_config.password);
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

The STA mode supports various types of security modes, as shown in Table 4. For more information about the authentication mode and encryption group types, refer to `wifi_auth_mode_t` and `wifi_encrypt_type_t` types in the Wi-Fi API reference guide.

Table 4. The supported security modes

Authentication mode	Encryption type
WIFI_AUTH_MODE_OPEN	WIFI_ENCRYPT_TYPE_WEP_ENABLED
WIFI_AUTH_MODE_WPA_PSK	WIFI_ENCRYPT_TYPE_TKIP_ENABLED
WIFI_AUTH_MODE_WPA_PSK	WIFI_ENCRYPT_TYPE_AES_ENABLED
WIFI_AUTH_MODE_WPA_PSK	WIFI_ENCRYPT_TYPE_TKIP_AES_MIX
WIFI_AUTH_MODE_WPA2_PSK	WIFI_ENCRYPT_TYPE_TKIP_ENABLED
WIFI_AUTH_MODE_WPA2_PSK	WIFI_ENCRYPT_TYPE_AES_ENABLED
WIFI_AUTH_MODE_WPA2_PSK	WIFI_ENCRYPT_TYPE_TKIP_AES_MIX
WIFI_AUTH_MODE_WPA_PSK_WPA2_PSK	WIFI_ENCRYPT_TYPE_TKIP_ENABLED
WIFI_AUTH_MODE_WPA_PSK_WPA2_PSK	WIFI_ENCRYPT_TYPE_AES_ENABLED
WIFI_AUTH_MODE_WPA_PSK_WPA2_PSK	WIFI_ENCRYPT_TYPE_TKIP_AES_MIX

2) To use configuration APIs in STA mode to connect to the AP router:

   a) Call the `wifi_config_set_opmode()` function to set the opmode to WIFI_MODE_STA_ONLY.

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
```

   b) Call the `wifi_config_set_ssid()` function to set the port to WIFI_PORT_STA and the SSID to "REMOTE_AP".

```
wifi_config_set_ssid(WIFI_PORT_STA, "REMOTE_AP", strlen("REMOTE_AP"));
```

   c) Set the password to "12345678" by calling the `wifi_config_set_wpa_psk_key()` function.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678",
strlen("12345678"));
```

   d) Apply the parameters by calling the `wifi_config_reload_setting()` function.

```
wifi_config_reload_setting();
```

The STA mode supports various types of security modes, as shown in Table 4. For more information about the authentication mode and encryption group types, refer to `wifi_auth_mode_t` and `wifi_encrypt_type_t` types in the Wi-Fi API Reference Manual.

2.1.3. The AP router operates in open WEP mode

1) In this example, the AP router operates in open WEP mode with the key index set to 0, the password set to "1234567890" and the SSID of the router is "REMOTE_AP". Initialize the module in STA mode, as shown below.

   a) Define the operation mode (`config.opmode`), the SSID (`config.sta_config.ssid`), password (`config.sta_config.password`) and key index (`config_ext.sta_wep_key_index`) in the `wifi_config_t` and `wifi_config_ext_t` data structures. Set `config_ext.sta_wep_key_index_present` to 1 to enable the presence of `config_ext.sta_wep_key_index`.

```
wifi_config_t config = {0};
wifi_config_ext_t config_ext = {0};
config.opmode = WIFI_MODE_STA_ONLY;
strcpy((char *)config.sta_config.ssid, "REMOTE_AP");
config.sta_config.ssid_length = strlen((char *)config.sta_config.ssid);
strcpy((char *)config.sta_config.password, "1234567890");
config.sta_config.password_length = strlen((char
*)config.sta_config.password);
config_ext.sta_wep_key_index_present = 1;
config_ext.sta_wep_key_index = 0;
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, &config_ext);
```

Note, the open WEP mode is configured slightly different from the other two examples described in this section. It requires both `wifi_config_t` and `wifi_config_ext_t` parameters.

2) To use configuration APIs in STA mode to connect to the AP router:

   a) Call the `wifi_config_set_opmode()` function to set the opmode to WIFI_MODE_STA_ONLY.

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
```

   b) Call the `wifi_config_set_ssid()` function to set the port to WIFI_PORT_STA and the SSID to "REMOTE_AP".

```
wifi_config_set_ssid(WIFI_PORT_STA, "REMOTE_AP", strlen("REMOTE_AP"));
```

   c) Set the WEP key by calling the `wifi_config_set_wep_key()` function.

```
wifi_wep_key_t wep_key = {{{0}}};
wep_key.wep_tx_key_index = 0;
wep_key.wep_key[0] = "1234567890";
wep_key.wep_key_length[0] = strlen("1234567890");
wifi_config_set_wep_key(WIFI_PORT_STA, &wep_key);
```

   d) Apply the settings by calling `wifi_config_reload_setting()` function.

```
wifi_config_reload_setting();
```

2.1.4. Two AP routers operate in WPA2PSK authentication mode

1) In this example, two AP routers (AP1 and AP2) operate in WPA2PSK authentication mode with AES encryption type. The password of each AP is "12345678" and the SSID of both routers is "REMOTE_AP".

   The BSSID of AP1 is “00:11:22:33:44:55”. The BSSID of AP2 is “55:44:33:22:11:00”. The goal is to connect to AP1. Initialize the module in STA mode, as shown below.

   a) Define the operation mode (`config.opmode`), password (`config.sta_config.password`), the SSID (`config.sta_config.ssid`), BSSID present (`config.sta_config.bssid_present`) and the BSSID (`config.sta_config.bssid`) in the `wifi_config_t` structure.

```
wifi_config_t config = {0};
uint8_t target_bssid[WIFI_MAC_ADDRESS_LENGTH] =
{0x00,0x11,0x22,0x33,0x44,0x55};
config.opmode = WIFI_MODE_STA_ONLY;
strcpy((char *)config.sta_config.ssid, "REMOTE_AP");
config.sta_config.ssid_length = strlen((char *)config.sta_config.ssid);
strcpy((char *)config.sta_config.password, "12345678");
config.sta_config.password_length = strlen((char
*)config.sta_config.password);
config.sta_config.bssid_present = 1;
os_memcpy(config.sta_config.bssid, target_bssid, WIFI_MAC_ADDRESS_LENGTH);
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

2) To use configuration APIs in STA mode to connect to the AP router:

   a) Call the `wifi_config_set_opmode()` function to set the opmode to WIFI_MODE_STA_ONLY.

```
wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
```

   b) Call the `wifi_config_set_ssid()` function to set the port to WIFI_PORT_STA and the SSID to "REMOTE_AP".

```
wifi_config_set_ssid(WIFI_PORT_STA, "REMOTE_AP", strlen("REMOTE_AP"));
```

   c) Set the password to "12345678" by calling the `wifi_config_set_wpa_psk_key()` function.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_STA, "12345678",
strlen("12345678"));
```

   d) Set the target BSSID to connect by calling the `wifi_config_set_bssid()` function.

```
uint8_t target_bssid = {0x00,0x11,0x22,0x33,0x44,0x55};
wifi_config_set_bssid(target_bssid);
```

   e) Apply the parameters by calling the `wifi_config_reload_setting()` function.

```
wifi_config_reload_setting();
```

2.1.5. Disable and enable auto connect option in STA mode

The auto-connect option can be enabled or disabled during Wi-Fi initialization. If the option is enabled, the station device automatically connects to the AP once the device is initialized, otherwise the device remains idle after the initialization is complete, and connects to the AP only after calling the `wifi_config_reload_setting()` function. Auto-connect option is enabled as a default setting in the Wi-Fi driver.

1) Enable the auto-connect option.

a) Initialize the `wifi_config_ext_t` structure, as shown below. Set `config_ext.sta_auto_connect_present` to 1 to enable the presence of `config_ext.sta_auto_connect`.

```
wifi_config_ext_t config_ext;
config_ext.sta_auto_connect_present = 1;
config_ext.sta_auto_connect = 1; // Enable the auto connect.
```

b) Call the `wifi_init()` function to initialize the Wi-Fi driver, see section 2.1, "Using the Wi-Fi module in STA mode".

```
wifi_init(&config, &config_ext); // Assuming the config is already
initialized.
```

2) Disable the auto connect option.

a) Initialize the `wifi_config_ext_t` structure.

```
wifi_config_ext_t config_ext;
config_ext.sta_auto_connect_present = 1;
config_ext.sta_auto_connect = 0; // Disable the auto connect.
```

b) Call the `wifi_init()` function to initialize the Wi-Fi driver, see section 2.1, "Using the Wi-Fi module in STA mode".

```
wifi_init(&config, &config_ext); // Assuming the config is already
initialized
```

Note, if the mode is changed to STA mode by calling `wifi_config_set_opmode()`, the Wi-Fi driver will not connect to the AP router using auto connect. And after calling `wifi_config_reload_setting()`, it will start try to connect to AP.

2.1.6. Wi-Fi connection support

The connection APIs are used to manage the link status, such as, disconnect from the AP, disconnect the station, get the link status, get the station list, start/stop the scan and register an event handler for scan, connect, or disconnect events. The connection APIs use an in-band mechanism and must be called after the OS task scheduler has started, to ensure the in-band task is running.

In the STA mode, the device can disconnect from the AP, get the link status, start or stop the scan and register an event handler for scan, connect or disconnect events.

1) Get the link status.

Call the function `wifi_connection_get_link_status()` to get the link status in STA mode, as shown below.

```
uint8_t status = 0;
uint8_t link = 0;
status = wifi_connection_get_link_status(&link);
if (link == 1){
    printf("link=%d, the station is connecting to an AP router.\n", link);
}else if (link == 0) {
    printf("link=%d, the station doesn't connect to an AP router.\n",
link);
}
```

2) Disconnect the AP.

Call the function `wifi_connection_disconnect_ap()` to disconnect the station from the AP router, as shown below.

```
uint8_t status = 0;
status = wifi_connection_disconnect_ap();
```

3) Register or unregister an event handler.

The event handlers to register or unregister events are listed in Table 5. The table shows the supported events generated by the Wi-Fi driver. The events will be sent to handlers registered by the upper layer. More details on the event handlers can be found in the Wi-Fi API Reference Manual.

Table 5. Wi-Fi events

Event	Description
WIFI_EVENT_IOT_CONNECTED	• Connected event The event is triggered when authentication and association are complete. • Event payload o MAC address (6 bytes) and port number (1 byte). ▪ MAC address. The MAC address of a remote AP that the device is connected to or the MAC address of a client connected to the device when it is in AP mode. ▪ Port number indicates which port the event initiated from. ▪ 0, STA port; 1, AP port.
WIFI_EVENT_IOT_SCAN_COMPLETE	• Scan complete event. Triggered when the scan process is complete. • Event payload NULL.
WIFI_EVENT_IOT_DISCONNECTED	• Disconnected event: The device status switches from connected to disconnected. • Event payload o MAC address (6 bytes) and port number (1 byte). ▪ MAC address: The MAC address of a remote AP that the device disconnected from or the MAC address of a disconnected client. ▪ Port number indicates which port the event initiated from. ▪ 0, STA port; 1, AP port.
WIFI_EVENT_IOT_PORT_SECURE	• Secure event, mainly processed in supplicant. It can be used at the DHCP start. o In open mode, this event is triggered when authentication and association are complete. o In WPA-PSK/WPA2-PSK mode, this event is triggered when 4-way handshake (group handshake) is complete. • Event payload o MAC address (6 bytes) and port number (1 byte). ▪ MAC address: the MAC address of a remote AP that the device is connected to. ▪ Port number indicates which port the event initiated from.

Event	Description
	▪ 0, STA port; 1, AP port.
WIFI_EVENT_IOT_REPORT_BEACON_PROBE_RESPONSE	• This event is triggered when beacon or probe response is received during scan process (see the wifi_event_t enumeration). • Event payload o The beacon or probe response raw data, the length is based on the packet frame received. o The payload can be parsed by the function wifi_connection_parse_beacon(). More details can be found in the Wi-Fi API Reference Manual.
WIFI_EVENT_IOT_WPS_COMPLETE	• Credential event, triggered by the Enrollee when it successfully parses the WPS credentials in M8 message. • Event payload Represents the credentials used for WPS, the payload format follows wifi_wps_credential_info_t structure. More details please refer to Wi-Fi API Reference Manual.
WIFI_EVENT_IOT_INIT_COMPLETE	• Initialization complete event for Wi-Fi module. Triggered when supplicant initialization is complete. • Event payload o Zero data (6 bytes) and port number (1 byte). ▪ Port number indicates which port the event initiated from. ▪ 0, STA port; 1, AP port.
WIFI_EVENT_IOT_REPORT_FILTERED_FRAME	• Reports the expected management packet frame (see the wifi_event_t enumeration). Triggered after invoking the function wifi_config_set_rx_filter().The user can choose several types of frames at the same time. More details please refer to Wi-Fi API Reference Manual. • Event payload Expected management packet frame.
WIFI_EVENT_IOT_CONNECTION_FAILED	• Connection has failed. This event can only be triggered when the current password is wrong, other cases are not available yet. • Event payload o Port number (1 byte) and 2 bytes reason code. ▪ Port number indicates which port the event initiated from. ▪ 0, STA port; 1, AP port. ▪ reason code: 100 means password wrong.

An example code to describe events to register or unregister a callback for the WIFI_EVENT_IOT_CONNECTED, WIFI_EVENT_IOT_SCAN_COMPLETE, WIFI_EVENT_IOT_DISCONNECTED, WIFI_EVENT_IOT_PORT_SECURE events is provided below. Each event needs to be registered or unregistered individually.

```
/* Register event handler */
```

```
uint8_t status = 0;
int user_event_callback(wifi_event_t event_id, unsigned char *payload,
unsigned int len) /* The handler is triggered when the device connects to
an AP router*/
{
    int handled = 0;
    switch (event_id) {
    case WIFI_EVENT_IOT_CONNECTED:
        handled = 1;
        if ((len == WIFI_MAC_ADDRESS_LENGTH) && (payload)) {
          printf("[User Event Callback Sample]: LinkUp! CONNECTED MAC
            = %02x:%02x:%02x:%02x:%02x:%02x\n",
            payload[0],payload[1],payload[2],
            payload[3],payload[4],payload[5]);
        } else {
           printf("[User Event Callback Sample]: LinkUp!\n");
        }
        break;
    case WIFI_EVENT_IOT_SCAN_COMPLETE:
        handled = 1;
        printf("[User Event Callback Sample]: Scan Done!\n");
        break;
    case WIFI_EVENT_IOT_DISCONNECTED:
        handled = 1;
        if ((len == WIFI_MAC_ADDRESS_LENGTH) && (payload)) {
           printf("[User Event Callback Sample]: Disconnect! DISCONNECTED
MAC
             = %02x:%02x:%02x:%02x:%02x:%02x\n",
             payload[0],payload[1],payload[2],
             payload[3],payload[4],payload[5]);
        } else {
            printf("[User Event Callback Sample]: Disconnect!\n");
        }
        break;
   case WIFI_EVENT_IOT_PORT_SECURE:
        handled = 1;
        if ((len == WIFI_MAC_ADDRESS_LENGTH) && (payload)) {
           printf("[User Event Callback Sample]: Port Secure! CONNECTED
MAC
             = %02x:%02x:%02x:%02x:%02x:%02x\n",
             payload[0],payload[1],payload[2],
             payload[3],payload[4],payload[5]);
        } else {
            printf("[User Event Callback Sample]: Port Secure! \n");
        }
        break;
    default:
        printf("[User Event Callback Sample]: Unknown event(%d)\n",
                event_id);
        break;
}
status = wifi_connection_register_event_handler(WIFI_EVENT_IOT_CONNECTED,
                            (wifi_event_handler_t)  user_event_callback);
status =
      wifi_connection_register_event_handler(WIFI_EVENT_IOT_SCAN_COMPLETE,
                            (wifi_event_handler_t) user_event_callback);
status =
      wifi_connection_register_event_handler(WIFI_EVENT_IOT_DISCONNECTED,
                            (wifi_event_handler_t) user_event_callback);
```

```
status =
      wifi_connection_register_event_handler(WIFI_EVENT_IOT_PORT_SECURE,
                                (wifi_event_handler_t) user_event_callback);

/* Unregister event handler */
status =
      wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_CONNECTED,
                                (wifi_event_handler_t)  user_event_callback);
status =
    wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_SCAN_COMPLETE,
                                (wifi_event_handler_t) user_event_callback);
status =
    wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_DISCONNECTED,
                                (wifi_event_handler_t) user_event_callback);
status =
    wifi_connection_unregister_event_handler(WIFI_EVENT_IOT_PORT_SECURE,
                                (wifi_event_handler_t) user_event_callback);
```

2.2. Using the Wi-Fi module in AP mode

This section introduces how to initialize and configure the Wi-Fi in AP mode. A wireless AP is a device that allows wireless devices to connect to a wired network through Wi-Fi or related standards. The AP usually connects to a router as a standalone device, but it can also be an integral component of the router. An AP is different from a hotspot, which is the physical space where the wireless service is provided. The development board can be configured as an AP. There are four authentication and three encryption modes supported on the platform. This enables 11 combinations of Wi-Fi security modes, as listed below:

- Open mode
- WPA_PSK and AES mode
- WPA_PSK and TKIP mode
- WPA_PSK and AES+TKIP mode
- WPA2_PSK and AES mode
- WPA2_PSK and TKIP mode
- WPA2_PSK and AES+TKIP mode
- WPA_PSK_WPA2_PSK and AES mode
- WPA_PSK_WPA2_PSK and TKIP mode
- WPA_PSK_WPA2_PSK and AES+TKIP mode
- WEP Open mode

The supporting APIs to set the platform to AP mode are shown in Table 6.

Table 6. Configuration and connection APIs in the AP mode

API	Description
`wifi_config_set_opmode()`	This function sets the Wi-Fi operation mode and it takes effect immediately.
`wifi_config_set_ssid()`	This function sets the SSID and SSID length that Wi-Fi driver uses for a specific wireless port.

API	Description
wifi_config_set_bandwidth()	This function sets the bandwidth configuration that the Wi-Fi driver uses for a specific wireless port.
wifi_config_set_wireless_mode()	This function sets the wireless mode that the Wi-Fi driver uses for a specific wireless port.
wifi_config_set_channel()	This function sets the channel number that the Wi-Fi driver uses for a specific wireless port.
wifi_config_set_security_mode()	This function sets the authentication and encryption modes used in the Wi-Fi driver for a specific wireless
wifi_config_set_wpa_psk_key()	This function sets the password of WPA-PSK or WPA2-PSK encryption type used in the Wi-Fi driver.
wifi_config_set_pmk()	This function sets the PMK used in the Wi-Fi driver for a specific wireless port.
wifi_config_set_wep_key()	This function sets the WEP key group used in the Wi-Fi driver for a specific wireless port.
wifi_config_set_country_code()	This function sets the country code used in the Wi-Fi driver.
wifi_config_get_opmode()	This function receives the wireless operation mode of the Wi-Fi driver.
wifi_config_get_mac_address()	This function gets the MAC address of a specific wireless port used by the Wi-Fi driver.
wifi_config_get_ssid()	This function gets the SSID and SSID length of a specific wireless port used by the Wi-Fi driver.
wifi_config_get_bandwidth()	This function gets the bandwidth configuration that the Wi-Fi driver uses for a specific wireless port.
wifi_config_get_channel()	This function gets the channel number that the Wi-Fi driver uses for a specific wireless port.
wifi_config_get_wireless_mode()	This function gets the wireless mode that the Wi-Fi driver uses for a specific wireless port.
wifi_config_get_security_mode()	This function gets the authentication and encryption modes used in the Wi-Fi driver for a specific wireless
wifi_config_get_wpa_psk_key()	This function gets the password of WPA-PSK or WPA2-PSK encryption type used in the Wi-Fi driver for a specific wireless port.
wifi_config_get_pmk()	This function gets the PMK used in the Wi-Fi driver for a specific wireless port.
wifi_config_get_wep_key()	This function gets the WEP key group used in the Wi-Fi driver for a specific wireless port.
wifi_config_get_country_code()	This function gets the country code used in the Wi-Fi driver.
wifi_connection_parse_beacon()	This function parses the beacon or probe request packets.
wifi_connection_register_event_handler()	This function registers the Wi-Fi Event handler.
wifi_connection_unregister_event_handler()	This function unregisters Wi-Fi Event handler.

API	Description
`wifi_connection_disconnect_sta()`	This function disconnects the connected Wi-Fi station.
`wifi_connection_get_sta_list()`	This function gets the Wi-Fi Associated Station List.
`wifi_connection_get_max_sta_number()`	This function gets the maximum number of the stations supported in AP mode or dual mode.
`wifi_connection_get_link_status()`	This function gets the current STA port link up / link down status of the connection.

You can find more information on the APIs in the Wi-Fi API Reference Manual.

To use the device in the AP mode, apply any of the three examples described below.

2.2.1. Use the device in AP open mode

1) In this example, initialize the device in AP open mode on channel 6 with the SSID of "MTK_SOFT_AP", as shown below.

   a) Define the operation mode (`config.opmode`), SSID (`config.ap_config.ssid`), channel (`config.ap_config.channel`), authentication mode (`config.ap_config.auth_mode`) and encryption type (`config.ap_config.encrypt_type`) in the `wifi_config_t` structure, as shown below.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_AP_ONLY;
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen((char *)config.ap_config.ssid);
config.ap_config.auth_mode = WIFI_AUTH_MODE_OPEN;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_WEP_DISABLED;
config.ap_config.channel = 6;
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

2) Use configuration APIs to set the device in AP mode that operates on channel 6 in open mode with an SSID of “MTK_SOFT_AP”.

   a) Call the function `wifi_config_set_opmode(opmode)` to set the opmode to

`WIFI_MODE_AP_ONLY`, as shown below.

```
wifi_config_set_opmode(WIFI_MODE_AP_ONLY);
```

Call the function `wifi_config_set_ssid(port, ssid_name, strlen(ssid_name))` to set the SSID ("MTK_SOFT_AP") of a given port (`WIFI_PORT_AP`), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
```

   b) Call the function `wifi_config_set_security_mode(port, auth, encrypt)` to set the security mode of the AP router, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_OPEN,
WIFI_ENCRYPT_TYPE_WEP_DISABLED);
```

   c) Call the function `wifi_config_set_channel(port, channel)` to setup the channel on a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
```

Apply the configuration by calling the function `wifi_config_reload_setting()`, as shown below.

```
wifi_config_reload_setting();
```

2.2.2. Use the device in AP mode with WPA2PSK method

1) In this example, the device is in AP mode and operates on channel 6, the authentication mode is WPA2PSK (`WIFI_AUTH_MODE_WPA2_PSK`) with AES encryption type (`WIFI_ENCRYPT_TYPE_AES_ENABLED`), the password is "`12345678`" and the SSID is "`MTK_SOFT_AP`". Initialize the module in AP mode, as shown below.

   a) Define the operation mode (`config.opmode`), SSID (`config.ap_config.ssid`), channel (`config.ap_config.channel`), authentication mode (`config.ap_config.auth_mode`), encryption type (`config.ap_config.encrypt_type`) and password (`config.ap_config.password`) in the `wifi_config_t` structure, as shown below.

```
wifi_config_t config = {0};
config.opmode = WIFI_MODE_AP_ONLY;
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen("MTK_SOFT_AP");
config.ap_config.auth_mode = WIFI_AUTH_MODE_WPA2_PSK;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_AES_ENABLED;
strcpy((char *)config.ap_config.password, "12345678");
config.ap_config.password_length = strlen("12345678");
config.ap_config.channel = 6;
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, NULL);
```

The device in AP mode supports various types of security modes, as shown in Table 4. More information about the authentication mode and encryption group types can be found in `wifi_auth_mode_t` and `wifi_encrypt_type_t` types in the Wi-Fi API Reference Manual.

2) Use configuration APIs to set the device in AP mode that operates in WPA2PSK mode with a given port, channel, password and SSID.

a) Call the function `wifi_config_set_opmode(opmode)` to set the opmode to `WIFI_MODE_AP_ONLY`, as shown below.

```
wifi_config_set_opmode(WIFI_MODE_AP_ONLY);
```

Call the function `wifi_config_set_ssid(port, ssid_name, strlen(ssid_name))` to set the SSID ("`MTK_SOFT_AP`") of a given port (`WIFI_PORT_AP`), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
```

   b) Call the function `wifi_config_set_security_mode(port, auth, encrypt)` to set the security mode of the AP router, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_WPA2_PSK,
WIFI_ENCRYPT_TYPE_AES_KEY_ENABLED);
```

   c) Call the function `wifi_config_set_wpa_psk_key(port, password, strlen(password))` to set the WPA2PSK key, as shown below.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_AP, "12345678", strlen("12345678"));
```

   d) Call the function `wifi_config_set_channel(port, channel)` to setup the channel on a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
```

Apply the configuration by calling the function `wifi_config_reload_setting()`, as shown below.

```
wifi_config_reload_setting();
```

The authentication and encryption method combinations are also available (see Table 4).

2.2.3. Use the device in AP mode with open WEP method

1) In this example, the device is in AP mode and operates on channel 6, the security mode is in open WEP mode with the key index is set to 0, the password is "1234567890" and the SSID is "MTK_SOFT_AP". Initialize the module in AP mode, as shown below.

   a) Define the operation mode (`config.opmode`), SSID (`config.ap_config.ssid`), channel (`config.ap_config.channel`), authentication mode (`config.ap_config.auth_mode`), encryption type (`config.ap_config.encrypt_type`) and password (`config.ap_config.password`) in the `wifi_config_t` structure and define the key index in the `wifi_config_ext_t` (`config_ext.ap_wep_key_index`) structure, as shown below. Set `config_ext.sta_wep_key_index_present` to 1 to enable the presence of `config_ext.sta_wep_key_index`.

```
wifi_config_t config = {0};
wifi_config_ext_t config_ext = {0};
config.opmode = WIFI_MODE_AP_ONLY;
strcpy((char *)config.ap_config.ssid, "MTK_SOFT_AP");
config.ap_config.ssid_length = strlen("MTK_SOFT_AP");
config.ap_config.auth_mode = WIFI_AUTH_MODE_OPEN;
config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_WEP_ENABLED;
strcpy((char *)config.ap_config.password, "1234567890");
config.ap_config.password_length = strlen("1234567890");
config.ap_config.channel = 6;
config_ext.ap_wep_key_index_present = 1;
config_ext.ap_wep_key_index = 0;
```

   b) Call the `wifi_init()` function to initialize the Wi-Fi driver.

```
wifi_init(&config, &config_ext);
```

2) Use configuration APIs to set the device in AP mode that operates in WEP mode with the SSID of "MTK_SOFT_AP".

   a) Determine if the given WEP key (`key_string`) has a proper length, 5 or 13 ASCII characters (8 bits), 10 or 26 HEX characters (4 bits).

```
wifi_wep_key_t keys = {{{0}}};
uint8_t key_id = 0;  //0~3
char *key_string = "1234567890";
uint8_t length = strlen(key_string);
if (key_id < 4) {
          keys.wep_key_length[key_id] = length;
          os_memcpy(&keys.wep_key[key_id], key_string, length);

}
```

   b) Call the function `wifi_config_set_opmode(opmode)` to set the opmode to `WIFI_MODE_AP_ONLY`, as shown below.

```
wifi_config_set_opmode(WIFI_MODE_AP_ONLY);
```

   c) Call the function `wifi_config_set_ssid(port, ssid_name, strlen(ssid_name))` to set the SSID ("MTK_SOFT_AP") of a given port (`WIFI_PORT_AP`), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "MTK_SOFT_AP", strlen("MTK_SOFT_AP"));
```

   d) Call the function `wifi_config_set_security_mode(port, auth, encrypt)` to set the security mode of the AP router, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_OPEN,
```

```
WIFI_ENCRYPT_TYPE_WEP_ENABLED);
```

e) Call the function `wifi_config_set_wep_key(port, &keys)` to set the WEP keys.

```
wifi_config_set_wep_key(WIFI_PORT_AP, &keys);
```

f) Call the function `wifi_config_set_channel(port, channel)` to setup the channel on a given port, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 4);
```

g) Apply the configuration by calling the function `wifi_config_reload_setting()`, as shown below.

```
wifi_config_reload_setting();
```

2.2.4. Wi-Fi connection support

The APIs for connection support in AP mode:

1) Get the station list associated with the device in AP mode.

Call the function `wifi_connection_get_sta_list()` to get the station's associated AP information. An example implementation is shown below.

```
uint8_t i;
uint8_t status = 0;
wifi_sta_list_t list[WIFI_MAX_NUMBER_OF_STA];
uint8_t size = 0;
status = wifi_connection_get_sta_list(&size, list);
printf("stalist size=%d\n", size);
for (i = 0; i < size; i++)
{
    printf("%d\n", i);
    printf("last_tx_rate: MCS=%d, LDPC=%d, MODE=%d\n",
           (list[i].last_tx_rate.field.mcs),
           (list[i].last_tx_rate.field.ldpc),
           (list[i].last_tx_rate.field.mode));
    printf("last_rx_rate: MCS=%d, LDPC=%d, MODE=%d\n",
           (list[i].last_rx_rate.field.mcs),
           (list[i].last_rx_rate.field.ldpc),
           (list[i].last_rx_rate.field.mode));
    printf("rssi_sample.LastRssi0)=%d\n",
           (int)(list[i].rssi_sample.last_rssi));
    printf("rssi_sample.AvgRssi0X8=%d\n",
           (int)(list[i].rssi_sample.average_rssi));
    printf("addr=%02x:%02x:%02x:%02x:%02x:%02x\n", list[i].mac_address[0],
           list[i].mac_address[1],  list[i].mac_address[2],
           list[i].mac_address[3],  list[i].mac_address[4],
           list[i].mac_address[5]);
    printf("power_save_mode=%d\n",
           (unsigned int)(list[i].power_save_mode));
    printf("bandwidth=%d\n", (unsigned int)(list[i].bandwidth));
    printf("keep_alive=%d\n", (unsigned int)(list[i].keep_alive));
}
```

2) Get the maximum number of stations the AP can support.

Call the `wifi_connection_get_max_sta_number()` function to get the maximum number of stations supported in AP mode and dual mode.

```
uint8_t status = 0;
uint8_t number = 0;
status = wifi_connection_get_max_sta_number(&number);
printf("wifi_connect_get_max_station_number_ex: max sta number=%d,
       status=%d\n", number, status);
```

3) Register or unregister event handler.

4) To register or unregister a callback function for an event, call the function `wifi_connection_register_event_handler()` or `wifi_connection_unregister_event_handler()`, respectively. Each event needs to be registered or unregistered individually. These functions are already described in the station mode, see section 2.1.6, “Wi-Fi connection support”.

2.3. Using the Wi-Fi module in a repeater mode

The repeater provides two virtual ports, one is the AP port, and the other is AP Client Port. The repeater can connect to the router to get the IP address. Multiple stations can connect to the repeater using the repeater’s SSID and each station can get IP address from the router. The data packets can transfer between station and router through the repeater.

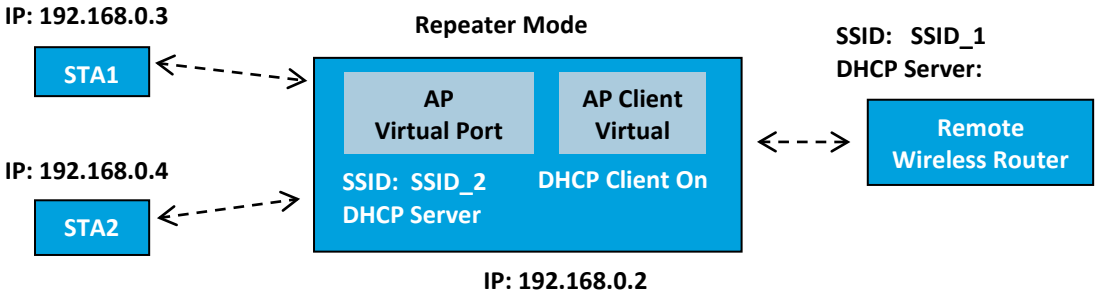

Figure 2. Repeater mode topology

The STA communicates with the router through the repeater. The repeater should connect to the router first. Then the stations can get IP address from the router. The remote AP also provides the DHCP server and can assign the IP address to the stations.

2.3.1. Features in the repeater mode

The channel and bandwidth use the same settings for both AP Port and AP Client (APCLI) Port. The other settings are independent, such as MAC address and security mode.

- The DHCP server is at the remote AP, not at the repeater.
- The MAC address of the AP and APCLI must be different.
- The AP and APCLI should have the same bandwidth in repeater mode.
- The AP and APCLI should stay on the same channel as the remote AP.
- The repeater mode doesn’t support WPS.
- The repeater mode doesn’t support power saving option.

The following Wi-Fi APIs cannot operate in repeater mode, as shown in Table 7.

Table 7. The functions not supported in repeater mode

API	API
`wifi_wps_connection_by_pin()`	`wifi_config_set_dtim_interval()`
`wifi_wps_config_set_device_info()`	`wifi_config_get_dtim_interval()`
`wifi_wps_config_get_device_info()`	`wifi_config_set_listen_interval()`
`wifi_wps_config_set_auto_connect()`	`wifi_config_get_listen_interval()`
`wifi_wps_config_get_auto_connect()`	`wifi_wps_config_get_pin_code()`
	`wifi_wps_connection_by_pbc()`

There are two methods to configure the Wi-Fi module in repeater mode:

- Using the static method to initialize the repeater mode at system reboot with the `wifi_init()` API.
- Using the dynamic method to provide Wi-Fi settings in the configuration APIs. The new settings take effect once the APIs reload.

2.3.2. Using the static method

Before applying this method, a few assumptions to make:

- The repeater operates on channel 6 with allocated bandwidth of 20MHz.
- The AP port is configured with an SSID of "`SSID_2`", the authorization mode is WPA-PSK, AES encrypted, and the password is "`87654321`".
- The AP Client port is configured with an SSID of "SSID_1", the authorization mode is WPA/WPA2-PSK and TKIP+AES encrypted and the password is "12345678".

1) Define a configuration structure.

```
wifi_config_t config = {0};
```

2) Set the parameters of the `wifi_config_t` structure.

```
  config.opmode = WIFI_MODE_REPEATER;
  strcpy((char *)config.sta_config.ssid, "SSID_1");
  config.sta_config.ssid_length = strlen("SSID_1");
  strcpy((char *)config.sta_config.password, "12345678");
  config.sta_config.password_length = strlen("12345678");
  strcpy((char *)config.ap_config.ssid, "SSID_2");
  config.ap_config.ssid_length = strlen("SSID_2");
  config.ap_config.auth_mode = WIFI_AUTH_MODE_WPA_PSK;
  config.ap_config.encrypt_type = WIFI_ENCRYPT_TYPE_AES_ENABLED;
  strcpy((char *)config.ap_config.password, "87654321");
  config.ap_config.password_length = strlen("87654321");
  config.ap_config.channel = 6;
  config.ap_config.bandwidth = WIFI_IOT_COMMAND_CONFIG_BANDWIDTH_20MHZ;
```

3) Apply the settings by calling the `wifi_init()` function.

```
  wifi_init(&config, NULL);
```

2.3.3. Using the dynamic method

Before applying this method, few assumptions to make:

- The repeater operates on channel 6 with allocated bandwidth of 20MHz.
- The AP port is configured with an SSID of "SSID_2", the authorization mode is WPA-PSK and the encryption type is AES, the password is "87654321".
- The AP Client port is configured with an SSID of "SSID_1", the authorization mode is WPA/WPA2-PSK and the encrypt type is TKIP+AES, the password is "12345678".

1) Call the function `wifi_config_set_opmode(opmode)` to set the opmode.

```
wifi_config_set_opmode(WIFI_MODE_REPEATER);
```

2) To configure the APCLI port in repeater mode:

   a) Call the function `wifi_config_set_ssid(port_apcli, ssid_apcli, strlen(ssid_apcli))` to set the SSID of the AP Client port, as shown below.

```
wifi_config_set_ssid(WIFI_PORT_APCLI, "SSID_1", strlen("SSID_1"));
```

   b) Call the function `wifi_config_set_wpa_psk_key(port_apcli, password_apcli,strlen(password_apcli))` to set the APCLI port password.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_APCLI, "12345678",
                                    strlen("12345678"));
```

3) To configure the AP port in repeater mode:

   a) Call the function `wifi_config_set_ssid(port_ap, ssid_ap, strlen(ssid_ap))` to set the SSID ("SSID_2") of a given port (`WIFI_PORT_AP`), as shown below.

```
wifi_config_set_ssid(WIFI_PORT_AP, "SSID_2", strlen("SSID_2"));
```

   b) Call the function `wifi_config_set_security_mode(port_ap, auth_mode, encrypt_mode)` to set authentication mode and encryption type of the AP port, as shown below.

```
wifi_config_set_security_mode(WIFI_PORT_AP, WIFI_AUTH_MODE_WPA_PSK,
                                WIFI_ENCRYPT_TYPE_AES_ENABLED);
```

4) Call the function `wifi_config_set_wpa_psk_key(port_ap, password_ap, strlen(password_ap))` to set the WPA2/PSK key, as shown below.

```
wifi_config_set_wpa_psk_key(WIFI_PORT_AP, "87654321", strlen("87654321"));
```

5) Call the function `wifi_config_set_channel(port_ap, channel)` to set the AP channel, as shown below.

```
wifi_config_set_channel(WIFI_PORT_AP, 6);
```

6) Apply the settings by calling the `wifi_config_reload_setting()` function.

```
wifi_config_reload_setting();
```

2.4. Scan

Scan modes and scan options are selected when scan function is executed. The modes and options are shown in Table 8 and Table 9. The Wi-Fi module supports scan mode and option combinations in the AP and station modes. The station mode supports all scan modes and options and the AP mode supports only the partial scan mode.

Table 8. Scan mode

Value	Definition

Value	Definition
0	Full scan
1	Partial scan, it's expected to use in SoftAP mode and keep stations online.

Table 9. Scan option

Value	Definition
0	Active scan. The channels to be scanned are decided by the product's country and region. A probe request will be sent to each of these channels. Not applied on regulatory channels.
1	Passive scan in all channels. Receives and processes the beacon. No probe request is sent.
2	Forced active scan. Forced active scan in all channels. A probe request is sent to each channel.

Table 10 and Table 11 provide the scanned channel list in different country regions and bands.

Table 10. Channels supporting 2.4G

Region ID	Channel
0	CH1-11 active scan
1	CH1-13 active scan
2	CH10-11 active scan
3	CH10-13 active scan
4	CH14 active scan
5	CH1-14 all active scan
6	CH3-9 active scan
7	CH5-13 active scan
31	CH1-11 active scan, CH12-14 passive scan
32	CH1-11 active scan, CH12-13 passive scan
33	CH1-14 all active scan

Table 11. Channels supporting 5G

Region ID	Channel List
0	CH36-64 active scan, CH149-165 active scan
1	CH36-64 active scan, CH100-140 active scan
2	CH36-64 active scan
3	CH52-64 active scan, CH149-161 active scan
4	CH149-165 active scan
5	CH149-161 active scan
6	CH36-48 active scan
7	CH36-64 active scan, CH100-140 active scan, CH149-165 active scan
8	CH52-64 active scan
9	CH36-64 active scan, CH100-116 active scan, CH132-140 active scan, CH149-165 active scan
10	CH36-48 active scan, CH149-165 active scan
11	CH36-64 active scan, CH100-120 active scan, CH149-161 active scan
12	CH36-64 active scan, CH100-140 active scan

Region ID	Channel List
13	CH52-64 active scan, CH100-140 active scan, CH149-161 active scan
14	CH36-64 active scan, CH100-116 active scan, CH136-140 active scan, CH149-165 active scan
15	CH149-173 active scan
16	CH52-64 active scan, CH149-165 active scan
17	CH36-48 active scan, CH149-161 active scan
18	CH36-64 active scan, CH100-116 active scan, CH132-140 active scan
19	CH56-64 active scan, CH100-140 active scan, CH149-161 active scan
20	CH36-64 active scan, CH100-124 active scan, CH149-161 active scan
21	CH36-64 active scan, CH100-140 active scan, CH149-161 active scan
22	CH36-64 active scan
30	CH36-48 active scan, CH52-64 passive scan, CH100-140 passive scan, CH149-165 active scan
31	CH52-64 passive scan, CH100-140 passive scan, CH149-165 active scan
32	CH36-48 active scan, CH52-64 passive scan, CH100-140 passive scan, CH149-161 active scan
33	CH36-48 active scan, CH52-64 passive scan, CH100-140 passive scan
34	CH36-48 active scan, CH52-64 passive scan, CH149-165 active scan
35	CH36-48 active scan, CH52-64 passive scan
36	CH36-48 active scan, CH100-140 passive scan, CH149-165 active scan
37	CH36-48 active scan, CH52-64 passive scan, CH149-165 active scan, CH173 active scan

2.4.1. Scan APIs

APIs to set the platform to AP mode are provided for your reference (see Table 12).

Table 12. Scan APIs

API	Definition
`wifi_connect_scan_init (wifi_scan_list_item_t *ap_list, uint32_t max_count)`	This function should be called before calling the function `wifi_connection_start_scan()`, and it should be called only once to initialize the scan to store the scanned AP list in the `ap_list` buffer in descending order of the RSSI, otherwise the list won't be stored.
`wifi_connection_start_scan (uint8_t *ssid, uint8_t ssid_length, uint8_t *bssid, uint8_t scan_mode, uint8_t scan_option)`	This function starts the Wi-Fi Scanning based on scan mode and scan option (see Table 8 and Table 9). The station mode supports all scan modes and options. The AP mode supports only the partial scan mode.
`wifi_connection_stop_scan (void)`	This function stops the Wi-Fi Scanning triggered by `wifi_connection_start_scan()`.
`wifi_connection_scan_deinit (void)`	This function de-initializes the scan table. When the scan is finished, `wifi_connection_scan_deinit()` should be called to unload the buffer from the driver. After that, the data in the `ap_list` can be processed by user applications safely, and then another scan can be initialized by calling `wifi_connection_scan_init()`. Not calling the `wifi_connection_scan_deinit()` function, may cause failure if another task attempts to call

API	Definition
	wifi_connection_scan_init().

The example implementation below uses full scan and active scan methods. Apply this as a reference for other scan mode and scan option implementations. There are two approaches to get the scan result, one is to register an event callback and parse the Wi-Fi beacon and probe-request raw packet and the other method applies a scan table initialized through `wifi_connection_scan_init()`.

2.4.1.1. Using the `wifi_connection_scan_init()` API

1) Define a buffer `g_ap_list` to store the scan results.

```
uint8_t size = 30;
wifi_scan_list_item_t g_ap_list[size] = {0};
```

2) Call the function `wifi_connection_scan_init()` to initialize the buffer in the Wi-Fi module. The scan result is recorded in `g_ap_list` in descending order of the RSSI. The number of APs to detect is limited by `size`, which is `30` in this example.

```
wifi_connection_scan_init(g_ap_list, size);
```

3) Call the function `wifi_connection_start_scan()` to start the scan.

```
wifi_connection_start_scan(NULL, 0, NULL, 0, 0);
```

4) Call the function `wifi_connection_stop_scan()` to stop the scan.

```
wifi_connect_stop_scan();
```

5) Call the function `wifi_connection_scan_deinit()` to de-initialize the scan and to unload the buffer from the driver.

```
wifi_connect_scan_deinit();
```

Note 1. The scan table `g_ap_list` is a user-defined array initialized in`wifi_connection_scan_init()`. It's required to call `wifi_connection_scan_deinit()` after the scan process is complete. Otherwise, calling the function `wifi_connection_scan_init()` again will fail.

Note 2. The information of the scanned APs is stored in the scan table in descending order of the RSSI.

Note 3. It is recommended to call `wifi_connection_scan_deinit()` first to release the scan table from the scan process, to give the ownership of the scan table back to the user. Then the user can safely access the information stored in the scan table, erase the table or free the memory allocated to the table.

Note 4. Call the function `wifi_connection_scan_deinit()` after the scan completes or after the scan has stopped by `wifi_connect_stop_scan()`.

2.4.1.2. Using a callback to parse the Wi-Fi beacon and probe-response raw packet

1) Register the event handler `WIFI_EVENT_IOT_REPORT_BEACON_PROBE_RESPONSE`. The beacon and probe-response raw packet in the air can be received and uploaded to the handler.

```
uint8_t status = 0;
int event_handler_sample(wifi_event_t event_id, unsigned char *payload,
unsigned int len)
{
```

```
    int handled = 0;
    if (event_id == WIFI_EVENT_IOT_REPORT_BEACON_PROBE_RESPONSE){
        handled = 1;
        if (len != 0) {
          wifi_scan_list_item_t ap_data;
          os_memset(&ap_data, 0, sizeof(wifi_scan_list_item_t));
          if (wifi_connection_parse_beacon(payload, len, &ap_data) >= 0) {
          printf("\n%-4s%-33s%-20s%-8s%-8s%-8s%-8s\n", "Ch", "SSID",
"BSSID", "Auth",
           "Cipher", "RSSI", "WPS");
          printf("%-4d", ap_data.channel);
          printf("%-33s", ap_data.ssid);
          printf("%02x:%02x:%02x:%02x:%02x:%02x   ",
          ap_data.bssid[0],
          ap_data.bssid[1],
          ap_data.bssid[2],
          ap_data.bssid[3],
          ap_data.bssid[4],
          ap_data.bssid[5]);
          printf("%-8d", ap_data.auth_mode);
          printf("%-8d", ap_data.encrypt_mode);
          printf("%-8d", ap_data.rssi);
          printf("%-8d", ap_data.wps);
          printf("\n");
    }
    }
    return handled;
}
status=wifi_connection_register_event_handler(WIFI_EVENT_IOT_REPORT_BEACON
_PROBE_RESPONSE, (wifi_event_handler_t) event_handler_sample);
```

2) Call the function `wifi_connection_start_scan()` to start scanning. During the scan, the beacon and probe response in the air can be received and uploaded to `event_handler_sample()` registered in the previous step.

```
wifi_connection_start_scan(NULL, 0, NULL, 0, 0);
```

3) Call the function `wifi_connection_stop_scan()` to stop or cancel the scan.

```
wifi_connect_stop_scan();
```

Note 1. The handler registered with the event `WIFI_EVENT_IOT_SCAN_COMPLETE` is triggered when the channel list is fully scanned or the function `wifi_connect_stop_scan()` is called.

Note 2. When the device in station mode is not connected with an AP, it'll continue the background scan until connection is established.

2.5. Smart Connection

Airoha Smart Connection configures the device without an input interface to connect to the wireless network. The Smart Connection broadcasts the encrypted wireless network information (SSID and password of the AP router) through SmartConnection, an Android application package (APK) built with the Smart Connection library, so that the IoT device based on the Airoha IoT development platformcan listen and decode the information to connect to the specified wireless network, as shown in Figure 3. The broadcast information is encrypted with a pre-defined key to ensure security.

The APK can be found in the <sdk_root>/tools/wifi_smart_connection folder, where the Smart Connection libraries for both Android and iOS are included. Users can also build custom Android or iOS applications with the Smart Connection libraries.

Note, the HDK supports versions 4 and 5 of the Smart Connection protocol. Version 5 is enabled by default.

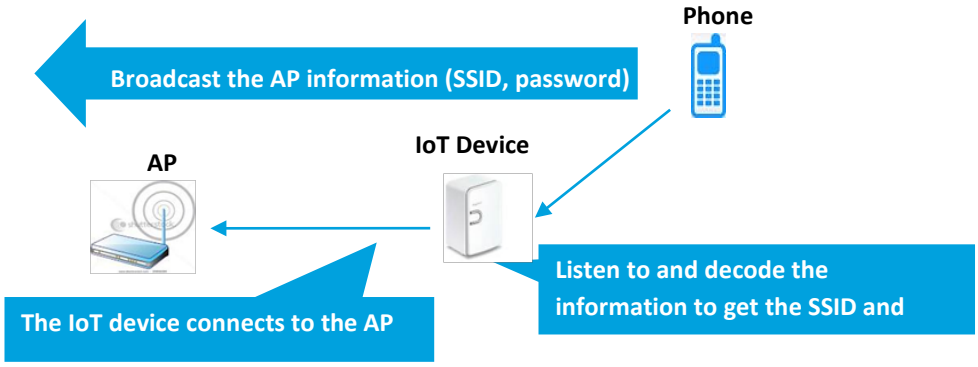

Figure 3. Communicating with an IoT device using Smart Connection

2.5.1. Smart Connection APIs

The Smart Connection APIs are provided in Table 13 for your reference.

Table 13. Smart Connection APIs

API	Definition

`wifi_smart_connection_init(`
`const uint8_t *key,`
`const uint8_t key_length,`
`wifi_smart_connection_callback_t callback)`
	This function sets a decryption key and registers a callback function to establish the Smart Connection. The callback function handles the connection result.

`wifi_smart_connection_deinit()`	This function de-initializes the smart connection and releases the allocated resources.
`wifi_smart_connection_start(` `uint16_t timeout_seconds)`	This function starts the Smart Connection. Waiting for the broadcast message. The timeout specifies the duration the device listens to the broadcast message. The Smart Connection is considered as failed, if timeout is reached.

`wifi_smart_connection_get_result(`
`uint8_t *ssid,`
`uint8_t *ssid_length,`
`uint8_t *password,`
`uint8_t *password_length,`
`uint8_t *tlv_data,`
`uint8_t *tlv_data_length)`
	This function gets the wireless network information (SSID and password) collected during the Smart Connection. The type length value (TLV) data is user-defined data sent from the Smart Connection mobile application.

`wifi_smart_connection_stop()`	This function stops the Smart Connection.

2.5.2. How to use the Smart Connection API

1) Call the function `wifi_smart_connection_init()` to initialize the Smart Connection and then call the function `wifi_smart_connection_start()` to start the Smart Connection process.

```
int32_t user_smart_connection(void)
{
    if(wifi_smart_connection_init(NULL, 0, smart_connection_event_handler)
< 0){
        return -1;
    }

    wifi_smart_connection_start(0);
    return 0;
}
```

2) Define the `wifi_smart_connection_callback_t` callback function to process a Smart Connection event.

When the WIFI_SMART_CONNECTION_EVENT_INFO_COLLECTED event is received, apply the function `wifi_smart_connection_get_result()` to get the SSID and the password. Then proceed with the rest of the implementation, such as calling the function `wifi_config_reload_setting()` to connect to the target AP or to store the received information into flash.

```
void smart_connection_event_handler(wifi_smart_connection_event_t event,
void *data)
{
    uint8_t password[WIFI_LENGTH_PASSPHRASE + 1] = {0};
    uint8_t ssid[WIFI_MAX_LENGTH_OF_SSID + 1] = {0};
    uint8_t ssid_len = 0;
    uint8_t password_len = 0;

    switch (event)
    {
        case WIFI_SMART_CONNECTION_EVENT_CHANNEL_LOCKED:
        case WIFI_SMART_CONNECTION_EVENT_TIMEOUT:
            break;
        case WIFI_SMART_CONNECTION_EVENT_INFO_COLLECTED:
            wifi_smart_connection_get_result(ssid, &ssid_len, password,
                                             &password_len, NULL, NULL);
            /* Configure the STA with the received SSID & passward. */
            wifi_config_set_ssid(WIFI_PORT_STA, ssid, ssid_len);
            if(password_len > 0) {
                /* If the target AP is WPA-PSK or WPA2-PSK. */
                wifi_config_set_wpa_psk_key(WIFI_PORT_STA, password,
                                            password_len);
            }
            wifi_config_reload_setting();
            wifi_smart_connection_deinit();
            break;
    }
}
```

3) Call the function `wifi_smart_connection_stop()` to stop the Smart Connection:

```
int32_t user_smart_exit(void)
{
```

```
    wifi_smart_connection_stop();
    return 0;
}
```

2.6. Monitor mode and RX filter

Monitor mode is the mode in which platform can receive all the packets transmitted over the air, call

`wifi_config_set_opmode()` to configure the platform in the monitor mode.

The RX filter, which only works in station mode, is able to filter out user-defined packet types. Set the Rx filter, as shown in section 2.6.1, “How to use the RX filter API”.

2.6.1. How to use the RX filter API

Set the RX filter control register (RFCR) to configure the behavior of receiving Wi-Fi packets by calling the `wifi_config_set_rx_filter()` API. The RX filter settings indicate the desired packet format to receive, such as broadcast, multicast and unicast frames, as shown in Table 14.

Table 14. RX filter definitions

Input Parameter	Description
WIFI_RX_FILTER_DROP_STBC_BCN_BC_MC	bit 0, drops STBC beacon/BC/MC frames.
WIFI_RX_FILTER_DROP_FCS_ERR	bit 1, drops FCS error frames.
WIFI_RX_FILTER_RESERVED	bit 2, a reserved bit, not used.
WIFI_RX_FILTER_DROP_VER_NOT_0	bit 3, drops version field of Frame Control field. It cannot be 0.
WIFI_RX_FILTER_DROP_PROBE_REQ	bit 4, drops probe request frame.
WIFI_RX_FILTER_DROP_MC_FRAME	bit 5, drops multicast frame.
WIFI_RX_FILTER_DROP_BC_FRAME	bit 6, drops broadcast frame.
WIFI_RX_FILTER_RM_FRAME_REPORT_EN	bit 12, enables report frames.
WIFI_RX_FILTER_DROP_NOT_MY_BSSID	bit 17, drops not my BSSID frames.
WIFI_RX_FILTER_DROP_NOT_UC2ME	bit 18, drops not unicast to me frames.
WIFI_RX_FILTER_DROP_DIFF_BSSID_BTIM	bit 19, drops different BSSID TIM Broadcast frame.
WIFI_RX_FILTER_DROP_NDPA	bit 20, drops the Null Data Packet Announcement (NDPA) or not.

The input parameter of `wifi_config_set_rx_filter()` API indicates the bits to configure the RX filter options. The details for each bit can be found in the Wi-Fi API Reference Manual.

2.6.2. How to receive raw packets

In order to process the raw packets, register a raw packet handler to process the packets with `wifi_config_register_rx_handler()`.

```
int smtcn_start(void)
{
    ……
  wifi_config_register_rx_handler(
              (wifi_rx_handler_t) wlan_raw_pkt_rx_filter_sample);
    ……
```

```
}
```

Note that besides the `wifi_config_set_rx_filter()` API, you can also use `wifi_config_set_opmode()` to configure the runtime settings in a monitor mode to receive all the packets transmitted over the air.

To filter the specified packets, configure the runtime settings in the station mode and use the RX filter.

An example to set the RX filter is shown below.

```
int smtcn_start(void)
{
    ……
    wifi_config_set_opmode(WIFI_MODE_STA_ONLY);
    ……
    uint32_t rx_filter;
    wifi_config_get_rx_filter(&origin_rxfilter);
    rx_filter = origin_rxfilter | BIT(WIFI_RX_FILTER_RM_FRAME_REPORT_EN);
    rx_filter &= ~BIT(WIFI_RX_FILTER_DROP_NOT_MY_BSSID);
    rx_filter &= ~BIT(WIFI_RX_FILTER_DROP_NOT_UC2ME);
    rx_filter &= ~BIT(WIFI_RX_FILTER_DROP_MC_FRAME);
    wifi_config_set_rx_filter(rx_filter);
    ……
}
```

2.7. WPS

Wi-Fi Protected Setup (WPS) is a network security standard to create a secure wireless home network and is used to simplify the security setup and management of Wi-Fi networks. It includes two types of configuration, in-band and out-of-band. The Airoha IoT SDK uses in-band configuration that supports Push Button Connection (PBC) and Personal Identification Number (PIN) methods.

In PBC method of WPS, the user has to push a button, either an actual or virtual one, on the access point and the new wireless client device. On most devices, this discovery mode turns itself off as soon as a connection is established or after a delay (typically 2 minutes or less), whichever comes first, thereby minimizing its vulnerability. Support of this mode is mandatory for access points and optional for connecting devices. The Wi-Fi Direct specification supersedes this requirement by stating that all devices must support the push button method.

In PIN method of the WPS, the wireless PIN can be found on the device label or display. This PIN must then be entered at the representant (the registrar) of the network, usually the network's access point. Alternately, a PIN provided by the access point may be entered into the new device.

A device seeking to join a wireless network is called Enrollee. A device with the authority to issue and revoke access to a network is called Registrar.

The Airoha IoT SDK provides APIs to modify default device configuration before triggering the WPS and new event handler to support credential callbacks. To set the WPS, apply the following steps.

- Set device information.
- Turn off WPS auto connection.
- Register credential event handler.
- Trigger WPS.

The device settings for WPS including Device Name, Manufacturer, Model Name, Model Number and Serial Number can only be modified before triggering the WPS.

Once the WPS Enrollee is assigned, it'll get new credentials from WPS Registrar. And, if the WPS auto-connection is set on, it will use the new credentials to connect to the Wi-Fi.

The Airoha IoT SDK supported features for WPS:

- STA and Enrollee PBC method.
- STA and Enrollee PIN method.
- AP and Registrar PBC method.
- AP and Registrar PIN method.
- AP and Enrollee PIN method.

Note, AP and Enrollee PIN method is only supported in MT7687 and MT7697.

The list of APIs to implement PBC and PIN methods is shown in Table 15.

Table 15. WPS APIs

Function	Description
`wifi_wps_config_set_device_info()`	This function sets the device information for WPS. Call this function before triggering WPS connection.
`wifi_wps_config_get_device_info()`	This function gets the device information for WPS. Call this function to check the device information that's already set.
`wifi_wps_config_set_auto_connection()`	This function sets the WPS auto connection after receiving the credentials. Set it to 1 to turn on the WI-FI connection after receiving the credentials and 0, otherwise.
`wifi_wps_config_get_auto_connection()`	This function gets the WPS auto connection control status. 0, turn off, 1, turn on
`wifi_connection_register_event_handler()`	This function registers the Wi-Fi event handler. The event is `WIFI_EVENT_IOT_WPS_COMPLETE`.
`wifi_wps_unregister_credential_handler()`	This function unregisters Wi-Fi event handler. The event is `WIFI_EVENT_IOT_WPS_COMPLETE`.
`wifi_wps_connection_by_pbc()`	This function triggers the WPS PBC process. AP must act as Registrar and STA must act as Enrollee.

Function	Description
`wifi_wps_connection_by_pin()`	This function triggers the WPS PIN process. AP can act as Registrar or Enrollee, STA should act as Enrollee. If the AP is an Enrollee, get the AP's PIN and provide it to the STA. The WPS PIN process of the AP will be automatically triggered by the STA.
`wifi_wps_config_get_pin_code()`	This function gets the PIN code of a specific wireless port. The PIN may be different each time the function is called in the STA mode.

More information can be found in the Wi-Fi API Reference Manual.

2.7.1. Using the APIs

To apply the Wi-Fi WPS APIs:

1) Call `wifi_wps_config_set_device_info()`, to set up the WPS device configuration information.

An example implementation shows how to set the device information.

```
wifi_wps_device_info_t input_devinfo;

int32_t wps_set_device_info_example()
{
    char * device_name   = "MTK IoT Test";
    char * manufacturer  = "MTK Test";
    char * model_name    = "MTK Wireless Device Test";
    char * model_number  = "MTK7687 Test";
    char * serial_number = "87654321";

    os_memset(&input_devinfo, 0, sizeof(input_devinfo));
    //copy device_name
    if (strlen(device_name) < WIFI_WPS_DEVICE_NAME_LENGTH)
      os_memcpy(input_devinfo.device_name,
                device_name,strlen(device_name));
    else {
      LOG_E(minisupp, "device_name length is too long\n");
      return -1;
    }
    //copy manufacturer
    if (strlen(manufacturer) < WIFI_WPS_MANUFACTURER_LENGTH)

os_memcpy(input_devinfo.manufacturer,manufacturer,strlen(manufacturer));
    else {
      LOG_E(minisupp, "manufacture length is too long\n");
      return -1;
    }
    //Set model_name as manufacturer
    //Set model_number as manufacturer
    //Set serial_number as manufacturer
    ......

    return wifi_wps_config_set_device_info(&input_devinfo);
```

```
}
```

2) Call the function `wifi_wps_config_get_device_info()` to retrieve the device information, once it's already set.

An example implementation shows how to get device information.

```
wifi_wps_device_info_t output_devinfo;

int32_t wps_get_device_info_example()
{
    os_memset(&output_devinfo, 0, sizeof(output_devinfo));

    return wifi_wps_config_get_device_info(&output_devinfo);
}
```

3) Call `wifi_wps_config_set_auto_connection()` to disable WPS auto-connection.

An example implementation to disable the auto-connection is shown below.

```
int32_t wps_disable_auto_connection_example()
{
    bool auto_conn = false;
    return wifi_wps_config_set_auto_connection(auto_conn);
}
```

2.7.1.1. Get WPS auto-connection control status.

1) Call the function `wifi_wps_config_get_auto_connection()` to retrieve the auto-connection control status.

An example implementation is shown below.

```
int32_t wps_get_auto_connection_status_example()
{
    bool auto_conn  = false;
    return wifi_wps_config_get_auto_connection(&auto_conn);
}
```

2.7.1.2. Register or unregister Wi-Fi WPS event handler

To register Wi-Fi WPS credential handler:

1) Call `wifi_connection_register_event_handler()`.

An example implementation for the event handler is shown below.

```
int32_t wps_register_credential_event_handler_example()
{
   return wifi_connection_register_event_handler(
            WIFI_EVENT_IOT_WPS_COMPLETE, (wifi_event_handler_t)
                              wifi_wps_credential_event_handler_example);
}
```

An example implementation for the event handler `wifi_wps_credential_event_handler_example()` is shown below.

```
int32_t int32_t wifi_wps_credential_event_handler_example(
                                    wifi_event_t event,
                                    uint8_t *payload, uint32_t length)
{
    wifi_wps_credential_information_t *customer_cred = payload;

    if (event = WIFI_EVENT_IOT_WPS_COMPLETE) {
      LOG_I(minisupp, "WPS: ssid--%s\n", customer_cred->ssid);
      LOG_I(minisupp, "WPS: ssid len--%d\n", customer_cred->ssid_len);
      LOG_I(minisupp, "WPS: auth_mode-- %d\n" customer_cred->auth_mode);
      LOG_I(minisupp, "WPS: encr_mode-- %d\n" customer_cred->encr_mode);
      LOG_I(minisupp, "WPS: key_length--%d\n", customer_cred->key_length);
      ......
      ......

      wifi_wps_unregister_credential_handler();
      return 0;
}
```

To unregister Wi-Fi WPS credential event handler.

1) Call the function `wifi_connection_unregister_event_handler()` to unregister the credential handler. An example implementation is shown below.

```
int32_t wps_unregister_credential_event_handler_example()
{
   return wifi_connection_unregister_event_handler(
                  WIFI_EVENT_IOT_WPS_COMPLETE, (wifi_event_handler_t)
                              wifi_wps_credential_event_handler_example);
}
```

2.7.2. Wi-Fi WPS connection support

Trigger Wi-Fi WPS connection using PBC (see section 2.7.2.1, "WPS PBC method") or PIN (see section 2.7.2.2, "WPS PIN method") method.

Call the function `wifi_wps_unregister_credential_handler()`, to unregister the credential handler, as described in section 2.7.1.2, “Register or unregister Wi-Fi WPS event handler”.

2.7.2.1. WPS PBC method

There are two cases when using the WPS PBC method to connect to the Wi-Fi network:

- The AP as a Registrar.
- The STA as an Enrollee.

Call the function `wifi_wps_connection_by_pbc()` to apply the API directly.

1) AP as a Registrar.

Call the function `wifi_wps_connection_by_pbc()` with an input AP port defined as WIFI_PORT_AP.

The AP can only act as Registrar in the WPS PBC method. The `bssid` parameter is ignored when the function `wps_ap_pbc_example()` is called.

An example implementation is shown below.

```
int32_t wps_ap_pbc_example()
{
    uint8_t *bssid = "00:11:22:33:44:55";

    return wifi_wps_connection_by_pbc(WIFI_PORT_AP, NULL);
}
```

After the AP calls this API, the Enrollee, such as a smart phone, also needs to push the WPS virtual button to trigger the Enrollee process. During 120 seconds, the Enrollee will connect with the AP and get an IP, or prompt fail.

2) STA as an Enrollee.

When the device is in station mode, define the port directly in the `wifi_wps_connection_by_pbc()` function (`WIFI_PORT_STA`), the STA can only act as an Enrollee in PBC method.

When STA calls the `wifi_wps_connection_by_pbc()` API, the parameter `bssid` shouldn't be NULL and should be set to the Registrar's BSSID, otherwise, if two APs push the WPS button at the same time, the STA's WPS may fail.

After the STA calls `the wifi_wps_connection_by_pbc()` API, the Registrar, such as an AP, also needs to push the WPS button, either an actual or virtual one, to trigger the Registrar process. During 120 seconds, the Enrollee will connect with the AP and get an IP, or prompt fail.

An example implementation is shown below.

```
int32_t wps_sta_pbc_example()
{
    uint8_t *bssid = "00:11:22:33:44:55";

    return wifi_wps_connection_by_pbc(WIFI_PORT_STA,  bssid);
}
```

2.7.2.2. WPS PIN method

There are three cases when using the WPS PIN method to connect to the Wi-Fi network:

- The access point as a Registrar.
- The access point as an Enrollee.
- The STA as an Enrollee.

Call the function `wifi_wps_connection_by_pin()` to apply the API directly.

1) AP as a Registrar

When the device is AP, assign the parameter `WIFI_PORT_AP` to the "port" parameter of this API, and then the AP will operate as a Registrar. The STA will operate as an Enrollee.

First of all, AP needs to know the STA's PIN code and we manually give STA's PIN to AP.

Before the AP calls the `wifi_wps_connection_by_pin()` API, the Enrollee, such as a smart phone, needs to push the WPS PIN virtual button to get a PIN code and trigger the Enrollee process. Then input this PIN code to AP and call the `wifi_wps_connection_by_pin()` API using Enrollee's PIN to trigger AP's Registrar process. During 120 seconds (based on the official WPS specification), the Enrollee will connect with the AP and get an IP, or prompt fail.

An example implementation is shown below.

```
int32_t wps_ap_pin_example()
```

```
{
    uint8_t *bssid = "00:11:22:33:44:55";
    uint8_t *sta_pin = "00309448";

    return wifi_wps_connection_by_pin(WIFI_PORT_AP, bssid, sta_pin);
}
```

2) AP as an Enrollee

When the device is AP as an Enrollee, the process is passive and there is no command or API to control it. The process will be auto triggered by a Registrar.

AP needs to give its PIN code to the Registrar when it acts as an Enrollee. AP can get the PIN code by calling the API described in section 2.7.2.3, "WPS get the PIN code".

3) STA as an Enrollee

When the device is STA, assign the parameter `WIFI_PORT_STA` to the "port" parameter of this API, to set the STA as an Enrollee. The AP will act as Registrar.

Once the STA calls this API, the Enrollee process will be triggered. Then the Registrar, such as an AP, needs to input the STA's PIN code and trigger the Registrar process. During 120 seconds, the Enrollee will connect with the AP and get an IP address, or prompt fail.

If the STA's "`pin`" parameter is NULL, a pin code will be auto-generated and displayed through UART, such as "`WPS PIN: 00309448`".

An example implementation is shown below.

```
int32_t wps_sta_pin_example()
{
    uint8_t *bssid = "00:11:22:33:44:55";
    uint8_t *pin = NULL;

    return wifi_wps_connection_by_pin( WIFI_PORT_STA, bssid, pin);
}
```

Another approach to get the PIN code is to call the API described in section 2.7.2.3, "WPS get the PIN code".

An example implementation is shown below.

```
int32_t  wps_sta_pin_example()
{
    uint8_t *bssid = "00:11:22:33:44:55";
    char pin_buf[9] = {0};//the size of the pin code is 8

    wifi_wps_config_get_pin_code(WIFI_PORT_STA, ( uint8_t *)pin_buf);
    printf("Get PIN:%s\n", (char *)pin_buf ); //ex.Get PIN: 00309448
    return wifi_wps_connection_by_pin( WIFI_PORT_STA, bssid, ( uint8_t
*)pin_buf);
}
```

2.7.2.3. WPS get the PIN code

This API is used to get the PIN code of AP or STA. Call the `wifi_wps_connection_by_pin()` API directly.

1) AP get the PIN code

When the device is an AP, assign the parameter `WIFI_PORT_AP` to the "`port`" parameter of this API.

The AP will auto-generate a random PIN code when device powers on, the PIN stays the same for each API call.

An example implementation is shown below.

```
int32_t wps_ap_get_pin_example()
{
    char pin_buf[9] = {0};//the size of the pin code is 8

    wifi_wps_config_get_pin_code(WIFI_PORT_AP, ( uint8_t *)pin_buf);
    printf("Get PIN:%s\n", (char *)pin_buf);

}
```

2) STA gets the PIN code

When the device is in STA, assign the parameter `WIFI_PORT_STA` to the "`port`" parameter of this API.

The STA's PIN code is randomly generated. An example implementation is shown below.

```
int32_t wps_sta_get_pin_example()
{
    char pin_buf[9] = {0};//the size of the pin code is 8

    wifi_wps_config_get_pin_code(WIFI_PORT_STA, ( uint8_t *)pin_buf);
    printf("Get PIN:%s\n", (char *)pin_buf);

}
```