# Wificonf Application Programming Interface

This document intends to provide a comprehensive guide to the implemented user interfaces for Wi-Fi station and AP mode configuration base on the functionalities provided by Realtek Wi-Fi driver.

# Table of Contents

# 1 Introduction

This document intends to provide a comprehensive guide to the implemented user interfaces for Wi-Fi station and AP mode configuration base on the functionalities provided by Realtek Wi-Fi driver.

# 2 Summary

| Usage | API & Keyword |
|---|---|
| How does station connect to AP? | 1. Use ATCMD<br>ATW0=<ssid><br>ATW1=<password><br>ATW2=<key_id><br>ATWC<br>2. Call API in wifi_conf.c<br>wifi_connect : use SSID to connect to AP<br>wifi_connect_bssid : use bssid to connect to AP |
| How does station disconnect from AP? | 1. Use ATCMD<br>ATWD<br>2. Call API in wifi_conf.c<br>wifi_disconnect |
| How to register wifi event callback function? | Search “wifi_reg_event_handler” as reference in wifi_conf.c |
| How to detect wlan condition of connect or disconnect event? | Register wifi event callback function for specific event. (Total event can reference 3.24) |

| | |
|---|---|
| | WIFI_EVENT_CONNECT : association done<br>WIFI_EVENT_FOURWAY_HANDSHAKE_DONE : fourway handshake done<br>WIFI_EVENT_BEACON_AFTER_DHCP : Get IP from DHCP<br>WIFI_EVENT_DISCONNECT : wifi disconnect |
| How to enable/disable power saving mode in station mode? | Call API in wifi_conf.c<br>wifi_enable_powersave<br>wifi_disable_powersave |
| How to start soft AP mode? | 1. Use ATCMD<br>ATW3=<ssid><br>ATW4=<>password<br>ATW5=<channel><br>ATWA<br>2. Call API in wifi_conf.c<br>wifi_start_ap |
| How to start soft AP mode with hidden ssid? | Call API in wifi_conf.c<br>wifi_start_ap_with_hidden_ssid |
| How to create concurrent mode? | Use ATCMD, start AP first then Station<br>ATW3=<ssid><br>ATW4=<>password<br>ATW5=<channel><br>ATWB<br>ATW0=<ssid> |

| | |
|---|---|
| | ATW1=<password><br>ATW2=<key_id><br>ATWC |
| How to set client number in AP mode? | Call API in wifi_util.c<br>wext_set_sta_num |
| How to delete station in AP mode? | Call API in wifi_util.c<br>wext_del_station |
| How to get auto-scan channel ? | Call API in wifi_util.c<br>wext_get_auto_chl |
| How to set partial scan channel in station mode? | Call API in wifi_conf.c<br>wifi_set_pscan_chan |
| How to set auto-reconnect in station mode? | Call API in wifi_conf.c<br>wifi_config_autoreconnect |
| How to get TX power? | Call API in wifi_util.c<br>wext_get_tx_power |
| How to get RX RSSI? | Call API in wifi_conf.c<br>wifi_get_rssi |

# 3 API Specific Data Types

## 3.1 rtw_result_t

```
typedef enum
{
        RTW_SUCCESS                             = 0,
        RTW_PENDING                             = 1
        RTW_TIMEOUT                             = 2
        RTW_PARTIAL_RESULTS                     = 3
        RTW_INVALID_KEY                         = 4
        RTW_DOES_NOT_EXIST                      = 5
        RTW_NOT_AUTHENTICATED                   = 6
        RTW_NOT_KEYED                           = 7
        RTW_IOCTL_FAIL                          = 8
        RTW_BUFFER_UNAVAILABLE_TEMPORARY        = 9
        RTW_BUFFER_UNAVAILABLE_PERMANENT        = 10
        RTW_WPS_PBC_OVERLAP                     = 11
        RTW_CONNECTION_LOST                     = 12
        RTW_ERROR                               = -1
        RTW_BADARG                              = -2
        RTW_BADOPTION                           = -3
        RTW_NOTUP                               = -4
        RTW_NOTDOWN                             = -5
        RTW_NOTAP                               = -6
        RTW_NOTSTA                              = -7
        RTW_BADKEYIDX                           = -8
        RTW_RADIOOFF                            = -9
        RTW_NOTBANDLOCKED                       = -10
        RTW_NOCLK                               = -11
        RTW_BADRATESET                          = -12
        RTW_BADBAND                             = -13
        RTW_BUFTOOSHORT                         = -14
        RTW_BUFTOOLONG                          = -15
        RTW_BUSY                                = -16
        RTW_NOTASSOCIATED                       = -17
        RTW_BADSSIDLEN                          = -18
        RTW_OUTOFRANGECHAN                      = -19
```

```
        RTW_BADCHAN                     = -20
        RTW_BADADDR                     = -21
        RTW_NORESOURCE                  = -22
        RTW_UNSUPPORTED                 = -23
        RTW_BADLEN                      = -24
        RTW_NOTREADY                    = -25
        RTW_EPERM                       = -26
        RTW_NOMEM                       = -27
        RTW_ASSOCIATED                  = -28
        RTW_RANGE                       = -29
        RTW_NOTFOUND                    = -30
        RTW_WME_NOT_ENABLED             = -31
        RTW_TSPEC_NOTFOUND              = -32
        RTW_ACM_NOTSUPPORTED            = -33
        RTW_NOT_WME_ASSOCIATION         = -34
        RTW_SDIO_ERROR                  = -35
        RTW_WLAN_DOWN                   = -36
        RTW_BAD_VERSION                 = -37
        RTW_TXFAIL                      = -38
        RTW_RXFAIL                      = -39
        RTW_NODEVICE                    = -40
        RTW_UNFINISHED                  = -41
        RTW_NONRESIDENT                 = -42
        RTW_DISABLED                    = -43
} rtw_result_t;
```

The enumeration lists the return result of the function.

## 3.2 rtw_802_11_band_t

```
typedef enum
{
        RTW_802_11_BAND_5GHZ        = 0,
        RTW_802_11_BAND_2_4GHZ      = 1
} rtw_802_11_band_t;
```

The enumeration lists the band type.

## 3.3 rtw_scan_type_t

```
typedef enum
{
        RTW_SCAN_TYPE_ACTIVE                    = 0x00,
        RTW_SCAN_TYPE_PASSIVE                   = 0x01,
        RTW_SCAN_TYPE_PROHIBITED_CHANNELS       = 0x04
} rtw_scan_type_t;
```

The enumeration lists the scan type. RTW_SCAN_TYPE_ACTIVE means actively scan a network by sending 802.11 probe(s). RTW_SCAN_TYPE_PASSIVE means passively scan a network by listening for beacons from APs. RTW_SCAN_TYPE_PROHIBITED_CHANNELS means passively scan on channels not enabled by the country code.

## 3.4 rtw_bss_type_t

```
typedef enum
{
        RTW_BSS_TYPE_INFRASTRUCTURE     = 0,
        RTW_BSS_TYPE_ADHOC              = 1,
        RTW_BSS_TYPE_ANY                = 2,
        RTW_BSS_TYPE_UNKNOWN            = -1
}rtw_bss_type_t;
```

The enumeration lists the bss types. RTW_BSS_TYPE_INFRASTRUCTURE denotes infrastructure network. RTW_BSS_TYPE_ADHOC denotes an 802.11 ad-hoc IBSS network. RTW_BSS_TYPE_ANY denotes either infrastructure or ad-hoc network. RTW_BSS_TYPE_UNKNOWN may be returned by scan function if BSS type is unknown.

## 3.5 rtw_wps_type_t

```
typedef enum
{
```

```
        RTW_WPS_TYPE_DEFAULT                = 0x0000,
        RTW_WPS_TYPE_USER_SPECIFIED         = 0x0001,
        RTW_WPS_TYPE_MACHINE_SPECIFIED      = 0x0002,
        RTW_WPS_TYPE_REKEY                  = 0x0003,
        RTW_WPS_TYPE_PUSHBUTTON             = 0x0004,
        RTW_WPS_TYPE_REGISTRAR_SPECIFIED    = 0x0005,
        RTW_WPS_TYPE_NONE                   = 0x0006
}rtw_wps_type_t;
```

The enumeration lists the wps type.

## 3.6 rtw_mode_t

```
typedef enum
{
        RTW_MODE_NONE               = 0,
        RTW_MODE_STA,
        RTW_MODE_AP,
        RTW_MODE_STA_AP,
        RTW_MODE_PROMISC,
        RTW_MODE_P2P
}rtw_mode_t;
```

The enumeration lists the supported operation mode by WIFI driver, including station and AP mode.

## 3.7 rtw_security_t

```
typedef enum
{
        RTW_SECURITY_OPEN               = 0,
        RTW_SECURITY_WEP_PSK            = WEP_ENABLED,
        RTW_SECURITY_WEP_SHARED         = ( WEP_ENABLED | SHARED_ENABLED ),
        RTW_SECURITY_WPA_TKIP_PSK       = ( WPA_SECURITY  | TKIP_ENABLED ),
        RTW_SECURITY_WPA_AES_PSK        = ( WPA_SECURITY  | AES_ENABLED ),
```

```
        RTW_SECURITY_WPA2_AES_PSK       = ( WPA2_SECURITY | AES_ENABLED ),
        RTW_SECURITY_WPA2_TKIP_PSK      = ( WPA2_SECURITY | TKIP_ENABLED ),
        RTW_SECURITY_WPA2_MIXED_PSK     = ( WPA2_SECURITY | AES_ENABLED |
                                        TKIP_ENABLED ),
        RTW_SECURITY_WPA_WPA2_MIXED     = ( WPA_SECURITY  | WPA2_SECURITY ),
        RTW_SECURITY_WPS_OPEN           = WPS_ENABLED,
        RTW_SECURITY_WPS_SECURE         = (WPS_ENABLED | AES_ENABLED),
        RTW_SECURITY_UNKNOWN            = -1,
        RTW_SECURITY_FORCE_32_BIT       = 0x7fffffff
}rtw_security_t;
```

The enumeration lists the possible security type to set when connection. Station mode supports OPEN, WEP and WPA2. AP mode support OPEN and WPA2.

## 3.8 rtw_link_status_t

```
typedef enum
{
        RTW_LINK_DISCONNECTED       = 0,
        RTW_LINK_CONNECTED
}rtw_link_status_t;
```

The enumeration lists the status to describe the connection link.

## 3.9 rtw_country_code_t

```
typedef enum
{
        RTW_COUNTRY_WORLD1,
        RTW_COUNTRY_ETSI1,
        RTW_COUNTRY_FCC1,
        RTW_COUNTRY_MKK1,
        RTW_COUNTRY_ETSI2,
        RTW_COUNTRY_FCC2,
        RTW_COUNTRY_WORLD2,
        RTW_COUNTRY_MKK2,
```

```
	/* SPECIAL */
	RTW_COUNTRY_WORLD,
	RTW_COUNTRY_EU,

	/* JAPANESE */
	RTW_COUNTRY_JP,

	/* FCC , 19 countries*/
	RTW_COUNTRY_AS,
	RTW_COUNTRY_BM,
	RTW_COUNTRY_CA,
	RTW_COUNTRY_DM,
	RTW_COUNTRY_DO,
	RTW_COUNTRY_FM,
	RTW_COUNTRY_GD,
	RTW_COUNTRY_GT,
	RTW_COUNTRY_GU,
	RTW_COUNTRY_HT,
	RTW_COUNTRY_MH,
	RTW_COUNTRY_MP,
	RTW_COUNTRY_NI,
	RTW_COUNTRY_PA,
	RTW_COUNTRY_PR,
	RTW_COUNTRY_PW,
	RTW_COUNTRY_TW,
	RTW_COUNTRY_US,
	RTW_COUNTRY_VI,

	/* others,  ETSI */
	RTW_COUNTRY_AD,
	RTW_COUNTRY_AE,
	RTW_COUNTRY_AF,
	RTW_COUNTRY_AI,
	RTW_COUNTRY_AL,
	RTW_COUNTRY_AM,
	RTW_COUNTRY_AN,
	RTW_COUNTRY_AR,
	RTW_COUNTRY_AT,
	RTW_COUNTRY_AU,
	RTW_COUNTRY_AW,
	RTW_COUNTRY_AZ,
	RTW_COUNTRY_BA,
```

```
    RTW_COUNTRY_BB,
    RTW_COUNTRY_BD,
    RTW_COUNTRY_BE,
    RTW_COUNTRY_BF,
    RTW_COUNTRY_BG,
    RTW_COUNTRY_BH,
    RTW_COUNTRY_BL,
    RTW_COUNTRY_BN,
    RTW_COUNTRY_BO,
    RTW_COUNTRY_BR,
    RTW_COUNTRY_BS,
    RTW_COUNTRY_BT,
    RTW_COUNTRY_BY,
    RTW_COUNTRY_BZ,
    RTW_COUNTRY_CF,
    RTW_COUNTRY_CH,
    RTW_COUNTRY_CI,
    RTW_COUNTRY_CL,
    RTW_COUNTRY_CN,
    RTW_COUNTRY_CO,
    RTW_COUNTRY_CR,
    RTW_COUNTRY_CX,
    RTW_COUNTRY_CY,
    RTW_COUNTRY_CZ,
    RTW_COUNTRY_DE,
    RTW_COUNTRY_DK,
    RTW_COUNTRY_DZ,
    RTW_COUNTRY_EC,
    RTW_COUNTRY_EE,
    RTW_COUNTRY_EG,
    RTW_COUNTRY_ES,
    RTW_COUNTRY_ET,
    RTW_COUNTRY_FI,
    RTW_COUNTRY_FR,
    RTW_COUNTRY_GB,
    RTW_COUNTRY_GE,
    RTW_COUNTRY_GF,
    RTW_COUNTRY_GH,
    RTW_COUNTRY_GL,
    RTW_COUNTRY_GP,
    RTW_COUNTRY_GR,
    RTW_COUNTRY_GY,
```

```
RTW_COUNTRY_HK,
RTW_COUNTRY_HN,
RTW_COUNTRY_HR,
RTW_COUNTRY_HU,
RTW_COUNTRY_ID,
RTW_COUNTRY_IE,
RTW_COUNTRY_IL,
RTW_COUNTRY_IN,
RTW_COUNTRY_IQ,
RTW_COUNTRY_IR,
RTW_COUNTRY_IS,
RTW_COUNTRY_IT,
RTW_COUNTRY_JM,
RTW_COUNTRY_JO,
RTW_COUNTRY_KE,
RTW_COUNTRY_KH,
RTW_COUNTRY_KN,
RTW_COUNTRY_KP,
RTW_COUNTRY_KR,
RTW_COUNTRY_KW,
RTW_COUNTRY_KY,
RTW_COUNTRY_KZ,
RTW_COUNTRY_LA,
RTW_COUNTRY_LB,
RTW_COUNTRY_LC,
RTW_COUNTRY_LI,
RTW_COUNTRY_LK,
RTW_COUNTRY_LR,
RTW_COUNTRY_LS,
RTW_COUNTRY_LT,
RTW_COUNTRY_LU,
RTW_COUNTRY_LV,
RTW_COUNTRY_MA,
RTW_COUNTRY_MC,
RTW_COUNTRY_MD,
RTW_COUNTRY_ME,
RTW_COUNTRY_MF,
RTW_COUNTRY_MK,
RTW_COUNTRY_MN,
RTW_COUNTRY_MO,
RTW_COUNTRY_MQ,
RTW_COUNTRY_MR,
```

```
    RTW_COUNTRY_MT,
    RTW_COUNTRY_MU,
    RTW_COUNTRY_MV,
    RTW_COUNTRY_MW,
    RTW_COUNTRY_MX,
    RTW_COUNTRY_MY,
    RTW_COUNTRY_NG,
    RTW_COUNTRY_NL,
    RTW_COUNTRY_NO,
    RTW_COUNTRY_NP,
    RTW_COUNTRY_NZ,
    RTW_COUNTRY_OM,
    RTW_COUNTRY_PE,
    RTW_COUNTRY_PF,
    RTW_COUNTRY_PG,
    RTW_COUNTRY_PH,
    RTW_COUNTRY_PK,
    RTW_COUNTRY_PL,
    RTW_COUNTRY_PM,
    RTW_COUNTRY_PT,
    RTW_COUNTRY_PY,
    RTW_COUNTRY_QA,
    RTW_COUNTRY_RS,
    RTW_COUNTRY_RU,
    RTW_COUNTRY_RW,
    RTW_COUNTRY_SA,
    RTW_COUNTRY_SE,
    RTW_COUNTRY_SG,
    RTW_COUNTRY_SI,
    RTW_COUNTRY_SK,
    RTW_COUNTRY_SN,
    RTW_COUNTRY_SR,
    RTW_COUNTRY_SV,
    RTW_COUNTRY_SY,
    RTW_COUNTRY_TC,
    RTW_COUNTRY_TD,
    RTW_COUNTRY_TG,
    RTW_COUNTRY_TH,
    RTW_COUNTRY_TN,
    RTW_COUNTRY_TR,
    RTW_COUNTRY_TT,
    RTW_COUNTRY_TZ,
```

```
        RTW_COUNTRY_UA,
        RTW_COUNTRY_UG,
        RTW_COUNTRY_UY,
        RTW_COUNTRY_UZ,
        RTW_COUNTRY_VC,
        RTW_COUNTRY_VE,
        RTW_COUNTRY_VN,
        RTW_COUNTRY_VU,
        RTW_COUNTRY_WF,
        RTW_COUNTRY_WS,
        RTW_COUNTRY_YE,
        RTW_COUNTRY_YT,
        RTW_COUNTRY_ZA,
        RTW_COUNTRY_ZW,

        RTW_COUNTRY_MAX
}rtw_country_code_t;
```

The enumeration lists all the country codes able to set to WIFI driver.

## 3.10 rtw_network_mode_t

```
typedef enum
{
        RTW_NETWORK_B                   = 1,
        RTW_NETWORK_BG                  = 3,
        RTW_NETWORK_BGN                 = 11
}rtw_network_mode_t;
```

The enumeration lists all the network  bgn mode .

## 3.11 rtw_interface_t

```
typedef enum
{
        RTW_STA_INTERFACE               = 0,
        RTW_AP_INTERFACE                = 1,
```

```
}rtw_interface_t;
```

The enumeration lists the interface. RTW_STA_INTERFACE means STA or client interface, RTW_AP_INTERFACE means softAP interface .

## 3.12 rtw_packet_filter_rule_t

```
typedef enum
{
        RTW_POSITIVE_MATCHING       = 0,
        RTW_NEGATIVE_MATCHING       = 1,
} rtw_packet_filter_rule_t;
```

The enumeration lists the packet filter rule. RTW_POSITIVE_MATCHING means receiving the data matching with this pattern and discard the other data. RTW_NEGATIVE_MATCHING means discard the data matching with this pattern and receive the other data.

## 3.13 rtw_rcr_level_t

```
typedef enum
{
        RTW_PROMISC_DISABLE         = 0,
        RTW_PROMISC_ENABLE          = 1,
        RTW_PROMISC_ENABLE_1        = 2,
        RTW_PROMISC_ENABLE_2        = 3,
        RTW_PROMISC_ENABLE_3        = 4
} rtw_rcr_level_t;
```

The enumeration lists the promisc level. RTW_PROMISC_DISABLE means disable the promisc, RTW_PROMISC_ENABLE means enable the promisc and fetch all ethernet packets, RTW_PROMISC_ENABLE_1 is used to fetch only B/M packets, RTW_PROMISC_ENABLE_2 is used to fetch all 802.11 packets, RTW_PROMISC_ENABLE_3 is used to fetch only B/M 802.11 packets.

## 3.14 rtw_ssid_t

```
typedef struct rtw_ssid
{
        unsigned char                   len;
        unsigned char                   val[33];
}rtw_ssid_t;
```

This struct is used to describe the SSID.

## 3.15 rtw_mac_t

```
typedef struct rtw_mac
{
        unsigned char                   octet[6];
}rtw_mac_t;
```

This struct is used to describe the unique 6-byte MAC address.

## 3.16 rtw_scan_result_t

```
typedef struct rtw_scan_result
{
        rtw_ssid_t                      SSID;
        rtw_mac_t                       BSSID;
        signed short                    signal_strength;
        rtw_bss_type_t                  bss_type;
        rtw_security_t                  security;
        rtw_wps_type_t                  wps_type;
        unsigned int                    channel;
        rtw_802_11_band_t               band;
}rtw_scan_result_t;
```

This struct is used to describe the scan result of the AP.

## 3.17 rtw_scan_handler_result_t

```
typedef struct rtw_scan_handler_result
{
        rtw_scan_result_t               ap_details;
        rtw_bool_t                      scan_complete;
        void*                           user_data;
}rtw_scan_handler_result_t;
```

This structure is used to describe the data needed by scan result handler function.

## 3.18 rtw_network_info_t

```
typedef struct rtw_network_info
{
        rtw_ssid_t                      ssid;
        rtw_mac_t                       bssid;
        rtw_security_t                  security_type;
        unsigned char *                 password;
        int                             password_len;
        int                             key_id;
}rtw_network_info_t;
```

This structure is used to describe the station mode setting about SSID, security type and password, used when connecting to an AP. The data length of string pointed by ssid should not exceed 32, and the data length of string pointed by password should not exceed 64.

## 3.19 rtw_ap_info_t

```
typedef struct rtw_ap_info
{
        rtw_ssid_t                      ssid;
        rtw_security_t                  security_type;
```

```
        unsigned char *                 password;
        int                             password_len;
        int                             channel;
} rtw_ap_info_t;
```

This structure is used to describe the setting about SSID, security type, password and default channel, used to start AP mode. The data length of string pointed by ssid should not exceed 32, and the data length of string pointed by password should not exceed 64.

## 3.20 rtw_wifi_setting_t

```
typedef struct rtw_wifi_setting
{
        rtw_mode_t                      mode;
        unsigned char                   ssid[33];
        unsigned char                   channel;
        rtw_security_t                  security_type;
        unsigned char                   password[65];
        unsigned char                   key_idx;
} rtw_wifi_setting_t;
```

This structure is used to store the WIFI setting gotten from WIFI driver.

## 3.21 rtw_wifi_config_t

```
typedef struct rtw_wifi_config {
        unsigned int                    boot_mode;
        unsigned char                   ssid[32]
        unsigned char                   ssid_len;
        unsigned char                   security_type;
        unsigned char                   password[65];
        unsigned char                   password_len;
        unsigned char                   channel;
} rtw_wifi_config_t;
```

The struct is used to describe the setting when config the network.

## 3.22 rtw_maclist_t

```
typedef struct {
        unsigned int                    count;
        rtw_mac_t                       mac_list[1];
} rtw_maclist_t;
```

The struct is used to describe the maclist. Count means number of MAC addresses in the list. Mac_list means variable length array of MAC address.

## 3.23 rtw_bss_info_t

```
typedef struct {
        unsigned int                    version;
        unsigned int                    length;
        rtw_mac_t                       BSSID;
        unsigned short                  beacon_period;
        unsigned short                  capability;
        unsigned char                   SSID_len;
        unsigned char                   SSID[32];
        unsigned char                   channel;
        unsigned short                  atim_window;
        unsigned char                   dtim_period;
        signed short                    RSSI;
        unsigned char                   n_cap;
        unsigned int                    nbss_cap;
        unsigned char                   basic_mcs[MCSSET_LEN];
        unsigned short                  ie_offset;
        unsigned int                    ie_length;
} rtw_bss_info_t;
```

The struct is used to describe the bss info of the network. It includes the version, BSSID, beacon_period, capability, SSID, channel, atim_window, dtim_period, RSSI e.g.

## 3.24 rtw_event_indicate_t

```
typedef enum _WIFI_EVENT_INDICATE {
    WIFI_EVENT_CONNECT = 0,
    WIFI_EVENT_DISCONNECT = 1,
    WIFI_EVENT_FOURWAY_HANDSHAKE_DONE = 2,
    WIFI_EVENT_SCAN_RESULT_REPORT = 3,
    WIFI_EVENT_SCAN_DONE = 4,
    WIFI_EVENT_RECONNECTION_FAIL = 5,
    WIFI_EVENT_SEND_ACTION_DONE = 6,
    WIFI_EVENT_RX_MGNT = 7,
    WIFI_EVENT_STA_ASSOC = 8,
    WIFI_EVENT_STA_DISASSOC = 9,
    WIFI_EVENT_STA_WPS_START = 10,
    WIFI_EVENT_WPS_FINISH = 11,
    WIFI_EVENT_EAPOL_START = 12,
    WIFI_EVENT_EAPOL_RECVD = 13,
    WIFI_EVENT_NO_NETWORK = 14,
    WIFI_EVENT_BEACON_AFTER_DHCP = 15,
    WIFI_EVENT_MAX,
} rtw_event_indicate_t;
```

This enumeration is event type indicated from wlan driver.

## 3.25 rtw_custom_ie_type_t

```
typedef enum CUSTOM_IE_TYPE {
        PROBE_REQ                                       = BIT(0),
        PROBE_RSP                                       = BIT(1),
        BEACON                                          = BIT(2)
} rtw_custom_ie_type_t;
```

This enumeration is transmission type for wifi custom ie.

## 3.26 rtw_custom_ie_t

```
typedef struct _cus_ie {
```

```
        __u8 *ie;
        __u8 type;
} rtw_custom_ie_t, *p_rtw_custom_ie_t;
```

This structure is used to set WIFI custom ie list, and type match rtw_custom_ie_type_t. The ie will be transmitted according to the type.

ie format:

| element ID | Length of Content | content in length byte |
|---|---|---|

## 3.27 rtw_packet_filter_pattern_t

```
typedef struct {
        unsigned short  offset;
        unsigned short  mask_size;
        unsigned char*  mask;
        unsigned char*  pattern;
} rtw_packet_filter_pattern_t;
```

This structure is used to set WIFI packet filter pattern. Offset in bytes is used to specify where to start filtering. Mask_size is the size of the mask in bytes. Mask means filter mask. Pattern is the bytes used to filter.

## 3.28 rtw_adaptivity_mode_t

```
typedef enum {
        RTW_ADAPTIVITY_DISABLE = 0,
        RTW_ADAPTIVITY_NORMAL,                  // CE
        RTW_ADAPTIVITY_CARRIER_SENSE            // MKK
} rtw_adaptivity_mode_t;
```

This enumeration is adaptivity type. RTW_ADAPTIVITY_NORMAL is for CE and RTW_ADAPTIVITY_CARRIER_SENSE is for MKK.

# 4 Application Programming Interface

## 4.1 Wifi enable/disable

### 4.1.1 wifi_on

This function uses to enable wifi .

Syntax

```
Int
wifi_on(
        rtw_mode_t mode
        )
```

***Parameters***

*mode*

Decide to enable WiFi in which mode. Such as STA mode, AP mode, STA+AP concurrent mode or Promiscuous mode.

***Return Value***

If the function succeeds, the return value is 0

### 4.1.2 wifi_off

```
Int
wifi_off(
        void
        )
```

***Parameters***

*None*

***Return Value***

If the function succeeds, the return value is 0

### 4.1.3 wifi_is_up

This function checked if the interface specified is up.

```
int
wifi_is_up(
        rtw_interface_t interface
        )
```

***Parameters***

*interface*

The interface can be set RTW_AP_INTERFACE or RTW_STA_INTERFACE.

***Return Value***

If the interface is up, the return value is 1

### 4.1.4 wifi_is_ready_to_transceive

This function checked if the interface specified is ready to transceiver Ethernet packets.

```
int
wifi_is_ready_to_transceive (
        rtw_interface_t interface
        )
```

***Parameters***

*interface*

The interface can be set RTW_AP_INTERFACE or RTW_STA_INTERFACE.

***Return Value***

If the function succeeds, the return value is 0

## 4.2 Station Mode Connection

### 4.2.1 wifi_connect

This function triggers connection to a WIFI network.

Syntax

```
int
wifi_connect(
        char *ssid,
        rtw_security_t security_type,
```

```
    char *password,
    int ssid_len,
    int password_len,
    int key_id,
    void *semaphore
    )
```

### *Parameters*

*ssid*

A null terminated string containing the SSID name of the network to join.

*Security_type*

The security type of the AP to connect.

*password*

A byte array containing the security_key.

*ssid_len*

The length of the SSID in bytes.

*password_len*

The length of the security_key in bytes.

*key_id*

The index of the wep key.

*semaphore*

A user provided semaphore that is flagged when the join is complete.

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None.

## 4.2.2 wifi_disconnect

This function triggers disconnection from current WIFI network.

### *Syntax*

```
int
wifi_disconnect (
        void
        )
```

### *Parameters*

None

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None

## 4.3 AP Mode Startup

### 4.3.1 wifi_start_ap

This function triggers WIFI driver to start the AP mode.

### *Syntax*

```
int
wifi_start_ap (
        char *ssid,
        rtw_security_t security_type,
        char *password,
        int ssid_len,
        int password_len,
        int channel
        )
```

### *Parameters*

*ssid*

A null terminated string containing the SSID name of the AP.

*security_type*

The security type of the AP to start.

*password*

A byte array containing the security key for the AP.

*ssid_len*

The length of the SSID in bytes.

*password_len*

The length of the security_key in bytes.

*channel*

802.11 channel number.

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None

## 4.3.2 wifi_restart_ap

This function triggers WIFI driver to restart an infrastructure WiFi network.

### *Syntax*

```
int
wifi_restart_ap (
        unsigned char *ssid,
        rtw_security_t security_type,
        unsigned char *password,
        int ssid_len,
        int password_len,
        int channel
        )
```

### *Parameters*

*ssid*

A null terminated string containing the SSID name of the network to join.

*security_type*

Authentication type.

*password*

A byte array containing the security key for the network.

*ssid_len*

The length of the SSID in bytes.

*password_len*

The length of the security_key in bytes.

*channel*

802.11 channel number.

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None

## 4.3.3 wifi_get_ap_info

This function gets the SoftAP information.

### *Syntax*

```
int
wifi_get_ap_info (
        rtw_bss_info_t *ap_info,
        rtw_security_t *security
        )
```

### *Parameters*

*ap_info*

The location where the AP info will be stored.

*security*

The security type.

### *Return Value*

If the result was successfully get return RTW_SUCCESS, else return RTW_ERROR.

### *Remarks*

None

### 4.3.4 wifi_get_associated_client_list

This function gets the associated clients with SoftAP.

#### *Syntax*

```
int
wifi_get_associated_client_list (
        void * client_list_buffer,
        unsigned short buffer_length
        )
```

#### *Parameters*

*client_list_buffer*

The location where the client list will be stored.

*buffer_length*

The buffer length.

#### *Return Value*

If the result was successfully get return RTW_SUCCESS, else return RTW_ERROR.

#### *Remarks*

None

## 4.4 Wifi Setting Information

### 4.4.1 wifi_get_setting

This function gets current WIFI setting from driver.

#### *Syntax*

```
int
wifi_get_setting (
        const char *ifname,
        rtw_wifi_setting_t *pSetting
        )
```

#### *Parameters*

*Ifname*

The wlan name， can use WLAN0_NAME or WLAN1_NAME.

*pSetting*

Points to the rtw_wifi_setting_t structure to store the WIFI setting gotten from driver.

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None

## 4.4.2 wifi_show_setting

This function simply shows the information stored in a rtw_wifi_setting_t structure.

### *Syntax*

```
Int
wifi_show_setting (
        const char *ifname,
        rtw_wifi_setting_t *pSetting
        )
```

### *Parameters*

*Ifname*

The wlan name， can use WLAN0_NAME or WLAN1_NAME.

*pSetting*

Points to the rtw_wifi_setting_t structure which information is gotten by wifi_get_setting().

### *Return Value*

If the function succeeds, the return value is RTW_SUCCESS.

### *Remarks*

None.

# 4.5 Wifi RF Control

## 4.5.1 wifi_rf_on

This function enables the WIFI RF.

*Syntax*

```
int
wifi_rf_on (
        void
        )
```

*Parameters*

None.

*Return Value*

If the function succeeds, the return value is 0.

*Remarks*

None.

### 4.5.2 wifi_rf_off

This function disables WIFI RF.

*Syntax*

```
int
wifi_rf_off (
        void
        )
```

*Parameters*

None.

*Return Value*

If the function succeeds, the return value is 0.

*Remarks*

None

## 4.6 Wifi RSSI Information

### 4.6.1 wifi_get_rssi

This function gets RSSI value from driver.

*Syntax*

```
int
wifi_get_rssi (
```

```
        int *pRSSI
        )
```

***Parameters***

*pRSSI*

Points to the integer to store the RSSI value gotten from driver.

***Return Value***

If the function succeeds, the return value is 0.

***Remarks***

None.

# 4.7 Country Code Setup

## 4.7.1 wifi_set_country

This function sets country code to driver.

***Syntax***

```
int
wifi_set_country (
        rtw_country_code_t country_code
        )
```

***Parameters***

*country_code*

Specifies the country code.

***Return Value***

If the function succeeds, the return value is 0.

***Remarks***

None.

# 4.8 Network Mode Setup

## 4.8.1 wifi_set_network_mode

Driver works in BGN mode in default after driver initialization. This function is used to change wireless network mode for station mode before connecting to AP

### *Syntax*

```
int
wifi_set_network_mode(
        rtw_network_mode_t mode
        );
```

### *Parameters*

*mode*

Network mode to set network to B, BG or BGN.

### *Return Value*

If the function succeeds, the return value is 0.

# 4.9 Wifi Scan List

## 4.9.1 wifi_scan_networks

This function is used to scan AP list.

### *Syntax*

```
int
wifi_scan_networks(
        rtw_scan_result_handler_t results_handler,
        void *user_data
        );
```

### *Parameters*

*results_handler*

The callback function which will receive and process the result data.

*user_data*

user specific data that will be passed directly to the callback function.

### *Return Value*

If the function succeeds, the return value is 0.

### *Remarks*

Callback must not use blocking functions, since it is called from the context of the RTW thread. The callback, user_data variables will be referenced after the function returns. Those variable must remain valid until the scan is complete.

# 4.10 Wlan Driver Indication

## 4.10.1 wifi_indication

Wlan driver indicate event to upper layer through wifi_indication.

### *Syntax*

```
void
wifi_indication (
        rtw_event_indicate_t event,
        char *buf,
        int buf_len,
        int flag
        );
```

### *Parameters*

Event

An Event reported from driver to upper layer application.

0: WIFI_EVENT_CONNECT

For WPA/WPA2 mode, indication of connection does not mean data can be correctly transmitted or received. Data can be correctly transmitted or received only when 4-way handshake is done. Please check WIFI_EVENT_FOURWAY_HANDSHAKE_DONE event.

1: WIFI_EVENT_DISCONNECT

2: WIFI_EVENT_FOURWAY_HANDSHAKE_DONE

3: WIFI_EVENT_SCAN_RESULT_REPORT

4: WIFI_EVENT_SCAN_DONE

5: WIFI_EVENT_RECONNECTION_FAIL

This flag works with CONFIG_AUTO_RECONNECT enabled, and will be called while auto reconnection failed.

6: WIFI_EVENT_SEND_ACTION_DONE

7: WIFI_EVENT_RX_MGNT

8: WIFI_EVENT_STA_ASSOC

9: WIFI_EVENT_STA_DISASSOC

*buf*

If it is not NUL, buf is a Pointer to the buffer for message string.

*buf_len*

It indicates the length of the buffer.

*flag*

It indicates some extra information, sometimes it is zero.

### *Return Value*

None

### *Remarks*

If upper layer application triggers additional operations on receiving of wext_wlan_indicate, please strictly check current stack size usage (by using uxTaskGetStackHighWaterMark() ) , and tries not to share the same stack with wlan driver if remaining stack space is not available for the following operations. ex: using semaphore to notice another thread instead of handing event directly in wifi_indication().

# 4.11 Wifi Partial Channel Scan

## 4.11.1 wifi_set_pscan_chan

This function sets the channel used to be partial scanned.

### *Syntax*

```
void
wifi_set_pscan_chan (
        __u8 *channel_list,
        __u8 * pscan_config,
        __u8 length,
        );
```

### *Parameters*

*channel_list*

An array stores the channel list.

*pscan_config*

The pscan_config of the channel set.

length

Indicate the length of the channel_list.

***Return Value***

Return 0 if success, otherwise return -1.

***Remarks***

This function should be used with wifi_scan function. First, use wifi_set_pscan_chan to indicate which channel will be scanned scan, and then use wifi_scan to get scanned results.

# 4.12 Wifi Packet filter

## 4.12.1 wifi_init_packet_filter

This function is used to init packet filter related data.

***Syntax***

```
void
wifi_init_packet_filter (
        );
```

***Parameters***

None.

***Return Value***

None.

## 4.12.2 wifi_add_packet_filter

This function is used to add packet filter, and now the maximum number of filter is 5 .

***Syntax***

```
int
wifi_add_packet_filter (
        unsigned char filter_id,
```

```
        rtw_packet_filter_pattern_t *patt,
        rtw_packet_filter_rule_t rule
        );
```

### *Parameters*

*filter_id*

The filter id.

patt

Point to the filter pattern.

rule

Point to the filter rule.

### *Return Value*

Return 0 if success, otherwise return -1.

## 4.12.3 wifi_enable_packet_filter

This function is used to enable packet filter. The filter can be used only if it has been enabled.

### *Syntax*

```
int
wifi_enable_packet_filter (
        unsigned char filter_id
        );
```

### *Parameters*

*filter_id*

The filter id.

### *Return Value*

Return 0 if success, otherwise return -1.

## 4.12.4 wifi_disable_packet_filter

This function is used to disable the packet filter.

*Syntax*

```
int
wifi_disable_packet_filter (
        unsigned char filter_id
        );
```

*Parameters*

*filter_id*

The filter id.

*Return Value*

Return 0 if success, otherwise return -1.

### 4.12.5 wifi_remove_packet_filter

This function is used to remove the packet filter.

*Syntax*

```
int
wifi_remove_packet_filter (
        unsigned char filter_id
        );
```

*Parameters*

*filter_id*

The filter id.

*Return Value*

Return 0 if success, otherwise return -1.

## 4.13 Wifi Promiscuous Mode

### 4.13.1 wifi_set_promisc

This function lets Wi-Fi to start or stop Promiscuous mode.

*Syntax*

```
void
wifi_set_promisc (
        rtw_rcr_level_t r enabled,
        void (*callback)(unsigned char*, unsigned int, void*),
```

```
        unsigned char len_used,
);
```

### *Parameters*

*enabled*

Enable or disable promiscuous mode. 0 means disable the promiscuous mode, 1 means enable the promiscuous mode and fetch all ethernet packets, 2 is used to fetch only B/M packets, 3 is used to fetch all 802.11 packets, 4 is used to fetch only B/M 802.11 packets.

*callback*

Callback function used to process packet information captured by Wi-Fi.

*len_used*

If len_used set to 1, beacon frames will be fetched.

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

This function can be used to implement vendor specified simple configure.

## 4.13.2 wifi_enter_promisc_mode

This function lets Wi-Fi enter promisc mode.

### *Syntax*

```
void
wifi_enter_promisc_mode (
        void
        );
```

### *Parameters*

*void*

### *Return Value*

void.

### *Remarks*

NONE.

# 4.14 Wifi Auto Reconnection

## 4.14.1 wifi_set_autoreconnect

This function sets reconnection mode.

### *Syntax*

```
int
wifi_set_autoreconnect (
        __u8 mode,
        );
```

### *Parameters*

*mode*

set 1/0 to enable/disable the reconnection mode.

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

Defining CONFIG_AUTO_RECONNECT as 1 in “autoconf.h” needs to be done before compiling, or this API won’t be effective.

## 4.14.2 wifi_get_autoreconnect

This function gets the result of setting reconnection mode

### *Syntax*

```
int
wifi_get_autoreconnect (
        __u8 *mode
        );
```

### *Parameters*

*mode*

Point to the result of setting reconnection mode.

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

Defining CONFIG_AUTO_RECONNECT as 1 in “autoconf.h” needs to be done before compiling, or this API won’t be effective.

# 4.15 Wifi Custom IE

## 4.15.1 wifi_add_custom_ie

This function setups custom ie list according to rtw_custom_ie_type_t.

### *Syntax*

```
int
wifi_add_custom_ie (
        void *cus_ie,
        int ie_num,
        );
```

### *Parameters*

cus_ie

pointer to WIFI CUSTOM IE list.

ie_num

It indicate the number of WIFI CUSTOM IE list.

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

Defining CONFIG_CUSTOM_IE in “autoconf.h” needs to be done before compiling, or this API won’t be effective. This API can’t be executed twice before deleting the previous custom ie list.

## 4.15.2 wifi_update_custom_ie

This function updates the item in WIFI CUSTOM IE list.

### *Syntax*

```
int
```

```
wifi_update_custom_ie (
        void *cus_ie,
        int ie_index
        );
```

### *Parameters*

cus_ie

pointer to WIFI CUSTOM IE address.

ie_index

index of WIFI CUSTOM IE list.

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

Defining CONFIG_CUSTOM_IE in “autoconf.h” needs to be done before compiling, or this API won’t be effective.

## 4.15.3 wifi_del_custom_ie

This function sets connection mode to reconnection mode.

### *Syntax*

```
int
wifi_del_custom_ie ();
```

### *Parameters*

*None*

### *Return Value*

Return 0 if success, otherwise return -1.

### *Remarks*

Defining CONFIG_CUSTOM_IE in “autoconf.h” needs to be done before compiling, or this API won’t be effective.

# 4.16 Wifi Mac Address

## 4.16.1 wifi_set_mac_address

This function sets mac address of the 802.11 device.

### *Syntax*

```
int
wifi_set_mac_address (
        char *mac
        );
```

### *Parameters*

*mac*

Pointer to a variable that the current MAC address will be written to. The mac format is “00E04C871100” or “00e04c871100” for example.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

## 4.16.2 wifi_get_mac_address

This function gets the mac address of the 802.11 device.

### *Syntax*

```
int
wifi_get_mac_address(
        char *mac
        );
```

### *Parameters*

*mac*

Point to the result of the mac address will be get. The mac format is “xx:xx:xx:xx:xx:xx”.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

# 4.17 Wifi Power save

## 4.17.1 wifi_enable_powersave

This function enable wifi powersave mode.

### *Syntax*

```
int
wifi_enable_powersave (
        void
        );
```

### *Parameters*

*void*

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

## 4.17.2 wifi_disable_powersave

This function disable wifi powersave mode.

### *Syntax*

```
int
wifi_disable_powersave (
        void
        );
```

### *Parameters*

*void*

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

# 4.18 Wifi Tx Power

## 4.18.1 wifi_set_txpower

This function set the tx power in index units.

### *Syntax*

```
int
wifi_set_txpower (
        int poweridx
        );
```

### *Parameters*

*poweridx*

The desired tx power in index.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

## 4.18.2 wifi_get_txpower

This function gets the tx power in index units.

### *Syntax*

```
int
wifi_get_txpower (
        int *poweridx
        );
```

### *Parameters*

*poweridx*

The variable to receive the tx power in index.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

# 4.19 Wifi Channel

## 4.19.1 wifi_set_channel

This function set the current channel on STA interface.

### *Syntax*

```
int
wifi_set_channel (
        int channel
        );
```

### *Parameters*

*channel*

Set the current channel on STA interface.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

## 4.19.2 wifi_get_channel

This function gets the current channel on STA interface.

### *Syntax*

```
int
wifi_get_channel (
        int *channel
        );
```

### *Parameters*

*channel*

A pointer to the variable where the channel value will be written..

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

# 4.20 Wifi Multicast Address

## 4.20.1 wifi_register_multicast_address

This function registers interest in a multicast address.

### *Syntax*

```
int
wifi_register_multicast_address (
        rtw_mac_t *mac
        );
```

### *Parameters*

*mac*

Ethernet MAC address.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

## 4.20.2 wifi_unregister_multicast_address

This function unregisters interest in a multicast address.

### *Syntax*

```
int
wifi_unregister_multicast_address (
        rtw_mac_t *mac
```

```
        );
```

### *Parameters*

*mac*

Ethernet MAC address.

### *Return Value*

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

### *Remarks*

NONE.

# 4.21 Wifi WPS

## 4.21.1 wifi_set_wps_phase

This function sets wps phase.

### *Syntax*

```
int
wifi_set_wps_phase (
        unsigned char is_trigger_wps
        );
```

### *Parameters*

*is_trigger_wps*

set 1/0 to enable/disable the wps phase.

### *Return Value*

Return 0 if is_trigger_wps is 1 or 0, otherwise return -1.

### *Remarks*

NONE.

# 4.22 Wifi Adaptivity

## 4.22.1 wifi_set_mib

This function can call wext_set_adaptivity to setup adaptivity.

### *Syntax*

```
int
wext_set_adaptivity (
        rtw_adaptivity_mode_t adaptivity_mode
        );
```

### *Parameters*

*Adaptivity mode*

set adpativity mode. RTW_ADAPTIVITY_DISABLE is disable, RTW_ADAPTIVITY_NORMAL is for CE and RTW_ADAPTIVITY_CARRIER_SENSE is for MKK.

### *Return Value*

Return 0 if enable is 1 or 0, otherwise return -1.

### *Remarks*

NONE.