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 API Specific Data Types

2.1 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_TEMPORA  = 9
        RY
        RTW_BUFFER_UNAVAILABLE_PERMANE  = 10
        NT
        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.

2.2 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.

2.3 rtw_scan_type_t

```
typedef enum
{
        RTW_SCAN_TYPE_ACTIVE                = 0x00,
        RTW_SCAN_TYPE_PASSIVE               = 0x01,
        RTW_SCAN_TYPE_PROHIBITED_CHAN       = 0x04
        NELS
} rtw_802_11_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.

2.4 rtw_bss_type_t

```
typedef enum
{
        RTW_BSS_TYPE_INFRASTRUC    = 0,
        TURE
        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_UNKNOWN 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.

2.5 rtw_wps_type_t

```
typedef enum
{
	RTW_WPS_TYPE_DEFAULT		= 0x0000,
	RTW_WPS_TYPE_USER_SPECIFIED	= 0x0001,
	RTW_WPS_TYPE_MACHINE_SPECIFI	= 0x0002,
	ED
	RTW_WPS_TYPE_REKEY		= 0x0003,
	RTW_WPS_TYPE_PUSHBUTTON		= 0x0004,
	RTW_WPS_TYPE_REGISTRAR_SPECI	= 0x0005,
	FIED
	RTW_WPS_TYPE_NONE		= 0x0006
}rtw_wps_type_t;
```

The enumeration lists the wps type.

2.6 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.

2.7 rtw_security_t

```
typedef enum
{
        RTW_SECURITY_OPEN           = 0,
        RTW_SECURITY_WEP_PSK        = WEP_ENABLED,
        RTW_SECURITY_WEP_SHARE      = ( WEP_ENABLED | SHARED_ENABLED ),
        D
        RTW_SECURITY_WPA_TKIP_P     = ( WPA_SECURITY  | TKIP_ENABLED ),
        SK
        RTW_SECURITY_WPA_AES_PS     = ( WPA_SECURITY  | AES_ENABLED ),
        K
        RTW_SECURITY_WPA2_AES_P     = ( WPA2_SECURITY | AES_ENABLED ),
        SK
        RTW_SECURITY_WPA2_TKIP_     = ( WPA2_SECURITY | TKIP_ENABLED ),
        PSK
        RTW_SECURITY_WPA2_MIXE      = ( WPA2_SECURITY | AES_ENABLED | TKIP_ENABLED ),
        D_PSK
        RTW_SECURITY_WPS_OPEN       = WPS_ENABLED,
        RTW_SECURITY_WPS_SECUR      = (WPS_ENABLED | AES_ENABLED),
        E
        RTW_SECURITY_UNKNOWN        = -1,
        RTW_SECURITY_FORCE_32_B     = 0x7fffffff
        IT
}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.

2.8 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.

2.9 rtw_country_code_t

```
typedef enum
{
        RTW_COUNTRY_US              = 0,
        RTW_COUNTRY_EU              ,
        RTW_COUNTRY_JP              ,
        RTW_COUNTRY_CN
}rtw_country_code_t;
```

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

2.10 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 .

2.11 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 .

2.12 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 datas 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.

2.13 rtw_rcr_level_t

```
typedef enum
{
        RTW_PROMISC_DISABLE         = 0,
        RTW_PROMISC_ENABLE          = 1,
        RTW_PROMISC_ENABLE_1        = 2
} rtw_rcr_level_t;
```

The enumeration lists the promisc level. RTW_PROMISC_DISABLE means disable the promisc, RTW_PROMISC_ENABLE means enable the promisc. RTW_PROMISC_ENABLE_1 is used to enable the promisc special when the length is used.

2.14 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.

2.15 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.

2.16 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 char                   channel;
        rtw_802_11_band_t               band;
}rtw_scan_result_t;
```

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

2.17 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.

2.18 rtw_network_info_t

```
typedef struct rtw_network_info
{
        rtw_ssid_t                      ssid;
        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 and password should not exceed 32.

2.19 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 and password should not exceed 32.

2.20 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[33];
        unsigned char               key_idx;
} rtw_wifi_setting_t;
```

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

2.21 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[32];
        unsigned char               password_len;
        unsigned char               channel;
} rtw_wifi_config_t;
```

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

2.22 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.

2.23 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_maclist_t;
```

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

2.24 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_MAX,
} rtw_event_indicate_t;
```

This enumeration is event type indicated from wlan driver.

2.25 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.

2.26 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

2.27 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 Application Programming Interface

3.1 Wifi enable/disable

3.1.1 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

3.1.2 wifi_off

```
Int
wifi_off(
        void
        )
```

Parameters

None

Return Value

If the function succeeds, the return value is 0

3.1.3 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_MODE_STA_AP.

Return Value

If the function succeeds, the return value is 0

3.1.4 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_MODE_STA_AP.

Return Value

If the function succeeds, the return value is 0

3.2 Station Mode Connection

3.2.1 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.

3.2.2 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

3.3 AP Mode Startup

3.3.1 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

3.3.2 wifi_restart_ap

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

Syntax

```
int
```

```
wifi_start_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

3.3.3 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

3.3.4 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

3.4 Wifi Setting Information

3.4.1 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

3.4.2 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.

3.5 Wifi RF Control

3.5.1 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.

3.5.2 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

3.6 Wifi RSSI Information

3.6.1 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.

3.7 Country Code Setup

3.7.1 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.

3.8 Network Mode Setup

3.8.1 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.

3.9 Wifi Scan List

3.9.1 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 the count of all scanned AP. Otherwise the return value is -1. If the return count is bigger than the count parsed from buffer, it indicated that the buffer length is not enough to store all scanned AP information.

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.

3.10 Wlan Driver Indication

3.10.1 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_RECONNECTION_FAIL

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

4: WIFI_EVENT_SCAN_DONE

5: WIFI_EVENT_RECONNECTION_FAIL

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, some times 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().

3.11 Wifi Partial Channel Scan

3.11.1 wifi_set_pscan_chan

This function sets the channel used to be partial scanned.

Syntax

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

Parameters

channel_list

An array stores the channel list.

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.

3.12 Wifi Packet filter

3.12.1 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.

3.12.2 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 (
        u8 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.

3.12.3 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.

3.12.4 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.

3.12.5 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.

3.13 Wifi Promiscuous Mode

3.13.1 wifi_set_promisc

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

Syntax

```
void
wifi_set_promisc (
        unsigned char enabled,
        void (*callback)(unsigned char*, unsigned char*, unsifned int, void*),
        unsigned char len_used,
        unsigned customer
);
```

Parameters

enabled

Enable or disable promiscuous mode. 0 means disable the promiscuous mode, 1 means enable the promiscuous, 2 is used special for the len_used.

callback

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

len_used

If len_used set to 1, packet length will be saved and transferred to callback function.

customer

If customer set to 1, packet function with extra packet information will be reported.

Return Value

None.

Remarks

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

3.13.2 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.

3.14 Wifi Auto Reconnection

3.14.1 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 in “autoconfig.h” needs to be done before compiling, or this API won’t be effective.

3.14.2 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 in “autoconfig.h” needs to be done before compiling, or this API won’t be effective.

3.15 Wifi Custom IE

3.15.1 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 “autoconfig.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.

3.15.2 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 “autoconfig.h” needs to be done before compiling, or this API won’t be effective.

3.15.3 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 “autoconfig.h” needs to be done before compiling, or this API won’t be effective.

3.16 Wifi Mac Address

3.16.1 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.

Return Value

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

Remarks

NONE.

3.16.2 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.

Return Value

Return RTW_SUCCESS if success, otherwise return RTW_ERROR.

Remarks

NONE.

3.17 Wifi Power save

3.17.1 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.

3.17.2 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.

3.18 Wifi Tx Power

3.18.1 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.

3.18.2 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.

3.19 Wifi Channel

3.19.1 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.

3.19.2 wifi_get_channel

This function gets the current channel on STA interface.

Syntax

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

Parameters

poweridx

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.

3.20 Wifi Multicast Address

3.20.1 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.

3.20.2 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.

3.21 Wifi WPS

3.21.1 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.