



---

# WIFI Reference Manual

C API Reference

© NXP Semiconductors, 2008-2020

Generated by Doxygen 1.8.18



<b>1 Main Page</b>	<b>1</b>
1.1 Introduction	1
1.1.1 Developer Documentation	1
<b>2 Data Structure Index</b>	<b>3</b>
2.1 Data Structures	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Data Structure Documentation</b>	<b>7</b>
4.1 cli_command Struct Reference	7
4.1.1 Detailed Description	7
4.1.2 Field Documentation	7
4.1.2.1 name	7
4.1.2.2 help	7
4.1.2.3 function	8
4.2 ipv4_config Struct Reference	8
4.2.1 Detailed Description	8
4.2.2 Field Documentation	8
4.2.2.1 addr_type	8
4.2.2.2 address	8
4.2.2.3 gw	9
4.2.2.4 netmask	9
4.2.2.5 dns1	9
4.2.2.6 dns2	9
4.3 os_queue_pool_t Struct Reference	9
4.3.1 Detailed Description	9
4.3.2 Field Documentation	9
4.3.2.1 size	10
4.4 os_thread_stack_t Struct Reference	10
4.4.1 Detailed Description	10
4.4.2 Field Documentation	10
4.4.2.1 size	10
4.5 wifi_antcfg_t Struct Reference	10
4.5.1 Detailed Description	11
4.5.2 Field Documentation	11
4.5.2.1 ant_mode	11
4.5.2.2 evaluate_time	11
4.6 wifi_auto_reconnect_config_t Struct Reference	11
4.6.1 Detailed Description	11
4.6.2 Field Documentation	11
4.6.2.1 reconnect_counter	11

4.6.2.2 reconnect_interval . . . . .	12
4.6.2.3 flags . . . . .	12
4.7 wifi_bandcfg_t Struct Reference . . . . .	12
4.7.1 Detailed Description . . . . .	12
4.7.2 Field Documentation . . . . .	12
4.7.2.1 config_bands . . . . .	12
4.7.2.2 fw_bands . . . . .	12
4.8 wifi_cal_data_t Struct Reference . . . . .	13
4.8.1 Detailed Description . . . . .	13
4.8.2 Field Documentation . . . . .	13
4.8.2.1 data_len . . . . .	13
4.8.2.2 data . . . . .	13
4.9 wifi_chan_info_t Struct Reference . . . . .	13
4.9.1 Detailed Description . . . . .	13
4.9.2 Field Documentation . . . . .	14
4.9.2.1 chan_num . . . . .	14
4.9.2.2 chan_freq . . . . .	14
4.9.2.3 passive_scan_or_radar_detect . . . . .	14
4.10 wifi_chan_list_param_set_t Struct Reference . . . . .	14
4.10.1 Detailed Description . . . . .	14
4.10.2 Field Documentation . . . . .	14
4.10.2.1 no_of_channels . . . . .	15
4.10.2.2 chan_scan_param . . . . .	15
4.11 wifi_chan_scan_param_set_t Struct Reference . . . . .	15
4.11.1 Detailed Description . . . . .	15
4.11.2 Field Documentation . . . . .	15
4.11.2.1 chan_number . . . . .	15
4.11.2.2 min_scan_time . . . . .	15
4.11.2.3 max_scan_time . . . . .	16
4.12 wifi_chanlist_t Struct Reference . . . . .	16
4.12.1 Detailed Description . . . . .	16
4.12.2 Field Documentation . . . . .	16
4.12.2.1 num_chans . . . . .	16
4.12.2.2 chan_info . . . . .	16
4.13 wifi_channel_desc_t Struct Reference . . . . .	17
4.13.1 Detailed Description . . . . .	17
4.13.2 Field Documentation . . . . .	17
4.13.2.1 start_freq . . . . .	17
4.13.2.2 chan_width . . . . .	17
4.13.2.3 chan_num . . . . .	17
4.14 wifi_cw_mode_ctrl_t Struct Reference . . . . .	18
4.14.1 Detailed Description . . . . .	18

4.14.2 Field Documentation	18
4.14.2.1 mode	18
4.14.2.2 channel	18
4.14.2.3 chanInfo	18
4.14.2.4 txPower	18
4.14.2.5 pktLength	19
4.14.2.6 rateInfo	19
4.15 wifi_data_rate_t Struct Reference	19
4.15.1 Detailed Description	19
4.15.2 Field Documentation	19
4.15.2.1 tx_data_rate	19
4.15.2.2 rx_data_rate	20
4.15.2.3 tx_ht_bw	20
4.15.2.4 tx_ht_gi	20
4.15.2.5 rx_ht_bw	20
4.15.2.6 rx_ht_gi	20
4.15.2.7 tx_mcs_index	20
4.15.2.8 rx_mcs_index	20
4.15.2.9 tx_rate_format	20
4.15.2.10 rx_rate_format	21
4.16 wifi_domain_param_t Struct Reference	21
4.16.1 Detailed Description	21
4.16.2 Field Documentation	21
4.16.2.1 country_code	21
4.16.2.2 no_of_sub_band	21
4.16.2.3 sub_band	22
4.17 wifi_ds_rate Struct Reference	22
4.17.1 Detailed Description	22
4.17.2 Field Documentation	22
4.17.2.1 sub_command	22
4.17.2.2 rate_cfg	22
4.17.2.3 data_rate	22
4.17.2.4 param	23
4.18 wifi_ed_mac_ctrl_t Struct Reference	23
4.18.1 Detailed Description	23
4.18.2 Field Documentation	23
4.18.2.1 ed_ctrl_2g	23
4.18.2.2 ed_offset_2g	23
4.19 wififlt_cfg_t Struct Reference	24
4.19.1 Detailed Description	24
4.19.2 Field Documentation	24
4.19.2.1 criteria	24

4.19.2.2 nentries	24
4.19.2.3 mef_entry	24
4.20 wifi_fw_version_ext_t Struct Reference	24
4.20.1 Detailed Description	25
4.20.2 Field Documentation	25
4.20.2.1 version_str_sel	25
4.20.2.2 version_str	25
4.21 wifi_fw_version_t Struct Reference	25
4.21.1 Detailed Description	25
4.21.2 Field Documentation	25
4.21.2.1 version_str	25
4.22 wifi_mac_addr_t Struct Reference	26
4.22.1 Detailed Description	26
4.22.2 Field Documentation	26
4.22.2.1 mac	26
4.23 wifi_mef_entry_t Struct Reference	26
4.23.1 Detailed Description	26
4.23.2 Field Documentation	26
4.23.2.1 mode	27
4.23.2.2 action	27
4.23.2.3 filter_num	27
4.23.2.4 filter_item	27
4.23.2.5 rpn	27
4.24 wifi_mef_filter_t Struct Reference	27
4.24.1 Detailed Description	28
4.24.2 Field Documentation	28
4.24.2.1 type	28
4.24.2.2 pattern	28
4.24.2.3 offset	28
4.24.2.4 num_bytes	28
4.24.2.5 repeat	28
4.24.2.6 num_byte_seq	28
4.24.2.7 byte_seq	29
4.24.2.8 num_mask_seq	29
4.24.2.9 mask_seq	29
4.25 wifi_mgmt_frame_t Struct Reference	29
4.25.1 Detailed Description	29
4.25.2 Field Documentation	29
4.25.2.1 frm_len	30
4.25.2.2 frame_type	30
4.25.2.3 frame_ctrl_flags	30
4.25.2.4 duration_id	30

4.25.2.5 addr1	30
4.25.2.6 addr2	30
4.25.2.7 addr3	30
4.25.2.8 seq_ctl	30
4.25.2.9 addr4	31
4.25.2.10 payload	31
4.26 wifi_nat_keep_alive_t Struct Reference	31
4.26.1 Detailed Description	31
4.26.2 Field Documentation	31
4.26.2.1 interval	31
4.26.2.2 dst_mac	31
4.26.2.3 dst_ip	32
4.26.2.4 dst_port	32
4.27 wifi_rate_cfg_t Struct Reference	32
4.27.1 Detailed Description	32
4.27.2 Field Documentation	32
4.27.2.1 rate_format	32
4.27.2.2 rate_index	32
4.27.2.3 rate	33
4.28 wifi_remain_on_channel_t Struct Reference	33
4.28.1 Detailed Description	33
4.28.2 Field Documentation	33
4.28.2.1 remove	33
4.28.2.2 status	33
4.28.2.3 bandcfg	33
4.28.2.4 channel	34
4.28.2.5 remain_period	34
4.29 wifi_rf_channel_t Struct Reference	34
4.29.1 Detailed Description	34
4.29.2 Field Documentation	34
4.29.2.1 current_channel	34
4.29.2.2 rf_type	34
4.30 wifi_rssi_info_t Struct Reference	35
4.30.1 Detailed Description	35
4.30.2 Field Documentation	35
4.30.2.1 data_rssi_last	35
4.30.2.2 data_nf_last	35
4.30.2.3 data_rssi_avg	35
4.30.2.4 data_nf_avg	35
4.30.2.5 bcn_snr_last	36
4.30.2.6 bcn_snr_avg	36
4.30.2.7 data_snr_last	36

4.30.2.8 data_snr_avg . . . . .	36
4.30.2.9 bcn_rssi_last . . . . .	36
4.30.2.10 bcn_nf_last . . . . .	36
4.30.2.11 bcn_rssi_avg . . . . .	36
4.30.2.12 bcn_nf_avg . . . . .	37
4.31 wifi_scan_chan_list_t Struct Reference . . . . .	37
4.31.1 Detailed Description . . . . .	37
4.31.2 Field Documentation . . . . .	37
4.31.2.1 num_of_chan . . . . .	37
4.31.2.2 chan_number . . . . .	37
4.32 wifi_scan_channel_list_t Struct Reference . . . . .	38
4.32.1 Detailed Description . . . . .	38
4.32.2 Field Documentation . . . . .	38
4.32.2.1 chan_number . . . . .	38
4.32.2.2 scan_type . . . . .	38
4.32.2.3 scan_time . . . . .	38
4.33 wifi_scan_params_v2_t Struct Reference . . . . .	38
4.33.1 Detailed Description . . . . .	39
4.33.2 Field Documentation . . . . .	39
4.33.2.1 bssid . . . . .	39
4.33.2.2 ssid . . . . .	39
4.33.2.3 num_channels . . . . .	39
4.33.2.4 chan_list . . . . .	39
4.33.2.5 num_probes . . . . .	39
4.33.2.6 cb . . . . .	39
4.34 wifi_scan_result Struct Reference . . . . .	40
4.34.1 Detailed Description . . . . .	40
4.34.2 Field Documentation . . . . .	40
4.34.2.1 bssid . . . . .	40
4.34.2.2 is_ibss_bit_set . . . . .	40
4.34.2.3 ssid . . . . .	41
4.34.2.4 ssid_len . . . . .	41
4.34.2.5 Channel . . . . .	41
4.34.2.6 RSSI . . . . .	41
4.34.2.7 beacon_period . . . . .	41
4.34.2.8 dtim_period . . . . .	41
4.34.2.9 WPA_WPA2_WEP . . . . .	41
4.34.2.10 wpa_mcstCipher . . . . .	41
4.34.2.11 wpa_ucstCipher . . . . .	42
4.34.2.12 rsn_mcstCipher . . . . .	42
4.34.2.13 rsn_ucstCipher . . . . .	42
4.34.2.14 is_pmf_required . . . . .	42



4.34.2.15 phtcap_ie_present . . . . .	42
4.34.2.16 phtinfo_ie_present . . . . .	42
4.34.2.17 wmm_ie_present . . . . .	42
4.34.2.18 band . . . . .	43
4.34.2.19 wps_IE_exist . . . . .	43
4.34.2.20 wps_session . . . . .	43
4.34.2.21 wpa2_entp_IE_exist . . . . .	43
4.34.2.22 trans_mode . . . . .	43
4.34.2.23 trans_bssid . . . . .	43
4.34.2.24 trans_ssid . . . . .	43
4.34.2.25 trans_ssid_len . . . . .	44
4.35 wifi_sta_info_t Struct Reference . . . . .	44
4.35.1 Detailed Description . . . . .	44
4.35.2 Field Documentation . . . . .	44
4.35.2.1 mac . . . . .	44
4.35.2.2 power_mgmt_status . . . . .	44
4.35.2.3 rssi . . . . .	44
4.36 wifi_sta_list_t Struct Reference . . . . .	45
4.36.1 Detailed Description . . . . .	45
4.36.2 Field Documentation . . . . .	45
4.36.2.1 count . . . . .	45
4.37 wifi_sub_band_set_t Struct Reference . . . . .	45
4.37.1 Detailed Description . . . . .	45
4.37.2 Field Documentation . . . . .	45
4.37.2.1 first_chan . . . . .	46
4.37.2.2 no_of_chan . . . . .	46
4.37.2.3 max_tx_pwr . . . . .	46
4.38 wifi_tbtt_offset_t Struct Reference . . . . .	46
4.38.1 Detailed Description . . . . .	46
4.38.2 Field Documentation . . . . .	46
4.38.2.1 min_tbtt_offset . . . . .	46
4.38.2.2 max_tbtt_offset . . . . .	47
4.38.2.3 avg_tbtt_offset . . . . .	47
4.39 wifi_tcp_keep_alive_t Struct Reference . . . . .	47
4.39.1 Detailed Description . . . . .	47
4.39.2 Field Documentation . . . . .	47
4.39.2.1 enable . . . . .	47
4.39.2.2 reset . . . . .	48
4.39.2.3 timeout . . . . .	48
4.39.2.4 interval . . . . .	48
4.39.2.5 max_keep_alives . . . . .	48
4.39.2.6 dst_mac . . . . .	48

4.39.2.7 dst_ip	48
4.39.2.8 dst_tcp_port	48
4.39.2.9 src_tcp_port	48
4.39.2.10 seq_no	49
4.40 wifi_tx_power_t Struct Reference	49
4.40.1 Detailed Description	49
4.40.2 Field Documentation	49
4.40.2.1 current_level	49
4.40.2.2 max_power	49
4.40.2.3 min_power	49
4.41 wifi_txpwrlimit_config_t Struct Reference	50
4.41.1 Detailed Description	50
4.41.2 Field Documentation	50
4.41.2.1 num_mod_grps	50
4.41.2.2 chan_desc	50
4.41.2.3 txpwrlimit_entry	50
4.42 wifi_txpwrlimit_entry_t Struct Reference	50
4.42.1 Detailed Description	51
4.42.2 Field Documentation	51
4.42.2.1 mod_group	51
4.42.2.2 tx_power	51
4.43 wifi_txpwrlimit_t Struct Reference	51
4.43.1 Detailed Description	52
4.43.2 Field Documentation	52
4.43.2.1 subband	52
4.43.2.2 num_chans	52
4.43.2.3 txpwrlimit_config	52
4.44 wlan_cipher Struct Reference	52
4.44.1 Detailed Description	52
4.44.2 Field Documentation	53
4.44.2.1 wep40	53
4.44.2.2 wep104	53
4.44.2.3 tkip	53
4.44.2.4 ccmp	53
4.44.2.5 rsvd	53
4.45 wlan_ip_config Struct Reference	53
4.45.1 Detailed Description	54
4.45.2 Field Documentation	54
4.45.2.1 ipv4	54
4.46 wlan_network Struct Reference	54
4.46.1 Detailed Description	54
4.46.2 Field Documentation	55

4.46.2.1 name . . . . .	55
4.46.2.2 ssid . . . . .	55
4.46.2.3 bssid . . . . .	55
4.46.2.4 channel . . . . .	55
4.46.2.5 type . . . . .	55
4.46.2.6 role . . . . .	56
4.46.2.7 security . . . . .	56
4.46.2.8 ip . . . . .	56
4.46.2.9 ssid_specific . . . . .	56
4.46.2.10 bssid_specific . . . . .	56
4.46.2.11 channel_specific . . . . .	56
4.46.2.12 security_specific . . . . .	57
4.46.2.13 beacon_period . . . . .	57
4.46.2.14 dtim_period . . . . .	57
4.47 wlan_network_security Struct Reference . . . . .	57
4.47.1 Detailed Description . . . . .	57
4.47.2 Field Documentation . . . . .	57
4.47.2.1 type . . . . .	58
4.47.2.2 mcstCipher . . . . .	58
4.47.2.3 ucstCipher . . . . .	58
4.47.2.4 is_pmf_required . . . . .	58
4.47.2.5 psk . . . . .	58
4.47.2.6 psk_len . . . . .	58
4.47.2.7 password . . . . .	58
4.47.2.8 password_len . . . . .	59
4.47.2.9 pmk . . . . .	59
4.47.2.10 pmk_valid . . . . .	59
4.47.2.11 mfpc . . . . .	59
4.47.2.12 mfpr . . . . .	59
4.48 wlan_scan_result Struct Reference . . . . .	60
4.48.1 Detailed Description . . . . .	60
4.48.2 Field Documentation . . . . .	60
4.48.2.1 ssid . . . . .	60
4.48.2.2 ssid_len . . . . .	60
4.48.2.3 bssid . . . . .	60
4.48.2.4 channel . . . . .	61
4.48.2.5 type . . . . .	61
4.48.2.6 role . . . . .	61
4.48.2.7 wmm . . . . .	61
4.48.2.8 wpa2_entp . . . . .	61
4.48.2.9 wep . . . . .	61
4.48.2.10 wpa . . . . .	61

4.48.2.11 wpa2	62
4.48.2.12 wpa3_sae	62
4.48.2.13 rssi	62
4.48.2.14 trans_ssid	62
4.48.2.15 trans_ssid_len	62
4.48.2.16 trans_bssid	62
4.48.2.17 beacon_period	62
4.48.2.18 dtim_period	62
<b>5 File Documentation</b>	<b>63</b>
5.1 cli.h File Reference	63
5.1.1 Detailed Description	63
5.1.2 Usage	63
5.1.3 Function Documentation	63
5.1.3.1 cli_register_command()	63
5.1.3.2 cli_unregister_command()	64
5.1.3.3 cli_init()	64
5.1.3.4 cli_stop()	64
5.1.3.5 cli_register_commands()	65
5.1.3.6 cli_unregister_commands()	65
5.2 dhcp-server.h File Reference	65
5.2.1 Detailed Description	65
5.2.2 Function Documentation	66
5.2.2.1 dhcpd_cli_init()	66
5.2.2.2 dhcp_server_start()	66
5.2.2.3 dhcp_enable_dns_server()	66
5.2.2.4 dhcp_server_stop()	67
5.2.2.5 dhcp_server_lease_timeout()	67
5.2.2.6 dhcp_get_ip_from_mac()	68
5.2.2.7 dhcp_stat()	68
5.2.3 Enumeration Type Documentation	68
5.2.3.1 wm_dhcpd_errno	68
5.3 iperf.h File Reference	69
5.3.1 Function Documentation	69
5.3.1.1 iperf_cli_init()	69
5.3.1.2 iperf_cli_deinit()	69
5.4 wifi-decl.h File Reference	70
5.4.1 Macro Documentation	70
5.4.1.1 MLAN_MAX_VER_STR_LEN	70
5.4.1.2 BSS_TYPE_STA	70
5.4.1.3 BSS_TYPE_UAP	70
5.4.1.4 MLAN_MAX_SSID_LENGTH	70

5.4.1.5 MLAN_MAX_PASS_LENGTH . . . . .	70
5.4.2 Enumeration Type Documentation . . . . .	70
5.4.2.1 wifi_SubBand_t . . . . .	70
5.4.2.2 wifi_frame_type_t . . . . .	71
5.5 wifi.h File Reference . . . . .	71
5.5.1 Function Documentation . . . . .	71
5.5.1.1 wifi_init() . . . . .	72
5.5.1.2 wifi_init_fcc() . . . . .	72
5.5.1.3 wifi_deinit() . . . . .	72
5.5.1.4 wifi_register_data_input_callback() . . . . .	73
5.5.1.5 wifi_deregister_data_input_callback() . . . . .	73
5.5.1.6 wifi_register_amsdu_data_input_callback() . . . . .	73
5.5.1.7 wifi_deregister_amsdu_data_input_callback() . . . . .	74
5.5.1.8 wifi_low_level_output() . . . . .	74
5.5.1.9 wifi_set_packet_retry_count() . . . . .	74
5.5.1.10 wifi_sta_ampdu_tx_enable() . . . . .	75
5.5.1.11 wifi_sta_ampdu_tx_disable() . . . . .	75
5.5.1.12 wifi_sta_ampdu_rx_enable() . . . . .	75
5.5.1.13 wifi_sta_ampdu_rx_disable() . . . . .	75
5.5.1.14 wifi_get_device_mac_addr() . . . . .	75
5.5.1.15 wifi_get_device_firmware_version_ext() . . . . .	76
5.5.1.16 wifi_get_last_cmd_sent_ms() . . . . .	76
5.5.1.17 wifi_update_last_cmd_sent_ms() . . . . .	76
5.5.1.18 wifi_register_event_queue() . . . . .	76
5.5.1.19 wifi_unregister_event_queue() . . . . .	77
5.5.1.20 wifi_get_scan_result() . . . . .	77
5.5.1.21 wifi_get_scan_result_count() . . . . .	78
5.5.1.22 wifi_uap_bss_sta_list() . . . . .	78
5.5.1.23 wifi_set_cal_data() . . . . .	79
5.5.1.24 wifi_set_mac_addr() . . . . .	79
5.5.1.25 _wifi_set_mac_addr() . . . . .	79
5.5.1.26 wifi_add_mcast_filter() . . . . .	80
5.5.1.27 wifi_remove_mcast_filter() . . . . .	80
5.5.1.28 wifi_get_ipv4_multicast_mac() . . . . .	80
5.5.1.29 wifi_get_region_code() . . . . .	81
5.5.1.30 wifi_set_region_code() . . . . .	81
5.5.1.31 wifi_get_uap_channel() . . . . .	82
5.5.1.32 wifi_enable_11d_support() . . . . .	82
5.5.2 Enumeration Type Documentation . . . . .	83
5.5.2.1 anonymous enum . . . . .	83
5.5.2.2 country_code_t . . . . .	83
5.6 wifi_events.h File Reference . . . . .	84

5.6.1 Enumeration Type Documentation	84
5.6.1.1 wifi_event	84
5.6.1.2 wifi_event_reason	85
5.6.1.3 wlan_bss_type	85
5.6.1.4 wlan_bss_role	86
5.6.1.5 wifi_wakeup_event_t	86
5.7 wlan.h File Reference	86
5.7.1 Detailed Description	86
5.7.2 Usage	87
5.7.3 Function Documentation	87
5.7.3.1 wlan_init()	87
5.7.3.2 wlan_start()	87
5.7.3.3 wlan_stop()	88
5.7.3.4 wlan_deinit()	88
5.7.3.5 wlan_initialize_uap_network()	89
5.7.3.6 wlan_add_network()	89
5.7.3.7 wlan_remove_network()	90
5.7.3.8 wlan_connect()	90
5.7.3.9 wlan_disconnect()	91
5.7.3.10 wlan_start_network()	92
5.7.3.11 wlan_stop_network()	92
5.7.3.12 wlan_get_mac_address()	93
5.7.3.13 wlan_get_address()	93
5.7.3.14 wlan_get_uap_address()	94
5.7.3.15 wlan_get_uap_channel()	94
5.7.3.16 wlan_get_current_network()	95
5.7.3.17 wlan_get_current_uap_network()	95
5.7.3.18 is_uap_started()	96
5.7.3.19 is_sta_connected()	96
5.7.3.20 is_sta_ipv4_connected()	96
5.7.3.21 wlan_get_network()	97
5.7.3.22 wlan_get_network_byname()	97
5.7.3.23 wlan_get_network_count()	98
5.7.3.24 wlan_get_connection_state()	98
5.7.3.25 wlan_get_uap_connection_state()	99
5.7.3.26 wlan_scan()	99
5.7.3.27 wlan_scan_with_opt()	100
5.7.3.28 wlan_get_scan_result()	100
5.7.3.29 wlan_set_ed_mac_mode()	101
5.7.3.30 wlan_get_ed_mac_mode()	102
5.7.3.31 wlan_set_cal_data()	102
5.7.3.32 wlan_set_mac_addr()	103

5.7.3.33 wlan_configure_listen_interval()	103
5.7.3.34 wlan_configure_null_pkt_interval()	104
5.7.3.35 wlan_set_antcfg()	104
5.7.3.36 wlan_get_antcfg()	105
5.7.3.37 wlan_get_firmware_version_ext()	105
5.7.3.38 wlan_version_extended()	106
5.7.3.39 wlan_get_tsf()	106
5.7.3.40 wlan_ieeepps_on()	106
5.7.3.41 wlan_ieeepps_off()	107
5.7.3.42 wlan_deepsleeps_on()	107
5.7.3.43 wlan_deepsleeps_off()	108
5.7.3.44 wlan_get_beacon_period()	108
5.7.3.45 wlan_get_dtim_period()	108
5.7.3.46 wlan_get_data_rate()	108
5.7.3.47 wlan_set_pmfcfg()	109
5.7.3.48 wlan_get_pmfcfg()	109
5.7.3.49 wlan_set_packet_filters()	110
5.7.3.50 wlan_set_auto_arp()	113
5.7.3.51 wlan_send_host_sleep()	113
5.7.3.52 wlan_get_current_bssid()	113
5.7.3.53 wlan_get_current_channel()	114
5.7.3.54 wlan_get_ps_mode()	114
5.7.3.55 wlan_wlcmgr_send_msg()	114
5.7.3.56 wlan_wfa_basic_cli_init()	115
5.7.3.57 wlan_basic_cli_init()	115
5.7.3.58 wlan_cli_init()	116
5.7.3.59 wlan_enhanced_cli_init()	116
5.7.3.60 wlan_get_uap_supported_max_clients()	117
5.7.3.61 wlan_get_uap_max_clients()	117
5.7.3.62 wlan_set_uap_max_clients()	117
5.7.3.63 wlan_set_htcapinfo()	118
5.7.3.64 wlan_set_httxcfg()	118
5.7.3.65 wlan_set_txratecfg()	120
5.7.3.66 wlan_get_txratecfg()	122
5.7.3.67 wlan_get_sta_tx_power()	122
5.7.3.68 wlan_set_sta_tx_power()	122
5.7.3.69 wlan_set_wwsm_txpwrlimit()	123
5.7.3.70 wlan_get_mgmt_ie()	123
5.7.3.71 wlan_set_mgmt_ie()	124
5.7.3.72 wlan_clear_mgmt_ie()	124
5.7.3.73 wlan_get_11d_enable_status()	124
5.7.3.74 wlan_get_current_signal_strength()	125

5.7.3.75 wlan_get_average_signal_strength()	125
5.7.3.76 wlan_remain_on_channel()	125
5.7.3.77 wlan_get_otp_user_data()	126
5.7.3.78 wlan_get_cal_data()	126
5.7.3.79 wlan_set_chanlist_and_txpwrlimit()	127
5.7.3.80 wlan_set_chanlist()	127
5.7.3.81 wlan_get_chanlist()	128
5.7.3.82 wlan_set_txpwrlimit()	128
5.7.3.83 wlan_get_txpwrlimit()	129
5.7.3.84 wlan_set_reassoc_control()	129
5.7.3.85 wlan_uap_set_beacon_period()	130
5.7.3.86 wlan_uap_set_bandwidth()	130
5.7.3.87 wlan_uap_set_hidden_ssid()	130
5.7.3.88 wlan_uap_ctrl_deauth()	131
5.7.3.89 wlan_uap_set_ecsa()	131
5.7.3.90 wlan_uap_set_htcapinfo()	131
5.7.3.91 wlan_uap_set_htxcfg()	132
5.7.3.92 wlan_sta_ampdu_tx_enable()	133
5.7.3.93 wlan_sta_ampdu_tx_disable()	133
5.7.3.94 wlan_sta_ampdu_rx_enable()	133
5.7.3.95 wlan_sta_ampdu_rx_disable()	133
5.7.3.96 wlan_uap_set_scan_chan_list()	133
5.7.3.97 wlan_set_crypto_RC4_encrypt()	134
5.7.3.98 wlan_set_crypto_RC4_decrypt()	134
5.7.3.99 wlan_set_crypto_AES_ECB_encrypt()	135
5.7.3.100 wlan_set_crypto_AES_ECB_decrypt()	136
5.7.3.101 wlan_set_crypto_AES_WRAP_encrypt()	136
5.7.3.102 wlan_set_crypto_AES_WRAP_decrypt()	137
5.7.3.103 wlan_set_crypto_AES_CCMP_encrypt()	138
5.7.3.104 wlan_set_crypto_AES_CCMP_decrypt()	139
5.7.3.105 wlan_set_crypto_AES_GCMP_encrypt()	139
5.7.3.106 wlan_set_crypto_AES_GCMP_decrypt()	140
5.7.3.107 wlan_send_hostcmd()	141
5.7.4 Macro Documentation	142
5.7.4.1 ACTION_GET	142
5.7.4.2 ACTION_SET	142
5.7.4.3 IEEEtypes_SSID_SIZE	142
5.7.4.4 IEEEtypes_ADDRESS_SIZE	142
5.7.4.5 WLAN_RESCAN_LIMIT	142
5.7.4.6 WLAN_RECONNECT_LIMIT	142
5.7.4.7 WLAN_NETWORK_NAME_MIN_LENGTH	142
5.7.4.8 WLAN_NETWORK_NAME_MAX_LENGTH	143



5.7.4.9 WLAN_PSK_MIN_LENGTH . . . . .	143
5.7.4.10 WLAN_MAX_KNOWN_NETWORKS . . . . .	143
5.7.4.11 WLAN_PMK_LENGTH . . . . .	143
5.7.4.12 WLAN_ERROR_NONE . . . . .	143
5.7.4.13 WLAN_ERROR_PARAM . . . . .	143
5.7.4.14 WLAN_ERROR_NOMEM . . . . .	143
5.7.4.15 WLAN_ERROR_STATE . . . . .	143
5.7.4.16 WLAN_ERROR_ACTION . . . . .	144
5.7.4.17 WLAN_ERROR_PS_ACTION . . . . .	144
5.7.4.18 WLAN_ERROR_NOT_SUPPORTED . . . . .	144
5.7.5 Typedef Documentation . . . . .	144
5.7.5.1 wlan_scan_channel_list_t . . . . .	144
5.7.5.2 wlan_scan_params_v2_t . . . . .	144
5.7.5.3 wlan_cal_data_t . . . . .	144
5.7.5.4 wlanflt_cfg_t . . . . .	144
5.7.5.5 wlan_wowlan_ptn_cfg_t . . . . .	145
5.7.5.6 wlan_tcp_keep_alive_t . . . . .	145
5.7.5.7 wlan_ds_rate . . . . .	145
5.7.5.8 wlan_ed_mac_ctrl_t . . . . .	145
5.7.5.9 wlan_bandcfg_t . . . . .	145
5.7.5.10 wlan_cw_mode_ctrl_t . . . . .	145
5.7.5.11 wlan_chanlist_t . . . . .	145
5.7.5.12 wlan_txpwrlimit_t . . . . .	145
5.7.6 Enumeration Type Documentation . . . . .	145
5.7.6.1 wm_wlan_errno . . . . .	145
5.7.6.2 wlan_event_reason . . . . .	146
5.7.6.3 wlan_wakeup_event_t . . . . .	147
5.7.6.4 wlan_connection_state . . . . .	147
5.7.6.5 wlan_ps_mode . . . . .	148
5.7.6.6 wlan_security_type . . . . .	148
5.7.6.7 address_types . . . . .	149
5.8 wlan_11d.h File Reference . . . . .	149
5.8.1 Function Documentation . . . . .	149
5.8.1.1 wlan_enable_11d() . . . . .	149
5.8.1.2 wlan_get_country() . . . . .	150
5.8.1.3 wlan_uap_set_country() . . . . .	150
5.8.1.4 wlan_set_country() . . . . .	151
5.8.1.5 wlan_set_domain_params() . . . . .	151
5.8.1.6 wlan_set_region_code() . . . . .	154
5.8.1.7 wlan_11d_country_index_2_string() . . . . .	155
5.9 wlan_tests.h File Reference . . . . .	155
5.9.1 Function Documentation . . . . .	155

5.9.1.1 print_txpwrlimit()	155
5.10 wm_net.h File Reference	156
5.10.1 Detailed Description	156
5.10.2 Function Documentation	156
5.10.2.1 net_dhcp_hostname_set()	156
5.10.2.2 net_stop_dhcp_timer()	156
5.10.2.3 net_socket_blocking()	156
5.10.2.4 net_get_sock_error()	157
5.10.2.5 net_inet_aton()	157
5.10.2.6 net_gethostbyname()	158
5.10.2.7 net_inet_ntoa()	158
5.10.2.8 net_is_ip_or_ipv6()	158
5.10.2.9 net_sock_to_interface()	159
5.10.2.10 net_wlan_init()	159
5.10.2.11 net_wlan_deinit()	159
5.10.2.12 net_get_sta_handle()	160
5.10.2.13 net_get_uap_handle()	160
5.10.2.14 net_interface_up()	160
5.10.2.15 net_interface_down()	161
5.10.2.16 net_interface_dhcp_stop()	161
5.10.2.17 net_configure_address()	161
5.10.2.18 net_configure_dns()	162
5.10.2.19 net_get_if_addr()	162
5.10.2.20 net_get_if_name()	162
5.10.2.21 net_get_if_ip_addr()	163
5.10.2.22 net_get_if_ip_mask()	163
5.10.2.23 net_ipv4stack_init()	164
5.10.2.24 net_stat()	164
5.11 wm_os.h File Reference	164
5.11.1 Detailed Description	164
5.11.2 Usage	165
5.11.3 Function Documentation	165
5.11.3.1 os_ticks_get()	165
5.11.3.2 os_get_timestamp()	165
5.11.3.3 os_msec_to_ticks()	166
5.11.3.4 os_ticks_to_msec()	166
5.11.3.5 os_thread_create()	166
5.11.3.6 os_thread_delete()	167
5.11.3.7 os_thread_sleep()	168
5.11.3.8 os_thread_self_complete()	168
5.11.3.9 os_queue_create()	169
5.11.3.10 os_queue_send()	169

5.11.3.11 os_queue_rcv()	170
5.11.3.12 os_queue_delete()	171
5.11.3.13 os_queue_get_msgs_waiting()	171
5.11.3.14 os_setup_idle_function()	171
5.11.3.15 os_setup_tick_function()	172
5.11.3.16 os_remove_idle_function()	172
5.11.3.17 os_remove_tick_function()	172
5.11.3.18 os_mutex_create()	173
5.11.3.19 os_mutex_get()	173
5.11.3.20 os_mutex_put()	174
5.11.3.21 os_recursive_mutex_create()	174
5.11.3.22 os_recursive_mutex_get()	175
5.11.3.23 os_recursive_mutex_put()	175
5.11.3.24 os_mutex_delete()	176
5.11.3.25 os_event_notify_get()	176
5.11.3.26 os_event_notify_put()	177
5.11.3.27 os_semaphore_create()	177
5.11.3.28 os_semaphore_create_counting()	178
5.11.3.29 os_semaphore_get()	178
5.11.3.30 os_semaphore_put()	179
5.11.3.31 os_semaphore_getcount()	179
5.11.3.32 os_semaphore_delete()	180
5.11.3.33 os_rwlock_create()	180
5.11.3.34 os_rwlock_delete()	181
5.11.3.35 os_rwlock_write_lock()	181
5.11.3.36 os_rwlock_write_unlock()	181
5.11.3.37 os_rwlock_read_lock()	182
5.11.3.38 os_rwlock_read_unlock()	182
5.11.3.39 os_timer_create()	183
5.11.3.40 os_timer_activate()	183
5.11.3.41 os_timer_change()	184
5.11.3.42 os_timer_is_running()	184
5.11.3.43 os_timer_get_context()	185
5.11.3.44 os_timer_reset()	185
5.11.3.45 os_timer_deactivate()	186
5.11.3.46 os_timer_delete()	186
5.11.3.47 os_mem_alloc()	186
5.11.3.48 os_mem_calloc()	187
5.11.3.49 os_mem_free()	187
5.11.3.50 os_disable_all_interrupts()	188
5.11.3.51 os_enable_all_interrupts()	188
5.11.4 Macro Documentation	188

5.11.4.1 os_thread_relinquish . . . . .	188
5.11.4.2 os_ticks_to_unblock . . . . .	188
5.11.4.3 os_thread_stack_define . . . . .	188
5.11.4.4 os_queue_pool_define . . . . .	189
5.11.4.5 OS_WAIT_FOREVER . . . . .	189
5.11.4.6 OS_NO_WAIT . . . . .	189
5.11.4.7 OS_MUTEX_INHERIT . . . . .	189
5.11.4.8 OS_MUTEX_NO_INHERIT . . . . .	189
5.11.4.9 os_get_runtime_stats . . . . .	189
5.11.5 Typedef Documentation . . . . .	189
5.11.5.1 cb_fn . . . . .	190
5.11.6 Enumeration Type Documentation . . . . .	190
5.11.6.1 os_timer_reload_t . . . . .	190
5.11.6.2 os_timer_activate_t . . . . .	190
5.12 wm_utils.h File Reference . . . . .	190
5.12.1 Detailed Description . . . . .	190
5.12.2 Function Documentation . . . . .	191
5.12.2.1 hex2bin() . . . . .	191
5.12.2.2 bin2hex() . . . . .	191
5.12.2.3 random_register_handler() . . . . .	192
5.12.2.4 random_unregister_handler() . . . . .	192
5.12.2.5 random_register_seed_handler() . . . . .	193
5.12.2.6 random_unregister_seed_handler() . . . . .	193
5.12.2.7 random_initialize_seed() . . . . .	193
5.12.2.8 sample_initialise_random_seed() . . . . .	194
5.12.2.9 get_random_sequence() . . . . .	194
5.12.2.10 strdup() . . . . .	194
5.12.2.11 soft_crc32() . . . . .	195
5.12.2.12 fill_sequential_pattern() . . . . .	195
5.12.2.13 verify_sequential_pattern() . . . . .	196
5.12.3 Macro Documentation . . . . .	196
5.12.3.1 dump_hex . . . . .	196
5.12.3.2 dump_hex_ascii . . . . .	197
5.12.3.3 dump_ascii . . . . .	197
5.12.3.4 print_ascii . . . . .	197
5.12.3.5 dump_json . . . . .	197
5.12.4 Typedef Documentation . . . . .	197
5.12.4.1 random_hdlr_t . . . . .	197

# Chapter 1

## Main Page

### 1.1 Introduction

NXP's WiFi functionality enables customers to quickly develop applications of interest to add connectivity to different sensors and appliances.

#### 1.1.1 Developer Documentation

This manual provides developer reference documentation for WiFi driver and WLAN Connection Manager.

In addition to the reference documentation in this manual, you can also explore the source code.

#### Note

The File Documentation provides documentation for all the APIs that are available in WiFi driver and connection manager.

Confidential

## Chapter 2

# Data Structure Index

### 2.1 Data Structures

Here are the data structures with brief descriptions:

cli_command	7
ipv4_config	8
os_queue_pool_t	9
os_thread_stack_t	10
wifi_antcfg_t	10
wifi_auto_reconnect_config_t	11
wifi_bandcfg_t	12
wifi_cal_data_t	13
wifi_chan_info_t	13
wifi_chan_list_param_set_t	14
wifi_chan_scan_param_set_t	15
wifi_chanlist_t	16
wifi_channel_desc_t	17
wifi_cw_mode_ctrl_t	18
wifi_data_rate_t	19
wifi_domain_param_t	21
wifi_ds_rate	22
wifi_ed_mac_ctrl_t	23
wififlt_cfg_t	24
wifi_fw_version_ext_t	24
wifi_fw_version_t	25
wifi_mac_addr_t	26
wifi_mef_entry_t	26
wifi_mef_filter_t	27
wifi_mgmt_frame_t	29
wifi_nat_keep_alive_t	31
wifi_rate_cfg_t	32
wifi_remain_on_channel_t	33
wifi_rf_channel_t	34
wifi_rssi_info_t	35
wifi_scan_chan_list_t	37
wifi_scan_channel_list_t	38
wifi_scan_params_v2_t	38
wifi_scan_result	40
wifi_sta_info_t	44

wifi_sta_list_t	45
wifi_sub_band_set_t	45
wifi_tbt_offset_t	46
wifi_tcp_keep_alive_t	47
wifi_tx_power_t	49
wifi_txpwrlimit_config_t	50
wifi_txpwrlimit_entry_t	50
wifi_txpwrlimit_t	51
wlan_cipher	52
wlan_ip_config	53
wlan_network	54
wlan_network_security	57
wlan_scan_result	60



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

<a href="#">cli.h</a>	CLI module . . . . .	63
<a href="#">dhcp-server.h</a>	DHCP server . . . . .	65
<a href="#">iperf.h</a>	This file provides the support for network utility iperf . . . . .	69
<a href="#">wifi-decl.h</a>	Wifi structure declarations . . . . .	70
<a href="#">wifi.h</a>	This file contains interface to wifi driver . . . . .	71
<a href="#">wifi_events.h</a>	Wi-Fi events . . . . .	84
<a href="#">wlan.h</a>	WLAN Connection Manager . . . . .	86
<a href="#">wlan_11d.h</a>	WLAN module 11d API . . . . .	149
<a href="#">wlan_tests.h</a>	WLAN Connection Manager Tests . . . . .	155
<a href="#">wm_net.h</a>	Network Abstraction Layer . . . . .	156
<a href="#">wm_os.h</a>	OS Abstraction Layer . . . . .	164
<a href="#">wm_utils.h</a>	Utility functions . . . . .	190

Confidential

## Chapter 4

# Data Structure Documentation

### 4.1 cli\_command Struct Reference

#### Data Fields

- const char \* [name](#)
- const char \* [help](#)
- void(\* [function](#) )(int argc, char \*\*argv)

#### 4.1.1 Detailed Description

Structure for registering CLI commands

#### 4.1.2 Field Documentation

##### 4.1.2.1 name

```
const char* cli_command::name
```

The name of the CLI command

##### 4.1.2.2 help

```
const char* cli_command::help
```

The help text associated with the command

#### 4.1.2.3 function

```
void(* cli_command::function) (int argc, char **argv)
```

The function that should be invoked for this command.

The documentation for this struct was generated from the following file:

- [cli.h](#)

## 4.2 ipv4\_config Struct Reference

### Data Fields

- enum [address\\_types](#) [addr\\_type](#)
- unsigned [address](#)
- unsigned [gw](#)
- unsigned [netmask](#)
- unsigned [dns1](#)
- unsigned [dns2](#)

### 4.2.1 Detailed Description

This data structure represents an IPv4 address

### 4.2.2 Field Documentation

#### 4.2.2.1 addr\_type

```
enum address\_types ipv4_config::addr_type
```

Set to [ADDR\\_TYPE\\_DHCP](#) to use DHCP to obtain the IP address or [ADDR\\_TYPE\\_STATIC](#) to use a static IP. In case of static IP address ip, gw, netmask and dns members must be specified. When using DHCP, the ip, gw, netmask and dns are overwritten by the values obtained from the DHCP server. They should be zeroed out if not used.

#### 4.2.2.2 address

```
unsigned ipv4_config::address
```

The system's IP address in network order.

#### 4.2.2.3 gw

```
unsigned ipv4_config::gw
```

The system's default gateway in network order.

#### 4.2.2.4 netmask

```
unsigned ipv4_config::netmask
```

The system's subnet mask in network order.

#### 4.2.2.5 dns1

```
unsigned ipv4_config::dns1
```

The system's primary dns server in network order.

#### 4.2.2.6 dns2

```
unsigned ipv4_config::dns2
```

The system's secondary dns server in network order.

The documentation for this struct was generated from the following file:

- [wlan.h](#)

## 4.3 os\_queue\_pool\_t Struct Reference

### Data Fields

- int [size](#)

#### 4.3.1 Detailed Description

Structure used for queue definition

#### 4.3.2 Field Documentation

#### 4.3.2.1 size

```
int os_queue_pool_t::size
```

Size of the queue

The documentation for this struct was generated from the following file:

- [wm\\_os.h](#)

## 4.4 os\_thread\_stack\_t Struct Reference

### Data Fields

- [size\\_t](#) [size](#)

#### 4.4.1 Detailed Description

Structure to be used during call to the function [os\\_thread\\_create\(\)](#). Please use the macro [os\\_thread\\_stack\\_define](#) instead of using this structure directly.

#### 4.4.2 Field Documentation

##### 4.4.2.1 size

```
size_t os_thread_stack_t::size
```

Total stack size

The documentation for this struct was generated from the following file:

- [wm\\_os.h](#)

## 4.5 wifi\_antcfg\_t Struct Reference

### Data Fields

- [t\\_u32](#) [ant\\_mode](#)
- [t\\_u16](#) [evaluate\\_time](#)

### 4.5.1 Detailed Description

Type definition of [wifi\\_antcfg\\_t](#)

### 4.5.2 Field Documentation

#### 4.5.2.1 ant\_mode

```
t_u32 wifi_antcfg_t::ant_mode
```

Antenna Mode

#### 4.5.2.2 evaluate\_time

```
t_u16 wifi_antcfg_t::evaluate_time
```

Evaluate Time

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.6 wifi\_auto\_reconnect\_config\_t Struct Reference

### Data Fields

- [t\\_u8 reconnect\\_counter](#)
- [t\\_u8 reconnect\\_interval](#)
- [t\\_u16 flags](#)

### 4.6.1 Detailed Description

Auto reconnect structure

### 4.6.2 Field Documentation

#### 4.6.2.1 reconnect\_counter

```
t_u8 wifi_auto_reconnect_config_t::reconnect_counter
```

Reconnect counter

#### 4.6.2.2 reconnect\_interval

t\_u8 wifi\_auto\_reconnect\_config\_t::reconnect\_interval

Reconnect interval

#### 4.6.2.3 flags

t\_u16 wifi\_auto\_reconnect\_config\_t::flags

Flags

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.7 wifi\_bandcfg\_t Struct Reference

### Data Fields

- t\_u16 [config\\_bands](#)
- t\_u16 [fw\\_bands](#)

#### 4.7.1 Detailed Description

Type definition of [wifi\\_bandcfg\\_t](#)

#### 4.7.2 Field Documentation

##### 4.7.2.1 config\_bands

t\_u16 wifi\_bandcfg\_t::config\_bands

Infra band

##### 4.7.2.2 fw\_bands

t\_u16 wifi\_bandcfg\_t::fw\_bands

fw supported band

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)



## 4.8 wifi\_cal\_data\_t Struct Reference

### Data Fields

- `t_u16` [data\\_len](#)
- `t_u8 *` [data](#)

### 4.8.1 Detailed Description

Calibration Data

### 4.8.2 Field Documentation

#### 4.8.2.1 data\_len

```
t_u16 wifi_cal_data_t::data_len
```

Calibration data length

#### 4.8.2.2 data

```
t_u8* wifi_cal_data_t::data
```

Calibration data

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.9 wifi\_chan\_info\_t Struct Reference

### Data Fields

- `t_u8` [chan\\_num](#)
- `t_u16` [chan\\_freq](#)
- `bool` [passive\\_scan\\_or\\_radar\\_detect](#)

### 4.9.1 Detailed Description

Data structure for Channel attributes

## 4.9.2 Field Documentation

### 4.9.2.1 chan\_num

```
t_u8 wifi_chan_info_t::chan_num
```

Channel Number

### 4.9.2.2 chan\_freq

```
t_u16 wifi_chan_info_t::chan_freq
```

Channel frequency for this channel

### 4.9.2.3 passive\_scan\_or\_radar\_detect

```
bool wifi_chan_info_t::passive_scan_or_radar_detect
```

Passive Scan or RADAR Detect

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.10 wifi\_chan\_list\_param\_set\_t Struct Reference

### Data Fields

- [t\\_u8 no\\_of\\_channels](#)
- [wifi\\_chan\\_scan\\_param\\_set\\_t chan\\_scan\\_param](#) [1]

### 4.10.1 Detailed Description

Channel list parameter set

### 4.10.2 Field Documentation

#### 4.10.2.1 no\_of\_channels

`t_u8 wifi_chan_list_param_set_t::no_of_channels`

number of channels

#### 4.10.2.2 chan\_scan\_param

`wifi_chan_scan_param_set_t wifi_chan_list_param_set_t::chan_scan_param[1]`

channel scan array

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.11 wifi\_chan\_scan\_param\_set\_t Struct Reference

### Data Fields

- `t_u8` [chan\\_number](#)
- `t_u16` [min\\_scan\\_time](#)
- `t_u16` [max\\_scan\\_time](#)

#### 4.11.1 Detailed Description

Channel scan parameters

#### 4.11.2 Field Documentation

##### 4.11.2.1 chan\_number

`t_u8 wifi_chan_scan_param_set_t::chan_number`

channel number

##### 4.11.2.2 min\_scan\_time

`t_u16 wifi_chan_scan_param_set_t::min_scan_time`

minimum scan time

#### 4.11.2.3 max\_scan\_time

```
t_u16 wifi_chan_scan_param_set_t::max_scan_time
```

maximum scan time

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.12 wifi\_chanlist\_t Struct Reference

### Data Fields

- [t\\_u8 num\\_chans](#)
- [wifi\\_chan\\_info\\_t chan\\_info](#) [54]

#### 4.12.1 Detailed Description

Data structure for Channel List Config

#### 4.12.2 Field Documentation

##### 4.12.2.1 num\_chans

```
t_u8 wifi_chanlist_t::num_chans
```

Number of Channels

##### 4.12.2.2 chan\_info

```
wifi_chan_info_t wifi_chanlist_t::chan_info[54]
```

Channel Info

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.13 wifi\_channel\_desc\_t Struct Reference

### Data Fields

- `t_u16 start_freq`
- `t_u8 chan_width`
- `t_u8 chan_num`

#### 4.13.1 Detailed Description

Data structure for Channel descriptor

Set CFG data for Tx power limitation

`start_freq`: Starting Frequency of the band for this channel  
2407, 2414 or 2400 for 2.4 GHz

5000

4000

`chan_width`: Channel Width

20

`chan_num` : Channel Number

#### 4.13.2 Field Documentation

##### 4.13.2.1 start\_freq

`t_u16 wifi_channel_desc_t::start_freq`

Starting frequency of the band for this channel

##### 4.13.2.2 chan\_width

`t_u8 wifi_channel_desc_t::chan_width`

Channel width

##### 4.13.2.3 chan\_num

`t_u8 wifi_channel_desc_t::chan_num`

Channel Number

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.14 wifi\_cw\_mode\_ctrl\_t Struct Reference

### Data Fields

- t\_u8 mode
- t\_u8 channel
- t\_u8 chanInfo
- t\_u16 txPower
- t\_u16 pktLength
- t\_u32 rateInfo

### 4.14.1 Detailed Description

CW\_MODE\_CTRL structure

### 4.14.2 Field Documentation

#### 4.14.2.1 mode

t\_u8 wifi\_cw\_mode\_ctrl\_t::mode

Mode of Operation 0:Disable 1: Tx Continuous Packet 2 : Tx Continuous Wave

#### 4.14.2.2 channel

t\_u8 wifi\_cw\_mode\_ctrl\_t::channel

channel

#### 4.14.2.3 chanInfo

t\_u8 wifi\_cw\_mode\_ctrl\_t::chanInfo

channel info

#### 4.14.2.4 txPower

t\_u16 wifi\_cw\_mode\_ctrl\_t::txPower

Tx Power level in dBm

#### 4.14.2.5 pktLength

t\_u16 wifi\_cw\_mode\_ctrl\_t::pktLength

Packet Length

#### 4.14.2.6 rateInfo

t\_u32 wifi\_cw\_mode\_ctrl\_t::rateInfo

bit rate info

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.15 wifi\_data\_rate\_t Struct Reference

### Data Fields

- t\_u32 [tx\\_data\\_rate](#)
- t\_u32 [rx\\_data\\_rate](#)
- t\_u32 [tx\\_ht\\_bw](#)
- t\_u32 [tx\\_ht\\_gi](#)
- t\_u32 [rx\\_ht\\_bw](#)
- t\_u32 [rx\\_ht\\_gi](#)
- t\_u32 [tx\\_mcs\\_index](#)
- t\_u32 [rx\\_mcs\\_index](#)
- mlan\_rate\_format [tx\\_rate\\_format](#)
- mlan\_rate\_format [rx\\_rate\\_format](#)

### 4.15.1 Detailed Description

Data structure for cmd get data rate

### 4.15.2 Field Documentation

#### 4.15.2.1 tx\_data\_rate

t\_u32 wifi\_data\_rate\_t::tx\_data\_rate

Tx data rate

#### 4.15.2.2 rx\_data\_rate

t\_u32 wifi\_data\_rate\_t::rx\_data\_rate

Rx data rate

#### 4.15.2.3 tx\_ht\_bw

t\_u32 wifi\_data\_rate\_t::tx\_ht\_bw

Tx channel bandwidth

#### 4.15.2.4 tx\_ht\_gi

t\_u32 wifi\_data\_rate\_t::tx\_ht\_gi

Tx guard interval

#### 4.15.2.5 rx\_ht\_bw

t\_u32 wifi\_data\_rate\_t::rx\_ht\_bw

Rx channel bandwidth

#### 4.15.2.6 rx\_ht\_gi

t\_u32 wifi\_data\_rate\_t::rx\_ht\_gi

Rx guard interval

#### 4.15.2.7 tx\_mcs\_index

t\_u32 wifi\_data\_rate\_t::tx\_mcs\_index

MCS index

#### 4.15.2.8 rx\_mcs\_index

t\_u32 wifi\_data\_rate\_t::rx\_mcs\_index

MCS index

#### 4.15.2.9 tx\_rate\_format

mlan\_rate\_format wifi\_data\_rate\_t::tx\_rate\_format

LG rate: 0, HT rate: 1, VHT rate: 2



## 4.15.2.10 rx\_rate\_format

```
mlan_rate_format wifi_data_rate_t::rx_rate_format
```

LG rate: 0, HT rate: 1, VHT rate: 2

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.16 wifi\_domain\_param\_t Struct Reference

### Data Fields

- [t\\_u8 country\\_code](#) [COUNTRY\_CODE\_LEN]
- [t\\_u8 no\\_of\\_sub\\_band](#)
- [wifi\\_sub\\_band\\_set\\_t sub\\_band](#) [1]

### 4.16.1 Detailed Description

Data structure for domain parameters

This structure is accepted by wlan\_uap\_set\_domain\_params() API. This information is used to generate the country info IE.

### 4.16.2 Field Documentation

#### 4.16.2.1 country\_code

```
t_u8 wifi_domain_param_t::country_code[COUNTRY_CODE_LEN]
```

Country code

#### 4.16.2.2 no\_of\_sub\_band

```
t_u8 wifi_domain_param_t::no_of_sub_band
```

subbands count

#### 4.16.2.3 sub\_band

`wifi_sub_band_set_t` `wifi_domain_param_t::sub_band[1]`

Set of subbands of `no_of_sub_band` number of elements

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.17 wifi\_ds\_rate Struct Reference

#### Data Fields

- enum `wifi_ds_command_type` [sub\\_command](#)
- union {
  - [wifi\\_rate\\_cfg\\_t](#) `rate_cfg`
  - [wifi\\_data\\_rate\\_t](#) `data_rate`
- } `param`

#### 4.17.1 Detailed Description

Type definition of [wifi\\_ds\\_rate](#)

#### 4.17.2 Field Documentation

##### 4.17.2.1 sub\_command

`enum wifi_ds_command_type` `wifi_ds_rate::sub_command`

Sub-command

##### 4.17.2.2 rate\_cfg

`wifi_rate_cfg_t` `wifi_ds_rate::rate_cfg`

Rate configuration for MLAN\_OID\_RATE\_CFG

##### 4.17.2.3 data\_rate

`wifi_data_rate_t` `wifi_ds_rate::data_rate`

Data rate for MLAN\_OID\_GET\_DATA\_RATE

#### 4.17.2.4 param

```
union { ... } wifi_ds_rate::param
```

Rate configuration parameter

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.18 wifi\_ed\_mac\_ctrl\_t Struct Reference

### Data Fields

- [t\\_u16 ed\\_ctrl\\_2g](#)
- [t\\_s16 ed\\_offset\\_2g](#)

### 4.18.1 Detailed Description

Type definition of [wifi\\_ed\\_mac\\_ctrl\\_t](#)

### 4.18.2 Field Documentation

#### 4.18.2.1 ed\_ctrl\_2g

```
t_u16 wifi_ed_mac_ctrl_t::ed_ctrl_2g
```

ED CTRL 2G

#### 4.18.2.2 ed\_offset\_2g

```
t_s16 wifi_ed_mac_ctrl_t::ed_offset_2g
```

ED Offset 2G

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.19 wififlt\_cfg\_t Struct Reference

### Data Fields

- [t\\_u32 criteria](#)
- [t\\_u16 nentries](#)
- [wifi\\_mef\\_entry\\_t mef\\_entry](#)

### 4.19.1 Detailed Description

Wifi filter config struct

### 4.19.2 Field Documentation

#### 4.19.2.1 criteria

`t_u32 wififlt_cfg_t::criteria`

Filter Criteria

#### 4.19.2.2 nentries

`t_u16 wififlt_cfg_t::nentries`

Number of entries

#### 4.19.2.3 mef\_entry

`wifi_mef_entry_t wififlt_cfg_t::mef_entry`

MEF entry

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.20 wifi\_fw\_version\_ext\_t Struct Reference

### Data Fields

- `uint8_t` [version\\_str\\_sel](#)
- `char` [version\\_str](#) [[MLAN\\_MAX\\_VER\\_STR\\_LEN](#)]

### 4.20.1 Detailed Description

Extended Firmware version

### 4.20.2 Field Documentation

#### 4.20.2.1 version\_str\_sel

```
uint8_t wifi_fw_version_ext_t::version_str_sel
```

ID for extended version select

#### 4.20.2.2 version\_str

```
char wifi_fw_version_ext_t::version_str[MLAN_MAX_VER_STR_LEN]
```

Firmware version string

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.21 wifi\_fw\_version\_t Struct Reference

### Data Fields

- char [version\\_str](#) [MLAN\_MAX\_VER\_STR\_LEN]

### 4.21.1 Detailed Description

Firmware version

### 4.21.2 Field Documentation

#### 4.21.2.1 version\_str

```
char wifi_fw_version_t::version_str[MLAN_MAX_VER_STR_LEN]
```

Firmware version string

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.22 wifi\_mac\_addr\_t Struct Reference

### Data Fields

- char [mac](#) [MLAN\_MAC\_ADDR\_LENGTH]

### 4.22.1 Detailed Description

MAC address

### 4.22.2 Field Documentation

#### 4.22.2.1 mac

```
char wifi_mac_addr_t::mac[MLAN_MAC_ADDR_LENGTH]
```

Mac address array

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.23 wifi\_mef\_entry\_t Struct Reference

### Data Fields

- t\_u8 [mode](#)
- t\_u8 [action](#)
- t\_u8 [filter\\_num](#)
- [wifi\\_mef\\_filter\\_t filter\\_item](#) [MAX\_NUM\_FILTERS]
- t\_u8 [rpn](#) [MAX\_NUM\_FILTERS]

### 4.23.1 Detailed Description

MEF entry struct

### 4.23.2 Field Documentation

#### 4.23.2.1 mode

t\_u8 wifi\_mef\_entry\_t::mode

mode: bit0—hosts sleep mode; bit1—non hosts sleep mode

#### 4.23.2.2 action

t\_u8 wifi\_mef\_entry\_t::action

action: 0—discard and not wake host; 1—discard and wake host; 3—allow and wake host;

#### 4.23.2.3 filter\_num

t\_u8 wifi\_mef\_entry\_t::filter\_num

filter number

#### 4.23.2.4 filter\_item

wifi\_mef\_filter\_t wifi\_mef\_entry\_t::filter\_item[MAX\_NUM\_FILTERS]

filter array

#### 4.23.2.5 rpn

t\_u8 wifi\_mef\_entry\_t::rpn[MAX\_NUM\_FILTERS]

rpn array

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.24 wifi\_mef\_filter\_t Struct Reference

### Data Fields

- t\_u16 [type](#)
- t\_u32 [pattern](#)
- t\_u16 [offset](#)
- t\_u16 [num\\_bytes](#)
- t\_u16 [repeat](#)
- t\_u8 [num\\_byte\\_seq](#)
- t\_u8 [byte\\_seq](#) [MAX\_NUM\_BYTE\_SEQ]
- t\_u8 [num\\_mask\\_seq](#)
- t\_u8 [mask\\_seq](#) [MAX\_NUM\_MASK\_SEQ]

#### 4.24.1 Detailed Description

Type definition of filter\_item support three match methods: <1>Byte comparison type=0x41 <2>Decimal comparison type=0x42 <3>Bit comparison type=0x43

#### 4.24.2 Field Documentation

##### 4.24.2.1 type

```
t_u16 wifi_mef_filter_t::type
```

BYTE 0X41; Decimal 0X42; Bit 0x43

##### 4.24.2.2 pattern

```
t_u32 wifi_mef_filter_t::pattern
```

value

##### 4.24.2.3 offset

```
t_u16 wifi_mef_filter_t::offset
```

offset

##### 4.24.2.4 num\_bytes

```
t_u16 wifi_mef_filter_t::num_bytes
```

number of bytes

##### 4.24.2.5 repeat

```
t_u16 wifi_mef_filter_t::repeat
```

repeat

##### 4.24.2.6 num\_byte\_seq

```
t_u8 wifi_mef_filter_t::num_byte_seq
```

byte number



#### 4.24.2.7 byte\_seq

```
t_u8 wifi_mef_filter_t::byte_seq[MAX_NUM_BYTE_SEQ]
```

array

#### 4.24.2.8 num\_mask\_seq

```
t_u8 wifi_mef_filter_t::num_mask_seq
```

mask numbers

#### 4.24.2.9 mask\_seq

```
t_u8 wifi_mef_filter_t::mask_seq[MAX_NUM_MASK_SEQ]
```

array

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.25 wifi\_mgmt\_frame\_t Struct Reference

### Data Fields

- t\_u16 [frm\\_len](#)
- [wifi\\_frame\\_type\\_t](#) [frame\\_type](#)
- t\_u8 [frame\\_ctrl\\_flags](#)
- t\_u16 [duration\\_id](#)
- t\_u8 [addr1](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u8 [addr2](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u8 [addr3](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u16 [seq\\_ctl](#)
- t\_u8 [addr4](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u8 [payload](#) [0]

### 4.25.1 Detailed Description

802\_11\_header packet

### 4.25.2 Field Documentation

#### 4.25.2.1 frm\_len

```
t_u16 wifi_mgmt_frame_t::frm_len
```

Packet Length

#### 4.25.2.2 frame\_type

```
wifi_frame_type_t wifi_mgmt_frame_t::frame_type
```

Frame Type

#### 4.25.2.3 frame\_ctrl\_flags

```
t_u8 wifi_mgmt_frame_t::frame_ctrl_flags
```

Frame Control flags

#### 4.25.2.4 duration\_id

```
t_u16 wifi_mgmt_frame_t::duration_id
```

Duration ID

#### 4.25.2.5 addr1

```
t_u8 wifi_mgmt_frame_t::addr1[MLAN_MAC_ADDR_LENGTH]
```

Address 1

#### 4.25.2.6 addr2

```
t_u8 wifi_mgmt_frame_t::addr2[MLAN_MAC_ADDR_LENGTH]
```

Address 2

#### 4.25.2.7 addr3

```
t_u8 wifi_mgmt_frame_t::addr3[MLAN_MAC_ADDR_LENGTH]
```

Address 3

#### 4.25.2.8 seq\_ctl

```
t_u16 wifi_mgmt_frame_t::seq_ctl
```

Sequence Control

#### 4.25.2.9 addr4

```
t_u8 wifi_mgmt_frame_t::addr4[MLAN_MAC_ADDR_LENGTH]
```

Address 4

#### 4.25.2.10 payload

```
t_u8 wifi_mgmt_frame_t::payload[0]
```

Frame payload

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.26 wifi\_nat\_keep\_alive\_t Struct Reference

### Data Fields

- t\_u16 [interval](#)
- t\_u8 [dst\\_mac](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u32 [dst\\_ip](#)
- t\_u16 [dst\\_port](#)

### 4.26.1 Detailed Description

TCP nat keep alive information

### 4.26.2 Field Documentation

#### 4.26.2.1 interval

```
t_u16 wifi_nat_keep_alive_t::interval
```

Keep alive interval

#### 4.26.2.2 dst\_mac

```
t_u8 wifi_nat_keep_alive_t::dst_mac[MLAN_MAC_ADDR_LENGTH]
```

Destination MAC address

#### 4.26.2.3 dst\_ip

```
t_u32 wifi_nat_keep_alive_t::dst_ip
```

Destination IP

#### 4.26.2.4 dst\_port

```
t_u16 wifi_nat_keep_alive_t::dst_port
```

Destination port

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.27 wifi\_rate\_cfg\_t Struct Reference

#### Data Fields

- mlan\_rate\_format [rate\\_format](#)
- t\_u32 [rate\\_index](#)
- t\_u32 [rate](#)

#### 4.27.1 Detailed Description

Data structure for cmd txratecfg

#### 4.27.2 Field Documentation

##### 4.27.2.1 rate\_format

```
mlan_rate_format wifi_rate_cfg_t::rate_format
```

LG rate: 0, HT rate: 1, VHT rate: 2

##### 4.27.2.2 rate\_index

```
t_u32 wifi_rate_cfg_t::rate_index
```

Rate/MCS index (0xFF: auto)

#### 4.27.2.3 rate

```
t_u32 wifi_rate_cfg_t::rate
```

Rate rate

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.28 wifi\_remain\_on\_channel\_t Struct Reference

### Data Fields

- uint16\_t [remove](#)
- uint8\_t [status](#)
- uint8\_t [bandcfg](#)
- uint8\_t [channel](#)
- uint32\_t [remain\\_period](#)

### 4.28.1 Detailed Description

Remain on channel info structure

### 4.28.2 Field Documentation

#### 4.28.2.1 remove

```
uint16_t wifi_remain_on_channel_t::remove
```

Remove

#### 4.28.2.2 status

```
uint8_t wifi_remain_on_channel_t::status
```

Current status

#### 4.28.2.3 bandcfg

```
uint8_t wifi_remain_on_channel_t::bandcfg
```

band configuration

#### 4.28.2.4 channel

```
uint8_t wifi_remain_on_channel_t::channel
```

Channel

#### 4.28.2.5 remain\_period

```
uint32_t wifi_remain_on_channel_t::remain_period
```

Remain on channel period

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.29 wifi\_rf\_channel\_t Struct Reference

#### Data Fields

- uint16\_t [current\\_channel](#)
- uint16\_t [rf\\_type](#)

#### 4.29.1 Detailed Description

Rf channel

#### 4.29.2 Field Documentation

##### 4.29.2.1 current\_channel

```
uint16_t wifi_rf_channel_t::current_channel
```

Current channel

##### 4.29.2.2 rf\_type

```
uint16_t wifi_rf_channel_t::rf_type
```

RF Type

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.30 wifi\_rssi\_info\_t Struct Reference

### Data Fields

- [int16\\_t data\\_rssi\\_last](#)
- [int16\\_t data\\_nf\\_last](#)
- [int16\\_t data\\_rssi\\_avg](#)
- [int16\\_t data\\_nf\\_avg](#)
- [int16\\_t bcn\\_snr\\_last](#)
- [int16\\_t bcn\\_snr\\_avg](#)
- [int16\\_t data\\_snr\\_last](#)
- [int16\\_t data\\_snr\\_avg](#)
- [int16\\_t bcn\\_rssi\\_last](#)
- [int16\\_t bcn\\_nf\\_last](#)
- [int16\\_t bcn\\_rssi\\_avg](#)
- [int16\\_t bcn\\_nf\\_avg](#)

### 4.30.1 Detailed Description

RSSI information

### 4.30.2 Field Documentation

#### 4.30.2.1 data\_rssi\_last

```
int16_t wifi_rssi_info_t::data_rssi_last
```

Data RSSI last

#### 4.30.2.2 data\_nf\_last

```
int16_t wifi_rssi_info_t::data_nf_last
```

Data nf last

#### 4.30.2.3 data\_rssi\_avg

```
int16_t wifi_rssi_info_t::data_rssi_avg
```

Data RSSI average

#### 4.30.2.4 data\_nf\_avg

```
int16_t wifi_rssi_info_t::data_nf_avg
```

Data nf average

#### 4.30.2.5 bcn\_snr\_last

```
int16_t wifi_rssi_info_t::bcn_snr_last
```

BCN SNR

#### 4.30.2.6 bcn\_snr\_avg

```
int16_t wifi_rssi_info_t::bcn_snr_avg
```

BCN SNR average

#### 4.30.2.7 data\_snr\_last

```
int16_t wifi_rssi_info_t::data_snr_last
```

Data SNR last

#### 4.30.2.8 data\_snr\_avg

```
int16_t wifi_rssi_info_t::data_snr_avg
```

Data SNR average

#### 4.30.2.9 bcn\_rssi\_last

```
int16_t wifi_rssi_info_t::bcn_rssi_last
```

BCN RSSI

#### 4.30.2.10 bcn\_nf\_last

```
int16_t wifi_rssi_info_t::bcn_nf_last
```

BCN nf

#### 4.30.2.11 bcn\_rssi\_avg

```
int16_t wifi_rssi_info_t::bcn_rssi_avg
```

BCN RSSI average



## 4.30.2.12 bcn\_nf\_avg

```
int16_t wifi_rssi_info_t::bcn_nf_avg
```

BCN nf average

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.31 wifi\_scan\_chan\_list\_t Struct Reference

## Data Fields

- uint8\_t [num\\_of\\_chan](#)
- uint8\_t [chan\\_number](#) [MLAN\_MAX\_CHANNEL]

## 4.31.1 Detailed Description

Channel list structure

## 4.31.2 Field Documentation

## 4.31.2.1 num\_of\_chan

```
uint8_t wifi_scan_chan_list_t::num_of_chan
```

Number of channels

## 4.31.2.2 chan\_number

```
uint8_t wifi_scan_chan_list_t::chan_number [MLAN_MAX_CHANNEL]
```

Channel number

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.32 wifi\_scan\_channel\_list\_t Struct Reference

### Data Fields

- t\_u8 [chan\\_number](#)
- mlan\_scan\_type [scan\\_type](#)
- t\_u16 [scan\\_time](#)

### 4.32.1 Detailed Description

Scan channel list

### 4.32.2 Field Documentation

#### 4.32.2.1 chan\_number

```
t_u8 wifi_scan_channel_list_t::chan_number
```

Channel number

#### 4.32.2.2 scan\_type

```
mlan_scan_type wifi_scan_channel_list_t::scan_type
```

Scan type Active = 1, Passive = 2

#### 4.32.2.3 scan\_time

```
t_u16 wifi_scan_channel_list_t::scan_time
```

Scan time

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.33 wifi\_scan\_params\_v2\_t Struct Reference

### Data Fields

- t\_u8 [bssid](#) [MLAN\_MAC\_ADDR\_LENGTH]
- char [ssid](#) [MLAN\_MAX\_SSID\_LENGTH+1]
- t\_u8 [num\\_channels](#)
- [wifi\\_scan\\_channel\\_list\\_t](#) [chan\\_list](#) [MAX\_CHANNEL\_LIST]
- t\_u8 [num\\_probes](#)
- int(\* [cb](#) )(unsigned int count)

### 4.33.1 Detailed Description

V2 scan parameters

### 4.33.2 Field Documentation

#### 4.33.2.1 bssid

```
t_u8 wifi_scan_params_v2_t::bssid[MLAN_MAC_ADDR_LENGTH]
```

BSSID to scan

#### 4.33.2.2 ssid

```
char wifi_scan_params_v2_t::ssid[MLAN_MAX_SSID_LENGTH+1]
```

SSID to scan

#### 4.33.2.3 num\_channels

```
t_u8 wifi_scan_params_v2_t::num_channels
```

Number of channels

#### 4.33.2.4 chan\_list

```
wifi_scan_channel_list_t wifi_scan_params_v2_t::chan_list[MAX_CHANNEL_LIST]
```

Channel list with channel information

#### 4.33.2.5 num\_probes

```
t_u8 wifi_scan_params_v2_t::num_probes
```

Number of probes

#### 4.33.2.6 cb

```
int(* wifi_scan_params_v2_t::cb) (unsigned int count)
```

Callback to be called when scan is completed

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.34 wifi\_scan\_result Struct Reference

### Data Fields

- uint8\_t [bssid](#) [MLAN\_MAC\_ADDR\_LENGTH]
- bool [is\\_ibss\\_bit\\_set](#)
- uint8\_t [ssid](#) [MLAN\_MAX\_SSID\_LENGTH]
- int [ssid\\_len](#)
- uint8\_t [Channel](#)
- uint8\_t [RSSI](#)
- uint16\_t [beacon\\_period](#)
- uint8\_t [dtim\\_period](#)
- \_SecurityMode\_t [WPA\\_WPA2\\_WEP](#)
- \_Cipher\_t [wpa\\_mcstCipher](#)
- \_Cipher\_t [wpa\\_ucstCipher](#)
- \_Cipher\_t [rsn\\_mcstCipher](#)
- \_Cipher\_t [rsn\\_ucstCipher](#)
- bool [is\\_pmf\\_required](#)
- bool [phtcap\\_ie\\_present](#)
- bool [phtinfo\\_ie\\_present](#)
- bool [wmm\\_ie\\_present](#)
- uint16\_t [band](#)
- bool [wps\\_IE\\_exist](#)
- uint16\_t [wps\\_session](#)
- bool [wpa2\\_entp\\_IE\\_exist](#)
- uint8\_t [trans\\_mode](#)
- uint8\_t [trans\\_bssid](#) [MLAN\_MAC\_ADDR\_LENGTH]
- uint8\_t [trans\\_ssid](#) [MLAN\_MAX\_SSID\_LENGTH]
- int [trans\\_ssid\\_len](#)

### 4.34.1 Detailed Description

Scan result information

### 4.34.2 Field Documentation

#### 4.34.2.1 bssid

```
uint8_t wifi_scan_result::bssid[MLAN_MAC_ADDR_LENGTH]
```

BSSID array

#### 4.34.2.2 is\_ibss\_bit\_set

```
bool wifi_scan_result::is_ibss_bit_set
```

Is bssid set?

#### 4.34.2.3 ssid

```
uint8_t wifi_scan_result::ssid[MLAN_MAX_SSID_LENGTH]
```

ssid array

#### 4.34.2.4 ssid\_len

```
int wifi_scan_result::ssid_len
```

SSID length

#### 4.34.2.5 Channel

```
uint8_t wifi_scan_result::Channel
```

Channel associated to the BSSID

#### 4.34.2.6 RSSI

```
uint8_t wifi_scan_result::RSSI
```

Received signal strength

#### 4.34.2.7 beacon\_period

```
uint16_t wifi_scan_result::beacon_period
```

Beacon period

#### 4.34.2.8 dtim\_period

```
uint8_t wifi_scan_result::dtim_period
```

DTIM period

#### 4.34.2.9 WPA\_WPA2\_WEP

```
_SecurityMode_t wifi_scan_result::WPA_WPA2_WEP
```

Security mode info

#### 4.34.2.10 wpa\_mcstCipher

```
_Cipher_t wifi_scan_result::wpa_mcstCipher
```

WPA multicast cipher

#### 4.34.2.11 wpa\_ucstCipher

`_Cipher_t wifi_scan_result::wpa_ucstCipher`

WPA unicast cipher

#### 4.34.2.12 rsn\_mcstCipher

`_Cipher_t wifi_scan_result::rsn_mcstCipher`

No security multicast cipher

#### 4.34.2.13 rsn\_ucstCipher

`_Cipher_t wifi_scan_result::rsn_ucstCipher`

No security unicast cipher

#### 4.34.2.14 is\_pmf\_required

`bool wifi_scan_result::is_pmf_required`

Is pmf required flag WPA\_WPA2 = 0 => Security not enabled = 1 => WPA mode = 2 => WPA2 mode = 3 => WEP mode

#### 4.34.2.15 phtcap\_ie\_present

`bool wifi_scan_result::phtcap_ie_present`

PHT CAP IE present info

#### 4.34.2.16 phtinfo\_ie\_present

`bool wifi_scan_result::phtinfo_ie_present`

PHT INFO IE present info

#### 4.34.2.17 wmm\_ie\_present

`bool wifi_scan_result::wmm_ie_present`

WMM IE present info

#### 4.34.2.18 band

```
uint16_t wifi_scan_result::band
```

Band info

#### 4.34.2.19 wps\_ie\_exist

```
bool wifi_scan_result::wps_ie_exist
```

WPS IE exist info

#### 4.34.2.20 wps\_session

```
uint16_t wifi_scan_result::wps_session
```

WPS session

#### 4.34.2.21 wpa2\_entp\_ie\_exist

```
bool wifi_scan_result::wpa2_entp_ie_exist
```

WPA2 enterprise IE exist info

#### 4.34.2.22 trans\_mode

```
uint8_t wifi_scan_result::trans_mode
```

Trans mode

#### 4.34.2.23 trans\_bssid

```
uint8_t wifi_scan_result::trans_bssid[MLAN_MAC_ADDR_LENGTH]
```

Trans bssid array

#### 4.34.2.24 trans\_ssid

```
uint8_t wifi_scan_result::trans_ssid[MLAN_MAX_SSID_LENGTH]
```

Trans ssid array

#### 4.34.2.25 trans\_ssid\_len

```
int wifi_scan_result::trans_ssid_len
```

Trans bssid length

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.35 wifi\_sta\_info\_t Struct Reference

#### Data Fields

- `t_u8 mac` [MLAN\_MAC\_ADDR\_LENGTH]
- `t_u8 power_mgmt_status`
- `t_s8 rssi`

#### 4.35.1 Detailed Description

Station information structure

#### 4.35.2 Field Documentation

##### 4.35.2.1 mac

```
t_u8 wifi_sta_info_t::mac[MLAN_MAC_ADDR_LENGTH]
```

MAC address buffer

##### 4.35.2.2 power\_mgmt\_status

```
t_u8 wifi_sta_info_t::power_mgmt_status
```

Power management status 0 = active (not in power save) 1 = in power save status

##### 4.35.2.3 rssi

```
t_s8 wifi_sta_info_t::rssi
```

RSSI: dBm

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)



## 4.36 wifi\_sta\_list\_t Struct Reference

### Data Fields

- int [count](#)

### 4.36.1 Detailed Description

Note: This is variable length structure. The size of array mac\_list is equal to count. The caller of the API which returns this structure does not need to separately free the array mac\_list. It only needs to free the sta\_list\_t object after use.

### 4.36.2 Field Documentation

#### 4.36.2.1 count

```
int wifi_sta_list_t::count
```

Count

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.37 wifi\_sub\_band\_set\_t Struct Reference

### Data Fields

- t\_u8 [first\\_chan](#)
- t\_u8 [no\\_of\\_chan](#)
- t\_u8 [max\\_tx\\_pwr](#)

### 4.37.1 Detailed Description

Data structure for subband set

For uAP 11d support

### 4.37.2 Field Documentation

#### 4.37.2.1 first\_chan

`t_u8 wifi_sub_band_set_t::first_chan`

First channel

#### 4.37.2.2 no\_of\_chan

`t_u8 wifi_sub_band_set_t::no_of_chan`

Number of channels

#### 4.37.2.3 max\_tx\_pwr

`t_u8 wifi_sub_band_set_t::max_tx_pwr`

Maximum Tx power in dBm

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.38 wifi\_tbtt\_offset\_t Struct Reference

#### Data Fields

- `t_u32 min_tbtt_offset`
- `t_u32 max_tbtt_offset`
- `t_u32 avg_tbtt_offset`

#### 4.38.1 Detailed Description

TBTT offset structure

#### 4.38.2 Field Documentation

##### 4.38.2.1 min\_tbtt\_offset

`t_u32 wifi_tbtt_offset_t::min_tbtt_offset`

Min TBTT offset

#### 4.38.2.2 max\_tbtt\_offset

t\_u32 wifi\_tbtt\_offset\_t::max\_tbtt\_offset

Max TBTT offset

#### 4.38.2.3 avg\_tbtt\_offset

t\_u32 wifi\_tbtt\_offset\_t::avg\_tbtt\_offset

AVG TBTT offset

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.39 wifi\_tcp\_keep\_alive\_t Struct Reference

### Data Fields

- t\_u8 [enable](#)
- t\_u8 [reset](#)
- t\_u32 [timeout](#)
- t\_u16 [interval](#)
- t\_u16 [max\\_keep\\_alives](#)
- t\_u8 [dst\\_mac](#) [MLAN\_MAC\_ADDR\_LENGTH]
- t\_u32 [dst\\_ip](#)
- t\_u16 [dst\\_tcp\\_port](#)
- t\_u16 [src\\_tcp\\_port](#)
- t\_u32 [seq\\_no](#)

### 4.39.1 Detailed Description

TCP keep alive information

### 4.39.2 Field Documentation

#### 4.39.2.1 enable

t\_u8 wifi\_tcp\_keep\_alive\_t::enable

Enable keep alive

#### 4.39.2.2 reset

```
t_u8 wifi_tcp_keep_alive_t::reset
```

Reset

#### 4.39.2.3 timeout

```
t_u32 wifi_tcp_keep_alive_t::timeout
```

Keep alive timeout

#### 4.39.2.4 interval

```
t_u16 wifi_tcp_keep_alive_t::interval
```

Keep alive interval

#### 4.39.2.5 max\_keep\_alives

```
t_u16 wifi_tcp_keep_alive_t::max_keep_alives
```

Maximum keep alives

#### 4.39.2.6 dst\_mac

```
t_u8 wifi_tcp_keep_alive_t::dst_mac[MLAN_MAC_ADDR_LENGTH]
```

Destination MAC address

#### 4.39.2.7 dst\_ip

```
t_u32 wifi_tcp_keep_alive_t::dst_ip
```

Destination IP

#### 4.39.2.8 dst\_tcp\_port

```
t_u16 wifi_tcp_keep_alive_t::dst_tcp_port
```

Destination TCP port

#### 4.39.2.9 src\_tcp\_port

```
t_u16 wifi_tcp_keep_alive_t::src_tcp_port
```

Source TCP port

#### 4.39.2.10 seq\_no

t\_u32 wifi\_tcp\_keep\_alive\_t::seq\_no

Sequence number

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.40 wifi\_tx\_power\_t Struct Reference

### Data Fields

- uint16\_t [current\\_level](#)
- uint8\_t [max\\_power](#)
- uint8\_t [min\\_power](#)

### 4.40.1 Detailed Description

Tx power levels

### 4.40.2 Field Documentation

#### 4.40.2.1 current\_level

uint16\_t wifi\_tx\_power\_t::current\_level

Current power level

#### 4.40.2.2 max\_power

uint8\_t wifi\_tx\_power\_t::max\_power

Maximum power level

#### 4.40.2.3 min\_power

uint8\_t wifi\_tx\_power\_t::min\_power

Minimum power level

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.41 wifi\_txpwrlimit\_config\_t Struct Reference

### Data Fields

- [t\\_u8 num\\_mod\\_grps](#)
- [wifi\\_channel\\_desc\\_t chan\\_desc](#)
- [wifi\\_txpwrlimit\\_entry\\_t txpwrlimit\\_entry](#) [10]

### 4.41.1 Detailed Description

Data structure for TRPC config

For TRPC support

### 4.41.2 Field Documentation

#### 4.41.2.1 num\_mod\_grps

`t_u8 wifi_txpwrlimit_config_t::num_mod_grps`

Number of modulation groups

#### 4.41.2.2 chan\_desc

`wifi_channel_desc_t wifi_txpwrlimit_config_t::chan_desc`

Channel descriptor

#### 4.41.2.3 txpwrlimit\_entry

`wifi_txpwrlimit_entry_t wifi_txpwrlimit_config_t::txpwrlimit_entry`[10]

Channel Modulation groups

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.42 wifi\_txpwrlimit\_entry\_t Struct Reference

### Data Fields

- [t\\_u8 mod\\_group](#)
- [t\\_u8 tx\\_power](#)

#### 4.42.1 Detailed Description

Data structure for Modulation Group

mod\_group : ModulationGroup  
0: CCK (1,2,5.5,11 Mbps)  
1: OFDM (6,9,12,18 Mbps)  
2: OFDM (24,36 Mbps)  
3: OFDM (48,54 Mbps)  
4: HT20 (0,1,2)  
5: HT20 (3,4)  
6: HT20 (5,6,7)  
7: HT40 (0,1,2)  
8: HT40 (3,4)  
9: HT40 (5,6,7)  
10: HT2\_20 (8,9,10)  
11: HT2\_20 (11,12)  
12: HT2\_20 (13,14,15)  
tx\_power : Power Limit in dBm

#### 4.42.2 Field Documentation

##### 4.42.2.1 mod\_group

t\_u8 wifi\_txpwrlimit\_entry\_t::mod\_group

Modulation group

##### 4.42.2.2 tx\_power

t\_u8 wifi\_txpwrlimit\_entry\_t::tx\_power

Tx Power

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

### 4.43 wifi\_txpwrlimit\_t Struct Reference

#### Data Fields

- [wifi\\_SubBand\\_t](#) subband
- t\_u8 num\_chans
- [wifi\\_txpwrlimit\\_config\\_t](#) txpwrlimit\_config [40]

### 4.43.1 Detailed Description

Data structure for Channel TRPC config

For TRPC support

### 4.43.2 Field Documentation

#### 4.43.2.1 subband

`wifi_SubBand_t` `wifi_txpwrlimit_t::subband`

SubBand

#### 4.43.2.2 num\_chans

`t_u8` `wifi_txpwrlimit_t::num_chans`

Number of Channels

#### 4.43.2.3 txpwrlimit\_config

`wifi_txpwrlimit_config_t` `wifi_txpwrlimit_t::txpwrlimit_config[40]`

TRPC config

The documentation for this struct was generated from the following file:

- [wifi-decl.h](#)

## 4.44 wlan\_cipher Struct Reference

### Data Fields

- `uint8_t` `wep40`: 1
- `uint8_t` `wep104`: 1
- `uint8_t` `tkip`: 1
- `uint8_t` `ccmp`: 1
- `uint8_t` `rsvd`: 4

### 4.44.1 Detailed Description

Wlan Cipher structure



#### 4.44.2 Field Documentation

##### 4.44.2.1 wep40

```
uint8_t wlan_cipher::wep40
```

1 bit value can be set for wep40

##### 4.44.2.2 wep104

```
uint8_t wlan_cipher::wep104
```

1 bit value can be set for wep104

##### 4.44.2.3 tkip

```
uint8_t wlan_cipher::tkip
```

1 bit value can be set for tkip

##### 4.44.2.4 ccmp

```
uint8_t wlan_cipher::ccmp
```

1 bit value can be set for ccmp

##### 4.44.2.5 rsvd

```
uint8_t wlan_cipher::rsvd
```

4 bits are reserved

The documentation for this struct was generated from the following file:

- [wlan.h](#)

## 4.45 wlan\_ip\_config Struct Reference

### Data Fields

- struct [ipv4\\_config](#) ipv4

### 4.45.1 Detailed Description

Network IP configuration.

This data structure represents the network IP configuration for IPv4 as well as IPv6 addresses

### 4.45.2 Field Documentation

#### 4.45.2.1 ipv4

```
struct ipv4_config wlan_ip_config::ipv4
```

The network IPv4 address configuration that should be associated with this interface.

The documentation for this struct was generated from the following file:

- [wlan.h](#)

## 4.46 wlan\_network Struct Reference

### Data Fields

- char [name](#) [[WLAN\\_NETWORK\\_NAME\\_MAX\\_LENGTH](#)]
- char [ssid](#) [[IEEEtypes\\_SSID\\_SIZE](#)+1]
- char [bssid](#) [[IEEEtypes\\_ADDRESS\\_SIZE](#)]
- unsigned int [channel](#)
- enum [wlan\\_bss\\_type](#) type
- enum [wlan\\_bss\\_role](#) role
- struct [wlan\\_network\\_security](#) security
- struct [wlan\\_ip\\_config](#) ip
- unsigned [ssid\\_specific](#): 1
- unsigned [bssid\\_specific](#): 1
- unsigned [channel\\_specific](#): 1
- unsigned [security\\_specific](#): 1
- uint16\_t [beacon\\_period](#)
- uint8\_t [dtim\\_period](#)

### 4.46.1 Detailed Description

#### WLAN Network Profile

This data structure represents a WLAN network profile. It consists of an arbitrary name, WiFi configuration, and IP address configuration.

Every network profile is associated with one of the two interfaces. The network profile can be used for the station interface (i.e. to connect to an Access Point) by setting the role field to [WLAN\\_BSS\\_ROLE\\_STA](#). The network profile can be used for the micro-AP interface (i.e. to start a network of our own.) by setting the mode field to [WLAN\\_BSS\\_ROLE\\_UAP](#).

If the mode field is [WLAN\\_BSS\\_ROLE\\_STA](#), either of the SSID or BSSID fields are used to identify the network, while the other members like channel and security settings characterize the network.

If the mode field is [WLAN\\_BSS\\_ROLE\\_UAP](#), the SSID, channel and security fields are used to define the network to be started.

In both the above cases, the address field is used to determine the type of address assignment to be used for this interface.

## 4.46.2 Field Documentation

### 4.46.2.1 name

```
char wlan_network::name[WLAN_NETWORK_NAME_MAX_LENGTH]
```

The name of this network profile. Each network profile that is added to the WLAN Connection Manager must have a unique name.

### 4.46.2.2 ssid

```
char wlan_network::ssid[IEEEtypes_SSID_SIZE+1]
```

The network SSID, represented as a C string of up to 32 characters in length. If this profile is used in the micro-AP mode, this field is used as the SSID of the network. If this profile is used in the station mode, this field is used to identify the network. Set the first byte of the SSID to NULL (a 0-length string) to use only the BSSID to find the network.

### 4.46.2.3 bssid

```
char wlan_network::bssid[IEEEtypes_ADDRESS_SIZE]
```

The network BSSID, represented as a 6-byte array. If this profile is used in the micro-AP mode, this field is ignored. If this profile is used in the station mode, this field is used to identify the network. Set all 6 bytes to 0 to use any BSSID, in which case only the SSID will be used to find the network.

### 4.46.2.4 channel

```
unsigned int wlan_network::channel
```

The channel for this network.

If this profile is used in micro-AP mode, this field specifies the channel to start the micro-AP interface on. Set this to 0 for auto channel selection.

If this profile is used in the station mode, this constrains the channel on which the network to connect should be present. Set this to 0 to allow the network to be found on any channel.

### 4.46.2.5 type

```
enum wlan_bss_type wlan_network::type
```

BSS type

#### 4.46.2.6 role

```
enum wlan_bss_role wlan_network::role
```

The network wireless mode enum `wlan_bss_role`. Set this to specify what type of wireless network mode to use. This can either be `WLAN_BSS_ROLE_STA` for use in the station mode, or it can be `WLAN_BSS_ROLE_UAP` for use in the micro-AP mode.

#### 4.46.2.7 security

```
struct wlan_network_security wlan_network::security
```

The network security configuration specified by struct `wlan_network_security` for the network.

#### 4.46.2.8 ip

```
struct wlan_ip_config wlan_network::ip
```

The network IP address configuration specified by struct `wlan_ip_config` that should be associated with this interface.

#### 4.46.2.9 ssid\_specific

```
unsigned wlan_network::ssid_specific
```

If set to 1, the `ssid` field contains the specific SSID for this network. The WLAN Connection Manager will only connect to networks whose SSID matches. If set to 0, the `ssid` field contents are not used when deciding whether to connect to a network, the BSSID field is used instead and any network whose BSSID matches is accepted.

This field will be set to 1 if the network is added with the SSID specified (not an empty string), otherwise it is set to 0.

#### 4.46.2.10 bssid\_specific

```
unsigned wlan_network::bssid_specific
```

If set to 1, the `bssid` field contains the specific BSSID for this network. The WLAN Connection Manager will not connect to any other network with the same SSID unless the BSSID matches. If set to 0, the WLAN Connection Manager will connect to any network whose SSID matches.

This field will be set to 1 if the network is added with the BSSID specified (not set to all zeroes), otherwise it is set to 0.

#### 4.46.2.11 channel\_specific

```
unsigned wlan_network::channel_specific
```

If set to 1, the `channel` field contains the specific channel for this network. The WLAN Connection Manager will not look for this network on any other channel. If set to 0, the WLAN Connection Manager will look for this network on any available channel.

This field will be set to 1 if the network is added with the channel specified (not set to 0), otherwise it is set to 0.

## 4.46.2.12 security\_specific

```
unsigned wlan_network::security_specific
```

If set to 0, any security that matches is used. This field is internally set when the security type parameter above is set to WLAN\_SECURITY\_WILDCARD.

## 4.46.2.13 beacon\_period

```
uint16_t wlan_network::beacon_period
```

Beacon period of associated BSS

## 4.46.2.14 dtim\_period

```
uint8_t wlan_network::dtim_period
```

DTIM period of associated BSS

The documentation for this struct was generated from the following file:

- [wlan.h](#)

## 4.47 wlan\_network\_security Struct Reference

## Data Fields

- enum [wlan\\_security\\_type](#) type
- struct [wlan\\_cipher](#) mcstCipher
- struct [wlan\\_cipher](#) ucstCipher
- bool [is\\_pmf\\_required](#)
- char [psk](#) [WLAN\_PSK\_MAX\_LENGTH]
- uint8\_t [psk\\_len](#)
- char [password](#) [WLAN\_PASSWORD\_MAX\_LENGTH]
- size\_t [password\\_len](#)
- char [pmk](#) [WLAN\_PMK\_LENGTH]
- bool [pmk\\_valid](#)
- bool [mfpc](#)
- bool [mfpr](#)

## 4.47.1 Detailed Description

Network security configuration

## 4.47.2 Field Documentation

#### 4.47.2.1 type

```
enum wlan_security_type wlan_network_security::type
```

Type of network security to use specified by enum wlan\_security\_type.

#### 4.47.2.2 mcstCipher

```
struct wlan_cipher wlan_network_security::mcstCipher
```

Type of network security Group Cipher suite used internally

#### 4.47.2.3 ucstCipher

```
struct wlan_cipher wlan_network_security::ucstCipher
```

Type of network security Pairwise Cipher suite used internally

#### 4.47.2.4 is\_pmf\_required

```
bool wlan_network_security::is_pmf_required
```

Is PMF required

#### 4.47.2.5 psk

```
char wlan_network_security::psk[WLAN_PSK_MAX_LENGTH]
```

Pre-shared key (network password). For WEP networks this is a hex byte sequence of length psk\_len, for WPA and WPA2 networks this is an ASCII pass-phrase of length psk\_len. This field is ignored for networks with no security.

#### 4.47.2.6 psk\_len

```
uint8_t wlan_network_security::psk_len
```

Length of the WEP key or WPA/WPA2 pass phrase, [WLAN\\_PSK\\_MIN\\_LENGTH](#) to [WLAN\\_PSK\\_MAX\\_LENGTH](#). Ignored for networks with no security.

#### 4.47.2.7 password

```
char wlan_network_security::password[WLAN_PASSWORD_MAX_LENGTH]
```

WPA3 SAE password, for WPA3 SAE networks this is an ASCII password of length password\_len. This field is ignored for networks with no security.

#### 4.47.2.8 password\_len

```
size_t wlan_network_security::password_len
```

Length of the WPA3 SAE Password, WLAN\_PASSWORD\_MIN\_LENGTH to WLAN\_PASSWORD\_MAX\_LENGTH. Ignored for networks with no security.

#### 4.47.2.9 pmk

```
char wlan_network_security::pmk[WLAN_PMK_LENGTH]
```

Pairwise Master Key. When pmk\_valid is set, this is the PMK calculated from the PSK for WPA/PSK networks. If pmk\_valid is not set, this field is not valid. When adding networks with [wlan\\_add\\_network](#), users can initialize pmk and set pmk\_valid in lieu of setting the psk. After successfully connecting to a WPA/PSK network, users can call [wlan\\_get\\_current\\_network](#) to inspect pmk\_valid and pmk. Thus, the pmk value can be populated in subsequent calls to [wlan\\_add\\_network](#). This saves the CPU time required to otherwise calculate the PMK.

#### 4.47.2.10 pmk\_valid

```
bool wlan_network_security::pmk_valid
```

Flag reporting whether pmk is valid or not.

#### 4.47.2.11 mfpc

```
bool wlan_network_security::mfpc
```

Management Frame Protection Capable (MFPC)

#### 4.47.2.12 mfpr

```
bool wlan_network_security::mfpr
```

Management Frame Protection Required (MFPR)

The documentation for this struct was generated from the following file:

- [wlan.h](#)

## 4.48 wlan\_scan\_result Struct Reference

### Data Fields

- char [ssid](#) [33]
- unsigned int [ssid\\_len](#)
- char [bssid](#) [6]
- unsigned int [channel](#)
- enum [wlan\\_bss\\_type](#) type
- enum [wlan\\_bss\\_role](#) role
- unsigned [wmm](#): 1
- unsigned [wpa2\\_entp](#): 1
- unsigned [wep](#): 1
- unsigned [wpa](#): 1
- unsigned [wpa2](#): 1
- unsigned [wpa3\\_sae](#): 1
- unsigned char [rssi](#)
- char [trans\\_ssid](#) [33]
- unsigned int [trans\\_ssid\\_len](#)
- char [trans\\_bssid](#) [6]
- uint16\_t [beacon\\_period](#)
- uint8\_t [dtim\\_period](#)

### 4.48.1 Detailed Description

Scan Result

### 4.48.2 Field Documentation

#### 4.48.2.1 ssid

```
char wlan_scan_result::ssid[33]
```

The network SSID, represented as a NULL-terminated C string of 0 to 32 characters. If the network has a hidden SSID, this will be the empty string.

#### 4.48.2.2 ssid\_len

```
unsigned int wlan_scan_result::ssid_len
```

SSID length

#### 4.48.2.3 bssid

```
char wlan_scan_result::bssid[6]
```

The network BSSID, represented as a 6-byte array.



#### 4.48.2.4 channel

```
unsigned int wlan_scan_result::channel
```

The network channel.

#### 4.48.2.5 type

```
enum wlan_bss_type wlan_scan_result::type
```

The network wireless type.

#### 4.48.2.6 role

```
enum wlan_bss_role wlan_scan_result::role
```

The network wireless mode.

#### 4.48.2.7 wmm

```
unsigned wlan_scan_result::wmm
```

The network supports WMM. This is set to 0 if the network does not support WMM or if the system does not have WMM support enabled.

#### 4.48.2.8 wpa2\_entp

```
unsigned wlan_scan_result::wpa2_entp
```

WPA2 Enterprise security

#### 4.48.2.9 wep

```
unsigned wlan_scan_result::wep
```

The network uses WEP security.

#### 4.48.2.10 wpa

```
unsigned wlan_scan_result::wpa
```

The network uses WPA security.

#### 4.48.2.11 wpa2

```
unsigned wlan_scan_result::wpa2
```

The network uses WPA2 security

#### 4.48.2.12 wpa3\_sae

```
unsigned wlan_scan_result::wpa3_sae
```

The network uses WPA3 SAE security

#### 4.48.2.13 rssi

```
unsigned char wlan_scan_result::rssi
```

The signal strength of the beacon

#### 4.48.2.14 trans\_ssid

```
char wlan_scan_result::trans_ssid[33]
```

The network SSID, represented as a NULL-terminated C string of 0 to 32 characters. If the network has a hidden SSID, this will be the empty string.

#### 4.48.2.15 trans\_ssid\_len

```
unsigned int wlan_scan_result::trans_ssid_len
```

SSID length

#### 4.48.2.16 trans\_bssid

```
char wlan_scan_result::trans_bssid[6]
```

The network BSSID, represented as a 6-byte array.

#### 4.48.2.17 beacon\_period

```
uint16_t wlan_scan_result::beacon_period
```

Beacon Period

#### 4.48.2.18 dtim\_period

```
uint8_t wlan_scan_result::dtim_period
```

DTIM Period

The documentation for this struct was generated from the following file:

- [wlan.h](#)

# Chapter 5

## File Documentation

### 5.1 cli.h File Reference

CLI module.

#### 5.1.1 Detailed Description

#### 5.1.2 Usage

The CLI module lets you register commands with the CLI interface. Modules that wish to register the commands should initialize the struct `cli_command` structure and pass this to `cli_register_command()`. These commands will then be available on the CLI.

#### 5.1.3 Function Documentation

##### 5.1.3.1 cli\_register\_command()

```
int cli_register_command (
    const struct cli_command * command )
```

Register a CLI command

This function registers a command with the command-line interface.

##### Parameters

in	<i>command</i>	The structure to register one CLI command
----	----------------	---

**Returns**

0 on success  
1 on failure

**5.1.3.2 cli\_unregister\_command()**

```
int cli_unregister_command (
    const struct cli_command * command )
```

**Unregister a CLI command**

This function unregisters a command from the command-line interface.

**Parameters**

in	<i>command</i>	The structure to unregister one CLI command
----	----------------	---

**Returns**

0 on success  
1 on failure

**5.1.3.3 cli\_init()**

```
int cli_init (
    void )
```

**Initialize the CLI module****Returns**

WM\_SUCCESS on success  
error code otherwise.

**5.1.3.4 cli\_stop()**

```
int cli_stop (
    void )
```

**Stop the CLI thread and carry out the cleanup****Returns**

WM\_SUCCESS on success  
error code otherwise.

### 5.1.3.5 cli\_register\_commands()

```
int cli_register_commands (
    const struct cli_command * commands,
    int num_commands )
```

Register a batch of CLI commands

Often, a module will want to register several commands.

#### Parameters

in	<i>commands</i>	Pointer to an array of commands.
in	<i>num_commands</i>	Number of commands in the array.

#### Returns

0 on success  
1 on failure

### 5.1.3.6 cli\_unregister\_commands()

```
int cli_unregister_commands (
    const struct cli_command * commands,
    int num_commands )
```

Unregister a batch of CLI commands

#### Parameters

in	<i>commands</i>	Pointer to an array of commands.
in	<i>num_commands</i>	Number of commands in the array.

#### Returns

0 on success  
1 on failure

## 5.2 dhcp-server.h File Reference

DHCP server.

### 5.2.1 Detailed Description

The DHCP Server is required in the provisioning mode of the application to assign IP Address to Wireless Clients that connect to the WM.

## 5.2.2 Function Documentation

### 5.2.2.1 dhcpd\_cli\_init()

```
int dhcpd_cli_init (
    void )
```

Register DHCP server commands

This function registers the CLI dhcp-stat for the DHCP server. dhcp-stat command displays ip to associated client mac mapping.

#### Returns

-WM\_E\_DHCPD\_REGISTER\_CMDS if cli init operation failed.  
WM\_SUCCESS if cli init operation success.

### 5.2.2.2 dhcp\_server\_start()

```
int dhcp_server_start (
    void * intrfc_handle )
```

Start DHCP server

This starts the DHCP server on the interface specified. Typically DHCP server should be running on the micro-AP interface but it can also run on wifi direct interface if configured as group owner. Use [net\\_get\\_uap\\_handle\(\)](#) to get micro-AP interface handle.

#### Parameters

in	<i>intrfc_handle</i>	The interface handle on which DHCP server will start
----	----------------------	--

#### Returns

WM\_SUCCESS on success or error code

### 5.2.2.3 dhcp\_enable\_dns\_server()

```
void dhcp_enable_dns_server (
    char ** domain_names )
```

Start DNS server

This starts the DNS server on the interface specified for dhcp server. This function needs to be used before `dhcp_server_start()` function and can be invoked on receiving `WLAN_REASON_INITIALIZED` event in the application before starting micro-AP.

The application needs to define its own list of domain names with the last entry as NULL. The dns server handles dns queries and if domain name match is found then resolves it to device ip address. Currently the maximum length for each domain name is set to 32 bytes.

Eg. `char *domain_names[] = {"nxpprov.net", "www.nxpprov.net", NULL};`

`dhcp_enable_dns_server(domain_names);`

However, application can also start dns server without any domain names specified to solve following issue. Some of the client devices do not show WiFi signal strength symbol when connected to micro-AP in open mode, if dns queries are not resolved. With dns server support enabled, dns server responds with `ERROR_REFUSED` indicating that the DNS server refuses to provide whatever data client is asking for.

#### Parameters

in	<i>domain_names</i>	Pointer to the list of domain names or NULL.
----	---------------------	--

#### 5.2.2.4 dhcp\_server\_stop()

```
void dhcp_server_stop (
    void )
```

Stop DHCP server

#### 5.2.2.5 dhcp\_server\_lease\_timeout()

```
int dhcp_server_lease_timeout (
    uint32_t val )
```

Configure the DHCP dynamic IP lease time

This API configures the dynamic IP lease time, which should be invoked before DHCP server initialization

#### Parameters

in	<i>val</i>	Number of seconds, use (60U*60U*number of hours) for clarity. Max value is (60U*60U*24U*49700U)
----	------------	---

#### Returns

Error status code

### 5.2.2.6 dhcp\_get\_ip\_from\_mac()

```
int dhcp_get_ip_from_mac (
    uint8_t * client_mac,
    uint32_t * client_ip )
```

Get IP address corresponding to MAC address from dhcpd ip-mac mapping

This API returns IP address mapping to the MAC address present in cache. IP-MAC cache stores MAC to IP mapping of previously or currently connected clients.

#### Parameters

in	<i>client_mac</i>	Pointer to a six byte array containing the MAC address of the client
out	<i>client_ip</i>	Pointer to IP address of the client

#### Returns

WM\_SUCCESS on success or -WM\_FAIL.

### 5.2.2.7 dhcp\_stat()

```
void dhcp_stat (
    void )
```

Print DHCP stats on the console

This API prints DHCP stats on the console

## 5.2.3 Enumeration Type Documentation

### 5.2.3.1 wm\_dhcpd\_errno

```
enum wm_dhcpd_errno
```

#### DHCPD Error Codes

##### Enumerator

WM_E_DHCPD_SERVER_RUNNING	Dhcp server is already running
WM_E_DHCPD_THREAD_CREATE	Failed to create dhcp thread
WM_E_DHCPD_MUTEX_CREATE	Failed to create dhcp mutex
WM_E_DHCPD_REGISTER_CMDS	Failed to register dhcp commands
WM_E_DHCPD_RESP_SEND	Failed to send dhcp response
WM_E_DHCPD_DNS_IGNORE	Ignore as msg is not a valid dns query



## Enumerator

WM_E_DHCPD_BUFFER_FULL	Buffer overflow occurred
WM_E_DHCPD_INVALID_INPUT	The input message is NULL or has incorrect length
WM_E_DHCPD_INVALID_OPCODE	Invalid opcode in the dhcp message
WM_E_DHCPD_INCORRECT_HEADER	Invalid header type or incorrect header length
WM_E_DHCPD_SPOOF_NAME	Spoof length is either NULL or it exceeds max length
WM_E_DHCPD_BCAST_ADDR	Failed to get broadcast address
WM_E_DHCPD_IP_ADDR	Failed to look up requested IP address from the interface
WM_E_DHCPD_NETMASK	Failed to look up requested netmask from the interface
WM_E_DHCPD_SOCKET	Failed to create the socket
WM_E_DHCPD_ARP_SEND	Failed to send Gratuitous ARP
WM_E_DHCPD_IOCTL_CALL	Error in ioctl call
WM_E_DHCPD_INIT	Failed to init dhcp server

## 5.3 iperf.h File Reference

This file provides the support for network utility iperf.

### 5.3.1 Function Documentation

#### 5.3.1.1 iperf\_cli\_init()

```
int iperf_cli_init ( )
```

Register the Network Utility CLI command iperf.

#### Note

This function can only be called by the application after [wlan\\_init\(\)](#) called.

#### Returns

WM\_SUCCESS if the CLI commands are registered  
 -WM\_FAIL otherwise (for example if this function was called while the CLI commands were already registered)

#### 5.3.1.2 iperf\_cli\_deinit()

```
int iperf_cli_deinit ( )
```

Unregister Network Utility CLI command iperf.

#### Returns

WM\_SUCCESS if the CLI commands are unregistered  
 -WM\_FAIL otherwise

## 5.4 wifi-decl.h File Reference

Wifi structure declarations.

### 5.4.1 Macro Documentation

#### 5.4.1.1 MLAN\_MAX\_VER\_STR\_LEN

```
#define MLAN_MAX_VER_STR_LEN 128
```

Version string buffer length

#### 5.4.1.2 BSS\_TYPE\_STA

```
#define BSS_TYPE_STA 0U
```

BSS type : STA

#### 5.4.1.3 BSS\_TYPE\_UAP

```
#define BSS_TYPE_UAP 1U
```

BSS type : UAP

#### 5.4.1.4 MLAN\_MAX\_SSID\_LENGTH

```
#define MLAN_MAX_SSID_LENGTH (32U)
```

MLAN Maximum SSID Length

#### 5.4.1.5 MLAN\_MAX\_PASS\_LENGTH

```
#define MLAN_MAX_PASS_LENGTH (64)
```

MLAN Maximum PASSPHRASE Length

### 5.4.2 Enumeration Type Documentation

#### 5.4.2.1 wifi\_SubBand\_t

```
enum wifi_SubBand_t
```

Wifi subband enum

## Enumerator

SubBand_2_4_GHz	Subband 2.4 GHz
SubBand_5_GHz↔ _0	Subband 5 GHz 0
SubBand_5_GHz↔ _1	Subband 5 GHz 1
SubBand_5_GHz↔ _2	Subband 5 GHz 2
SubBand_5_GHz↔ _3	Subband 5 GHz 3

## 5.4.2.2 wifi\_frame\_type\_t

```
enum wifi_frame_type_t
```

Wifi frame types

## Enumerator

ASSOC_REQ_FRAME	Assoc request frame
ASSOC_RESP_FRAME	Assoc response frame
REASSOC_REQ_FRAME	ReAssoc request frame
REASSOC_RESP_FRAME	ReAssoc response frame
PROBE_REQ_FRAME	Probe request frame
PROBE_RESP_FRAME	Probe response frame
BEACON_FRAME	BEACON frame
DISASSOC_FRAME	Dis assoc frame
AUTH_FRAME	Auth frame
DEAUTH_FRAME	Deauth frame
ACTION_FRAME	Action frame
DATA_FRAME	Data frame
QOS_DATA_FRAME	QOS frame

## 5.5 wifi.h File Reference

This file contains interface to wifi driver.

## 5.5.1 Function Documentation

### 5.5.1.1 wifi\_init()

```
int wifi_init (
    const uint8_t * fw_start_addr,
    const size_t size )
```

Initialize Wi-Fi driver module.

Performs SDIO init, downloads Wi-Fi Firmware, creates Wi-Fi Driver and command response processor thread.

Also creates mutex, and semaphores used in command and data synchronizations.

#### Parameters

in	<i>fw_start_addr</i>	address of stored Wi-Fi Firmware.
in	<i>size</i>	Size of Wi-Fi Firmware.

#### Returns

WM\_SUCCESS on success or -WM\_FAIL on error.

### 5.5.1.2 wifi\_init\_fcc()

```
int wifi_init_fcc (
    const uint8_t * fw_start_addr,
    const size_t size )
```

Initialize Wi-Fi driver module for FCC Certification.

Performs SDIO init, downloads Wi-Fi Firmware, creates Wi-Fi Driver and command response processor thread.

Also creates mutex, and semaphores used in command and data synchronizations.

#### Parameters

in	<i>fw_start_addr</i>	address of stored Manufacturing Wi-Fi Firmware.
in	<i>size</i>	Size of Manufacturing Wi-Fi Firmware.

#### Returns

WM\_SUCCESS on success or -WM\_FAIL on error.

### 5.5.1.3 wifi\_deinit()

```
void wifi_deinit (
    void )
```

Deinitialize Wi-Fi driver module.

Performs SDIO deinit, send shutdown command to Wi-Fi Firmware, deletes Wi-Fi Driver and command processor thread.

Also deletes mutex and semaphores used in command and data synchronizations.

#### 5.5.1.4 wifi\_register\_data\_input\_callback()

```
int wifi_register_data_input_callback (
    void(*) (const uint8_t interface, const uint8_t *buffer, const uint16_t len) data↔
    _input_callback )
```

Register Data callback function with Wi-Fi Driver to receive DATA from SDIO.

This callback function is used to send data received from Wi-Fi firmware to the networking stack.

##### Parameters

in	<i>data_input_callback</i>	Function that needs to be called
----	----------------------------	----------------------------------

##### Returns

WM\_SUCCESS

#### 5.5.1.5 wifi\_deregister\_data\_input\_callback()

```
void wifi_deregister_data_input_callback (
    void )
```

Deregister Data callback function from Wi-Fi Driver

#### 5.5.1.6 wifi\_register\_amsdu\_data\_input\_callback()

```
int wifi_register_amsdu_data_input_callback (
    void(*) (uint8_t interface, uint8_t *buffer, uint16_t len) amsdu_data_input↔
    callback )
```

Register Data callback function with Wi-Fi Driver to receive processed AMSDU DATA from Wi-Fi driver.

This callback function is used to send data received from Wi-Fi firmware to the networking stack.

##### Parameters

in	<i>amsdu_data_input_callback</i>	Function that needs to be called
----	----------------------------------	----------------------------------

**Returns**

WM\_SUCCESS

**5.5.1.7 wifi\_deregister\_amsdu\_data\_input\_callback()**

```
void wifi_deregister_amsdu_data_input_callback (
    void )
```

Deregister Data callback function from Wi-Fi Driver

**5.5.1.8 wifi\_low\_level\_output()**

```
int wifi_low_level_output (
    const uint8_t interface,
    const uint8_t * buffer,
    const uint16_t len )
```

Wi-Fi Driver low level output function.

Data received from upper layer is passed to Wi-Fi Driver for transmission.

**Parameters**

in	<i>interface</i>	Interface on which DATA frame will be transmitted. 0 for Station interface, 1 for uAP interface and 2 for Wi-Fi Direct interface.
in	<i>buffer</i>	A pointer pointing to DATA frame.
in	<i>len</i>	Length of DATA frame.

**Returns**

WM\_SUCCESS on success or -WM\_E\_NOMEM if memory is not available or -WM\_E\_BUSY if SDIO is busy.

**5.5.1.9 wifi\_set\_packet\_retry\_count()**

```
void wifi_set_packet_retry_count (
    const int count )
```

API to enable packet retries at wifi driver level.

This API sets retry count which will be used by wifi driver to retry packet transmission in case there was failure in earlier attempt. Failure may happen due to SDIO write port un-availability or other failures in SDIO write operation.

**Note**

Default value of retry count is zero.

## Parameters

in	<i>count</i>	No of retry attempts.
----	--------------	-----------------------

5.5.1.10 `wifi_sta_ampdu_tx_enable()`

```
void wifi_sta_ampdu_tx_enable (  
    void )
```

This API can be used to enable AMPDU support on the go when station is a transmitter.

5.5.1.11 `wifi_sta_ampdu_tx_disable()`

```
void wifi_sta_ampdu_tx_disable (  
    void )
```

This API can be used to disable AMPDU support on the go when station is a transmitter.

5.5.1.12 `wifi_sta_ampdu_rx_enable()`

```
void wifi_sta_ampdu_rx_enable (  
    void )
```

This API can be used to enable AMPDU support on the go when station is a receiver.

5.5.1.13 `wifi_sta_ampdu_rx_disable()`

```
void wifi_sta_ampdu_rx_disable (  
    void )
```

This API can be used to disable AMPDU support on the go when station is a receiver.

5.5.1.14 `wifi_get_device_mac_addr()`

```
int wifi_get_device_mac_addr (  
    wifi_mac_addr_t * mac_addr )
```

Get the device MAC address

## Parameters

out	<i>mac_addr</i>	Mac address
-----	-----------------	-------------

**Returns**

WM\_SUCCESS

**5.5.1.15 wifi\_get\_device\_firmware\_version\_ext()**

```
int wifi_get_device_firmware_version_ext (
    wifi_fw_version_ext_t * fw_ver_ext )
```

Get the cached string representation of the wlan firmware extended version.

**Parameters**

in	<i>fw_ver_ext</i>	Firmware Version Extended
----	-------------------	---------------------------

**Returns**

WM\_SUCCESS

**5.5.1.16 wifi\_get\_last\_cmd\_sent\_ms()**

```
unsigned wifi_get_last_cmd_sent_ms (
    void )
```

Get the timestamp of the last command sent to the firmware

**Returns**

Timestamp in millisec of the last command sent

**5.5.1.17 wifi\_update\_last\_cmd\_sent\_ms()**

```
void wifi_update_last_cmd_sent_ms (
    void )
```

This will update the last command sent variable value to current time. This is used for power management.

**5.5.1.18 wifi\_register\_event\_queue()**

```
int wifi_register_event_queue (
    os_queue_t * event_queue )
```

Register an event queue with the wifi driver to receive events

The list of events which can be received from the wifi driver are enumerated in the file [wifi\\_events.h](#)



## Parameters

in	<i>event_queue</i>	The queue to which wifi driver will post events.
----	--------------------	--

## Note

Only one queue can be registered. If the registered queue needs to be changed unregister the earlier queue first.

## Returns

Standard SDK return codes

## 5.5.1.19 wifi\_unregister\_event\_queue()

```
int wifi_unregister_event_queue (
    os_queue_t * event_queue )
```

Unregister an event queue from the wifi driver.

## Parameters

in	<i>event_queue</i>	The queue to which was registered earlier with the wifi driver.
----	--------------------	---

## Returns

Standard SDK return codes

## 5.5.1.20 wifi\_get\_scan\_result()

```
int wifi_get_scan_result (
    unsigned int index,
    struct wifi_scan_result ** desc )
```

Get scan list

## Parameters

in	<i>index</i>	Index
out	<i>desc</i>	Descriptor of type <a href="#">wifi_scan_result</a>

## Returns

WM\_SUCCESS on success or error code.

### 5.5.1.21 wifi\_get\_scan\_result\_count()

```
int wifi_get_scan_result_count (
    unsigned * count )
```

Get the count of elements in the scan list

#### Parameters

in, out	count	Pointer to a variable which will hold the count after this call returns
---------	-------	---

#### Warning

The count returned by this function is the current count of the elements. A scan command given to the driver or some other background event may change this count in the wifi driver. Thus when the API [wifi\\_get\\_scan\\_result](#) is used to get individual elements of the scan list, do not assume that it will return exactly 'count' number of elements. Your application should not consider such situations as a major event.

#### Returns

Standard SDK return codes.

### 5.5.1.22 wifi\_uap\_bss\_sta\_list()

```
int wifi_uap_bss_sta_list (
    wifi_sta_list_t ** list )
```

Returns the current STA list connected to our uAP

This function gets its information after querying the firmware. It will block till the response is received from firmware or a timeout.

#### Parameters

in, out	list	After this call returns this points to the structure <a href="#">wifi_sta_list_t</a> allocated by the callee. This is variable length structure and depends on count variable inside it. <b>The caller needs to free this buffer after use..</b> If this function is unable to get the sta list, the value of list parameter will be NULL
---------	------	---

#### Note

The caller needs to explicitly free the buffer returned by this function.

#### Returns

void

#### 5.5.1.23 wifi\_set\_cal\_data()

```
void wifi_set_cal_data (
    uint8_t * cdata,
    unsigned int clen )
```

Set wifi calibration data in firmware.

This function may be used to set wifi calibration data in firmware.

##### Parameters

in	<i>cdata</i>	The calibration data
in	<i>clen</i>	Length of calibration data

#### 5.5.1.24 wifi\_set\_mac\_addr()

```
void wifi_set_mac_addr (
    uint8_t * mac )
```

Set wifi MAC address in firmware at load time.

This function may be used to set wifi MAC address in firmware.

##### Parameters

in	<i>mac</i>	The new MAC Address
----	------------	---------------------

#### 5.5.1.25 \_wifi\_set\_mac\_addr()

```
void _wifi_set_mac_addr (
    uint8_t * mac )
```

Set wifi MAC address in firmware at run time.

This function may be used to set wifi MAC address in firmware.

##### Parameters

in	<i>mac</i>	The new MAC Address
----	------------	---------------------

### 5.5.1.26 wifi\_add\_mcast\_filter()

```
int wifi_add_mcast_filter (
    uint8_t * mac_addr )
```

#### Add Multicast Filter by MAC Address

Multicast filters should be registered with the WiFi driver for IP-level multicast addresses to work. This API allows for registration of such filters with the WiFi driver.

If multicast-mapped MAC address is 00:12:23:34:45:56 then pass mac\_addr as below: mac\_addr[0] = 0x00 mac\_addr[1] = 0x12 mac\_addr[2] = 0x23 mac\_addr[3] = 0x34 mac\_addr[4] = 0x45 mac\_addr[5] = 0x56

#### Parameters

in	mac_addr	multicast mapped MAC address
----	----------	------------------------------

#### Returns

0 on Success or else Error

### 5.5.1.27 wifi\_remove\_mcast\_filter()

```
int wifi_remove_mcast_filter (
    uint8_t * mac_addr )
```

#### Remove Multicast Filter by MAC Address

This function removes multicast filters for the given multicast-mapped MAC address. If multicast-mapped MAC address is 00:12:23:34:45:56 then pass mac\_addr as below: mac\_addr[0] = 0x00 mac\_addr[1] = 0x12 mac\_addr[2] = 0x23 mac\_addr[3] = 0x34 mac\_addr[4] = 0x45 mac\_addr[5] = 0x56

#### Parameters

in	mac_addr	multicast mapped MAC address
----	----------	------------------------------

#### Returns

0 on Success or else Error

### 5.5.1.28 wifi\_get\_ipv4\_multicast\_mac()

```
void wifi_get_ipv4_multicast_mac (
    uint32_t ipaddr,
    uint8_t * mac_addr )
```

Get Multicast Mapped Mac address from IPv4

This function will generate Multicast Mapped MAC address from IPv4 Multicast Mapped MAC address will be in following format: 1) Higher 24-bits filled with IANA Multicast OUI (01-00-5E) 2) 24th bit set as Zero 3) Lower 23-bits filled with IP address (ignoring higher 9bits).

#### Parameters

in	<i>ipaddr</i>	ipaddress(input)
in	<i>mac_addr</i>	multicast mapped MAC address(output)

#### Returns

void

#### 5.5.1.29 wifi\_get\_region\_code()

```
int wifi_get_region_code (
    t_u32 * region_code )
```

Get the wifi region code

This function will return one of the following values in the region\_code variable.

0x10 : US FCC  
 0x20 : CANADA  
 0x30 : EU  
 0x32 : FRANCE  
 0x40 : JAPAN  
 0x41 : JAPAN  
 0x50 : China  
 0xfe : JAPAN  
 0xff : Special

#### Parameters

out	<i>region_code</i>	Region Code
-----	--------------------	-------------

#### Returns

Standard WMSDK return codes.

#### 5.5.1.30 wifi\_set\_region\_code()

```
int wifi_set_region_code (
    t_u32 region_code )
```

Set the wifi region code.

This function takes one of the values from the following array.

0x10 : US FCC  
0x20 : CANADA  
0x30 : EU  
0x32 : FRANCE  
0x40 : JAPAN  
0x41 : JAPAN  
0x50 : China  
0xfe : JAPAN  
0xff : Special

#### Parameters

in	<i>region_code</i>	Region Code
----	--------------------	-------------

#### Returns

Standard WMSDK return codes.

#### 5.5.1.31 wifi\_get\_uap\_channel()

```
int wifi_get_uap_channel (
    int * channel )
```

Get the uAP channel number

#### Parameters

in	<i>channel</i>	Pointer to channel number. Will be initialized by callee
----	----------------	--

#### Returns

Standard WMSDK return code

#### 5.5.1.32 wifi\_enable\_11d\_support()

```
int wifi_enable_11d_support (
    void )
```

Sets the domain parameters for the uAP.

**Note**

This API only saves the domain params inside the driver internal structures. The actual application of the params will happen only during starting phase of uAP. So, if the uAP is already started then the configuration will not apply till uAP re-start.

To use this API you will need to fill up the structure `wifi_domain_param_t` with correct parameters.

E.g. Programming for US country code

```
wifi_sub_band_set_t sb = { .first_chan = 1, .no_of_chan= 11, .max_tx_pwr = 30, };
```

```
wifi_domain_param_t *dp = os_mem_alloc(sizeof(wifi_domain_param_t) + sizeof(wifi_sub_band_set_t));
```

```
(void)memcpy(dp->country_code, "US\0", COUNTRY_CODE_LEN); dp->no_of_sub_band = 1; (void)memcpy(dp->sub_band, &sb, sizeof(wifi_sub_band_set_t));
```

```
wmprintf("wifi uap set domain params\n\r"); wifi_uap_set_domain_params(dp); os_mem_free(dp);
```

**Returns**

WM\_SUCCESS on success or error code.

**5.5.2 Enumeration Type Documentation****5.5.2.1 anonymous enum**

anonymous enum

WiFi Error Code

Enumerator

WIFI_ERROR_FW_DNLD_FAILED	The Firmware download operation failed.
WIFI_ERROR_FW_NOT_READY	The Firmware ready register not set.
WIFI_ERROR_CARD_NOT_DETECTED	The WiFi card not found.
WIFI_ERROR_FW_NOT_DETECTED	The WiFi Firmware not found.

**5.5.2.2 country\_code\_t**

enum `country_code_t`

802.11d country codes

## Enumerator

COUNTRY_WW	World Wide Safe Mode
COUNTRY_US	US FCC
COUNTRY_CA	IC Canada
COUNTRY_SG	Singapore
COUNTRY_EU	ETSI
COUNTRY_AU	Australia
COUNTRY_KR	Republic Of Korea
COUNTRY_FR	France
COUNTRY_JP	Japan
COUNTRY_CN	China

## 5.6 wifi\_events.h File Reference

Wi-Fi events.

### 5.6.1 Enumeration Type Documentation

#### 5.6.1.1 wifi\_event

```
enum wifi_event
```

Wifi events

## Enumerator

WIFI_EVENT_UAP_STARTED	uAP Started
WIFI_EVENT_UAP_CLIENT_ASSOC	uAP Client Assoc
WIFI_EVENT_UAP_CLIENT_DEAUTH	uAP Client De-authentication
WIFI_EVENT_UAP_NET_ADDR_CONFIG	uAP Network Address Configuration
WIFI_EVENT_UAP_STOPPED	uAP Stopped
WIFI_EVENT_UAP_LAST	uAP Last
WIFI_EVENT_SCAN_RESULT	Scan Result
WIFI_EVENT_GET_HW_SPEC	Get hardware spec
WIFI_EVENT_ASSOCIATION	Association
WIFI_EVENT_PMK	PMK
WIFI_EVENT_AUTHENTICATION	Authentication
WIFI_EVENT_DISASSOCIATION	Disassociation
WIFI_EVENT_DEAUTHENTICATION	De-authentication
WIFI_EVENT_LINK_LOSS	Link Loss
WIFI_EVENT_NET_STA_ADDR_CONFIG	Network station address configuration
WIFI_EVENT_NET_INTERFACE_CONFIG	Network interface configuration
WIFI_EVENT_WEP_CONFIG	WEP configuration



## Enumerator

WIFI_EVENT_MAC_ADDR_CONFIG	MAC address configuration
WIFI_EVENT_NET_DHCP_CONFIG	Network DHCP configuration
WIFI_EVENT_SUPPLICANT_PMK	Supplicant PMK
WIFI_EVENT_SLEEP	Sleep
WIFI_EVENT_AWAKE	Awake
WIFI_EVENT_IEEE_PS	IEEE PS
WIFI_EVENT_DEEP_SLEEP	Deep Sleep
WIFI_EVENT_PS_INVALID	PS Invalid
WIFI_EVENT_HS_CONFIG	HS configuration
WIFI_EVENT_ERR_MULTICAST	Error Multicast
WIFI_EVENT_ERR_UNICAST	error Unicast
WIFI_EVENT_11N_ADDBA	802.11N add block ack
WIFI_EVENT_11N_BA_STREAM_TIMEOUT	802.11N block Ack stream timeout
WIFI_EVENT_11N_DELBA	802.11n Delete block add
WIFI_EVENT_11N_AGGR_CTRL	802.11n aggregation control
WIFI_EVENT_CHAN_SWITCH_ANN	Channel Switch Announcement
WIFI_EVENT_CHAN_SWITCH	Channel Switch
WIFI_EVENT_LAST	Event to indicate end of Wi-Fi events

## 5.6.1.2 wifi\_event\_reason

```
enum wifi_event_reason
```

## WiFi Event Reason

## Enumerator

WIFI_EVENT_REASON_SUCCESS	Success
WIFI_EVENT_REASON_TIMEOUT	Timeout
WIFI_EVENT_REASON_FAILURE	Failure

## 5.6.1.3 wlan\_bss\_type

```
enum wlan_bss_type
```

## Network wireless BSS Type

## Enumerator

WLAN_BSS_TYPE_STA	Station
WLAN_BSS_TYPE_UAP	uAP
WLAN_BSS_TYPE_ANY	Any

#### 5.6.1.4 wlan\_bss\_role

enum `wlan_bss_role`

Network wireless BSS Role

##### Enumerator

WLAN_BSS_ROLE_STA	Infrastructure network. The system will act as a station connected to an Access Point.
WLAN_BSS_ROLE_UAP	uAP (micro-AP) network. The system will act as an uAP node to which other Wireless clients can connect.
WLAN_BSS_ROLE_ANY	Either Infrastructure network or micro-AP network

#### 5.6.1.5 wifi\_wakeup\_event\_t

enum `wifi_wakeup_event_t`

This enum defines various wakeup events for which wakeup will occur

##### Enumerator

WIFI_WAKE_ON_ALL_BROADCAST	Wakeup on broadcast
WIFI_WAKE_ON_UNICAST	Wakeup on unicast
WIFI_WAKE_ON_MAC_EVENT	Wakeup on MAC event
WIFI_WAKE_ON_MULTICAST	Wakeup on multicast
WIFI_WAKE_ON_ARP_BROADCAST	Wakeup on ARP broadcast
WIFI_WAKE_ON_MGMT_FRAME	Wakeup on receiving a management frame

## 5.7 wlan.h File Reference

WLAN Connection Manager.

### 5.7.1 Detailed Description

The WLAN Connection Manager (WLCMGR) is one of the core components that provides WiFi-level functionality like scanning for networks, starting a network (Access Point) and associating / disassociating with other wireless networks. The WLCMGR manages two logical interfaces, the station interface and the micro-AP interface. Both these interfaces can be active at the same time.

## 5.7.2 Usage

The WLCMGR is initialized by calling [wlan\\_init\(\)](#) and started by calling [wlan\\_start\(\)](#), one of the arguments of this function is a callback handler. Many of the WLCMGR tasks are asynchronous in nature, and the events are provided by invoking the callback handler. The various usage scenarios of the WLCMGR are outlined below:

- **Scanning:** A call to [wlan\\_scan\(\)](#) initiates an asynchronous scan of the nearby wireless networks. The results are reported via the callback handler.
- **Network Profiles:** Starting / stopping wireless interfaces or associating / disassociating with other wireless networks is managed through network profiles. The network profiles record details about the wireless network like the SSID, type of security, security passphrase among other things. The network profiles can be managed by means of the [wlan\\_add\\_network\(\)](#) and [wlan\\_remove\\_network\(\)](#) calls.
- **Association:** The [wlan\\_connect\(\)](#) and [wlan\\_disconnect\(\)](#) calls can be used to manage connectivity with other wireless networks (Access Points). These calls manage the station interface of the system.
- **Starting a Wireless Network:** The [wlan\\_start\\_network\(\)](#) and [wlan\\_stop\\_network\(\)](#) calls can be used to start/stop our own (micro-AP) network. These calls manage the micro-AP interface of the system.

## 5.7.3 Function Documentation

### 5.7.3.1 wlan\_init()

```
int wlan_init (
    const uint8_t * fw_start_addr,
    const size_t size )
```

Initialize the SDIO driver and create the wifi driver thread.

#### Parameters

in	<i>fw_start_addr</i>	Start address of the WLAN firmware.
in	<i>size</i>	Size of the WLAN firmware.

#### Returns

WM\_SUCCESS if the WLAN Connection Manager service has initialized successfully.  
Negative value if initialization failed.

### 5.7.3.2 wlan\_start()

```
int wlan_start (
    int (*) (enum wlan_event_reason reason, void *data) cb )
```

Start the WLAN Connection Manager service.

This function starts the WLAN Connection Manager.

**Note**

The status of the WLAN Connection Manager is notified asynchronously through the callback, *cb*, with a `WLAN_REASON_INITIALIZED` event (if initialization succeeded) or `WLAN_REASON_INITIALIZATION_FAILED` (if initialization failed).

If the WLAN Connection Manager fails to initialize, the caller should stop WLAN Connection Manager via `wlan_stop()` and try `wlan_start()` again.

**Parameters**

in	<i>cb</i>	A pointer to a callback function that handles WLAN events. All further WLCMGR events will be notified in this callback. Refer to enum <code>wlan_event_reason</code> for the various events for which this callback is called.
----	-----------	--

**Returns**

`WM_SUCCESS` if the WLAN Connection Manager service has started successfully.

-`WM_E_INVALID` if the *cb* pointer is NULL.

-`WM_FAIL` if an internal error occurred.

`WLAN_ERROR_STATE` if the WLAN Connection Manager is already running.

**5.7.3.3 wlan\_stop()**

```
int wlan_stop (
    void )
```

Stop the WLAN Connection Manager service.

This function stops the WLAN Connection Manager, causing station interface to disconnect from the currently connected network and stop the micro-AP interface.

**Returns**

`WM_SUCCESS` if the WLAN Connection Manager service has been stopped successfully.

`WLAN_ERROR_STATE` if the WLAN Connection Manager was not running.

**5.7.3.4 wlan\_deinit()**

```
void wlan_deinit (
    int action )
```

Deinitialize SDIO driver, send shutdown command to WLAN firmware and delete the wifi driver thread.

**Parameters**

<i>action</i>	Additional action to be taken with deinit <code>WLAN_ACTIVE</code> : no action to be taken
---------------	--

### 5.7.3.5 wlan\_initialize\_uap\_network()

```
void wlan_initialize_uap_network (
    struct wlan_network * net )
```

WLAN initialize micro-AP network information

This API initializes a default micro-AP network. The network ssid, passphrase is initialized to NULL. Channel is set to auto. The IP Address of the micro-AP interface is 192.168.10.1/255.255.255.0. Network name is set to 'uap-network'.

#### Parameters

out	<i>net</i>	Pointer to the initialized micro-AP network
-----	------------	---

### 5.7.3.6 wlan\_add\_network()

```
int wlan_add_network (
    struct wlan_network * network )
```

Add a network profile to the list of known networks.

This function copies the contents of *network* to the list of known networks in the WLAN Connection Manager. The network's 'name' field must be unique and between [WLAN\\_NETWORK\\_NAME\\_MIN\\_LENGTH](#) and [WLAN\\_NETWORK\\_NAME\\_MAX\\_LENGTH](#) characters. The network must specify at least an SSID or BSSID. The WLAN Connection Manager may store up to [WLAN\\_MAX\\_KNOWN\\_NETWORKS](#) networks.

#### Note

Profiles for the station interface may be added only when the station interface is in the [WLAN\\_DISCONNECTED](#) or [WLAN\\_CONNECTED](#) state.

This API can be used to add profiles for station or micro-AP interfaces.

#### Parameters

in	<i>network</i>	A pointer to the <a href="#">wlan_network</a> that will be copied to the list of known networks in the WLAN Connection Manager successfully.
----	----------------	--

#### Returns

WM\_SUCCESS if the contents pointed to by *network* have been added to the WLAN Connection Manager.  
 -WM\_E\_INVALID if *network* is NULL or the network name is not unique or the network name length is not valid or network security is [WLAN\\_SECURITY\\_WPA3\\_SAE](#) but Management Frame Protection Capable is not enabled. in [wlan\\_network\\_security](#) field. if network security type is [WLAN\\_SECURITY\\_WPA](#) or [WLAN\\_SECURITY\\_WPA2](#) or [WLAN\\_SECURITY\\_WPA\\_WPA2\\_MIXED](#), but the passphrase length is less than 8 or greater than 63, or the psk length equal to 64 but not hexadecimal digits. if network security type is [WLAN\\_SECURITY\\_WPA3\\_SAE](#), but the password length is less than 8 or greater than 255. if network security type is [WLAN\\_SECURITY\\_WEP\\_OPEN](#) or [WLAN\\_SECURITY\\_WEP\\_SHARED](#).

-WM\_E\_NOMEM if there was no room to add the network.

WLAN\_ERROR\_STATE if the WLAN Connection Manager was running and not in the [WLAN\\_DISCONNECTED](#), [WLAN\\_ASSOCIATED](#) or [WLAN\\_CONNECTED](#) state.

#### 5.7.3.7 wlan\_remove\_network()

```
int wlan_remove_network (
    const char * name )
```

Remove a network profile from the list of known networks.

This function removes a network (identified by its name) from the WLAN Connection Manager, disconnecting from that network if connected.

##### Note

This function is asynchronous if it is called while the WLAN Connection Manager is running and connected to the network to be removed. In that case, the WLAN Connection Manager will disconnect from the network and generate an event with reason [WLAN\\_REASON\\_USER\\_DISCONNECT](#). This function is synchronous otherwise.

This API can be used to remove profiles for station or micro-AP interfaces. Station network will not be removed if it is in [WLAN\\_CONNECTED](#) state and uAP network will not be removed if it is in [WLAN\\_UAP\\_STARTED](#) state.

##### Parameters

in	<i>name</i>	A pointer to the string representing the name of the network to remove.
----	-------------	---

##### Returns

WM\_SUCCESS if the network named *name* was removed from the WLAN Connection Manager successfully. Otherwise, the network is not removed.

WLAN\_ERROR\_STATE if the WLAN Connection Manager was running and the station interface was not in the [WLAN\\_DISCONNECTED](#) state.

-WM\_E\_INVALID if *name* is NULL or the network was not found in the list of known networks.

-WM\_FAIL if an internal error occurred while trying to disconnect from the network specified for removal.

#### 5.7.3.8 wlan\_connect()

```
int wlan_connect (
    char * name )
```

Connect to a wireless network (Access Point).

When this function is called, WLAN Connection Manager starts connection attempts to the network specified by *name*. The connection result will be notified asynchronously to the WLCMGR callback when the connection process has completed.

When connecting to a network, the event refers to the connection attempt to that network.

Calling this function when the station interface is in the [WLAN\\_DISCONNECTED](#) state will, if successful, cause the interface to transition into the [WLAN\\_CONNECTING](#) state. If the connection attempt succeeds, the station interface will transition to the [WLAN\\_CONNECTED](#) state, otherwise it will return to the [WLAN\\_DISCONNECTED](#) state. If this function is called while the station interface is in the [WLAN\\_CONNECTING](#) or [WLAN\\_CONNECTED](#) state, the WLAN Connection Manager will first cancel its connection attempt or disconnect from the network, respectively, and generate an event with reason [WLAN\\_REASON\\_USER\\_DISCONNECT](#). This will be followed by a second event that reports the result of the new connection attempt.

If the connection attempt was successful the WLCMGR callback is notified with the event [WLAN\\_REASON\\_SUCCESS](#), while if the connection attempt fails then either of the events, [WLAN\\_REASON\\_NETWORK\\_NOT\\_FOUND](#), [WLAN\\_REASON\\_NETWORK\\_AUTH\\_FAILED](#), [WLAN\\_REASON\\_CONNECT\\_FAILED](#) or [WLAN\\_REASON\\_ADDRESS\\_FAILED](#) are reported as appropriate.

#### Parameters

in	<i>name</i>	A pointer to a string representing the name of the network to connect to.
----	-------------	---

#### Returns

- WM\_SUCCESS if a connection attempt was started successfully
- WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running.
- WM\_E\_INVALID if there are no known networks to connect to or the network specified by *name* is not in the list of known networks or network *name* is NULL.
- WM\_FAIL if an internal error has occurred.

#### 5.7.3.9 wlan\_disconnect()

```
int wlan_disconnect (
    void )
```

Disconnect from the current wireless network (Access Point).

When this function is called, the WLAN Connection Manager attempts to disconnect the station interface from its currently connected network (or cancel an in-progress connection attempt) and return to the [WLAN\\_DISCONNECTED](#) state. Calling this function has no effect if the station interface is already disconnected.

#### Note

This is an asynchronous function and successful disconnection will be notified using the [WLAN\\_REASON\\_USER\\_DISCONNECT](#).

#### Returns

- WM\_SUCCESS if successful
- WLAN\_ERROR\_STATE otherwise

### 5.7.3.10 wlan\_start\_network()

```
int wlan_start_network (
    const char * name )
```

Start a wireless network (Access Point).

When this function is called, the WLAN Connection Manager starts the network specified by *name*. The network with the specified *name* must be first added using [wlan\\_add\\_network](#) and must be a micro-AP network with a valid SSID.

#### Note

The WLCMGR callback is asynchronously notified of the status. On success, the event [WLAN\\_REASON\\_UAP\\_SUCCESS](#) is reported, while on failure, the event [WLAN\\_REASON\\_UAP\\_START\\_FAILED](#) is reported.

#### Parameters

in	<i>name</i>	A pointer to string representing the name of the network to connect to.
----	-------------	---

#### Returns

WM\_SUCCESS if successful.  
 WLAN\_ERROR\_STATE if in power save state or uAP already running.  
 -WM\_E\_INVALID if *name* was NULL or the network *name* was not found or it not have a specified SSID.

### 5.7.3.11 wlan\_stop\_network()

```
int wlan_stop_network (
    const char * name )
```

Stop a wireless network (Access Point).

When this function is called, the WLAN Connection Manager stops the network specified by *name*. The specified network must be a valid micro-AP network that has already been started.

#### Note

The WLCMGR callback is asynchronously notified of the status. On success, the event [WLAN\\_REASON\\_UAP\\_STOPPED](#) is reported, while on failure, the event [WLAN\\_REASON\\_UAP\\_STOP\\_FAILED](#) is reported.

#### Parameters

in	<i>name</i>	A pointer to a string representing the name of the network to stop.
----	-------------	---



## Returns

WM\_SUCCESS if successful.  
 WLAN\_ERROR\_STATE if uAP is in power save state.  
 -WM\_E\_INVALID if *name* was NULL or the network *name* was not found or that the network *name* is not a micro-AP network or it is a micro-AP network but does not have a specified SSID.

## 5.7.3.12 wlan\_get\_mac\_address()

```
int wlan_get_mac_address (
    unsigned char * dest )
```

Retrieve the wireless MAC address of station/micro-AP interface.

This function copies the MAC address of the wireless interface to the 6-byte array pointed to by *dest*. In the event of an error, nothing is copied to *dest*.

## Parameters

out	<i>dest</i>	A pointer to a 6-byte array where the MAC address will be copied.
-----	-------------	---

## Returns

WM\_SUCCESS if the MAC address was copied.  
 -WM\_E\_INVALID if *dest* is NULL.

## 5.7.3.13 wlan\_get\_address()

```
int wlan_get_address (
    struct wlan_ip_config * addr )
```

Retrieve the IP address configuration of the station interface.

This function retrieves the IP address configuration of the station interface and copies it to the memory location pointed to by *addr*.

## Note

This function may only be called when the station interface is in the [WLAN\\_CONNECTED](#) state.

## Parameters

out	<i>addr</i>	A pointer to the <a href="#">wlan_ip_config</a> .
-----	-------------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *addr* is NULL.  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or was not in the [WLAN\\_CONNECTED](#) state.  
 -WM\_FAIL if an internal error occurred when retrieving IP address information from the TCP stack.

**5.7.3.14 wlan\_get\_uap\_address()**

```
int wlan_get_uap_address (
    struct wlan_ip_config * addr )
```

Retrieve the IP address of micro-AP interface.

This function retrieves the current IP address configuration of micro-AP and copies it to the memory location pointed to by *addr*.

**Note**

This function may only be called when the micro-AP interface is in the [WLAN\\_UAP\\_STARTED](#) state.

**Parameters**

out	<i>addr</i>	A pointer to the <a href="#">wlan_ip_config</a> .
-----	-------------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *addr* is NULL.  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or the micro-AP interface was not in the [WLAN\\_UAP\\_STARTED](#) state.  
 -WM\_FAIL if an internal error occurred when retrieving IP address information from the TCP stack.

**5.7.3.15 wlan\_get\_uap\_channel()**

```
int wlan_get_uap_channel (
    int * channel )
```

Retrieve the channel of micro-AP interface.

This function retrieves the channel number of micro-AP and copies it to the memory location pointed to by *channel*.

**Note**

This function may only be called when the micro-AP interface is in the [WLAN\\_UAP\\_STARTED](#) state.

## Parameters

out	<i>channel</i>	A pointer to variable that stores channel number.
-----	----------------	---

## Returns

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *channel* is NULL.  
 -WM\_FAIL if an internal error has occurred.

## 5.7.3.16 wlan\_get\_current\_network()

```
int wlan_get_current_network (
    struct wlan_network * network )
```

Retrieve the current network configuration of station interface.

This function retrieves the current network configuration of station interface when the station interface is in the [WLAN\\_CONNECTED](#) state.

## Parameters

out	<i>network</i>	A pointer to the <a href="#">wlan_network</a> .
-----	----------------	---

## Returns

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *network* is NULL.  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or not in the [WLAN\\_CONNECTED](#) state.

## 5.7.3.17 wlan\_get\_current\_uap\_network()

```
int wlan_get_current_uap_network (
    struct wlan_network * network )
```

Retrieve the current network configuration of micro-AP interface.

This function retrieves the current network configuration of micro-AP interface when the micro-AP interface is in the [WLAN\\_UAP\\_STARTED](#) state.

## Parameters

out	<i>network</i>	A pointer to the <a href="#">wlan_network</a> .
-----	----------------	---

**Returns**

WM\_SUCCESS if successful.  
-WM\_E\_INVALID if *network* is NULL.  
WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or not in the [WLAN\\_UAP\\_STARTED](#) state.

**5.7.3.18 is\_uap\_started()**

```
bool is_uap_started (
    void )
```

Retrieve the status information of the micro-AP interface.

**Returns**

TRUE if micro-AP interface is in [WLAN\\_UAP\\_STARTED](#) state.  
FALSE otherwise.

**5.7.3.19 is\_sta\_connected()**

```
bool is_sta_connected (
    void )
```

Retrieve the status information of the station interface.

**Returns**

TRUE if station interface is in [WLAN\\_CONNECTED](#) state.  
FALSE otherwise.

**5.7.3.20 is\_sta\_ipv4\_connected()**

```
bool is_sta_ipv4_connected (
    void )
```

Retrieve the status information of the ipv4 network of station interface.

**Returns**

TRUE if ipv4 network of station interface is in [WLAN\\_CONNECTED](#) state.  
FALSE otherwise.

## 5.7.3.21 wlan\_get\_network()

```
int wlan_get_network (
    unsigned int index,
    struct wlan_network * network )
```

Retrieve the information about a known network using *index*.

This function retrieves the contents of a network at *index* in the list of known networks maintained by the WLAN Connection Manager and copies it to the location pointed to by *network*.

## Note

[wlan\\_get\\_network\\_count\(\)](#) may be used to retrieve the number of known networks. [wlan\\_get\\_network\(\)](#) may be used to retrieve information about networks at *index* 0 to one minus the number of networks.

This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

## Parameters

in	<i>index</i>	The index of the network to retrieve.
out	<i>network</i>	A pointer to the <a href="#">wlan_network</a> where the network configuration for the network at <i>index</i> will be copied.

## Returns

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *network* is NULL or *index* is out of range.

## 5.7.3.22 wlan\_get\_network\_byname()

```
int wlan_get_network_byname (
    char * name,
    struct wlan_network * network )
```

Retrieve information about a known network using *name*.

This function retrieves the contents of a named network in the list of known networks maintained by the WLAN Connection Manager and copies it to the location pointed to by *network*.

## Note

This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

## Parameters

in	<i>name</i>	The name of the network to retrieve.
out	<i>network</i>	A pointer to the <a href="#">wlan_network</a> where the network configuration for the network having name as <i>name</i> will be copied.

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *network* is NULL or *name* is NULL.

**5.7.3.23 wlan\_get\_network\_count()**

```
int wlan_get_network_count (
    unsigned int * count )
```

Retrieve the number of networks known to the WLAN Connection Manager.

This function retrieves the number of known networks in the list maintained by the WLAN Connection Manager and copies it to *count*.

**Note**

This function may be called regardless of whether the WLAN Connection Manager is running. Calls to this function are synchronous.

**Parameters**

out	<i>count</i>	A pointer to the memory location where the number of networks will be copied.
-----	--------------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *count* is NULL.

**5.7.3.24 wlan\_get\_connection\_state()**

```
int wlan_get_connection_state (
    enum wlan_connection_state * state )
```

Retrieve the connection state of station interface.

This function retrieves the connection state of station interface, which is one of [WLAN\\_DISCONNECTED](#), [WLAN\\_CONNECTING](#), [WLAN\\_ASSOCIATED](#) or [WLAN\\_CONNECTED](#).

**Parameters**

out	<i>state</i>	A pointer to the <a href="#">wlan_connection_state</a> where the current connection state will be copied.
-----	--------------	---

## Returns

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *state* is NULL  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running.

## 5.7.3.25 wlan\_get\_uap\_connection\_state()

```
int wlan_get_uap_connection_state (
    enum wlan_connection_state * state )
```

Retrieve the connection state of micro-AP interface.

This function retrieves the connection state of micro-AP interface, which is one of [WLAN\\_UAP\\_STARTED](#), or [WLAN\\_UAP\\_STOPPED](#).

## Parameters

out	<i>state</i>	A pointer to the <a href="#">wlan_connection_state</a> where the current connection state will be copied.
-----	--------------	---

## Returns

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *state* is NULL  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running.

## 5.7.3.26 wlan\_scan()

```
int wlan_scan (
    int(*) (unsigned int count) cb )
```

Scan for wireless networks.

When this function is called, the WLAN Connection Manager starts scan for wireless networks. On completion of the scan the WLAN Connection Manager will call the specified callback function *cb*. The callback function can then retrieve the scan results by using the [wlan\\_get\\_scan\\_result\(\)](#) function.

## Note

This function may only be called when the station interface is in the [WLAN\\_DISCONNECTED](#) or [WLAN\\_CONNECTING](#) state. Scanning is disabled in the [WLAN\\_CONNECTED](#) state.  
 This function will block until it can issue a scan request if called while another scan is in progress.

## Parameters

in	<i>cb</i>	A pointer to the function that will be called to handle scan results when they are available.
----	-----------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_NOMEM if failed to allocated memory for [wlan\\_scan\\_params\\_v2\\_t](#) structure.  
 -WM\_E\_INVALID if *cb* scan result callack function pointer is NULL.  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or not in the [WLAN\\_DISCONNECTED](#) or [WLAN\\_CONNECTED](#) states.  
 -WM\_FAIL if an internal error has occurred and the system is unable to scan.

**5.7.3.27 wlan\_scan\_with\_opt()**

```
int wlan_scan_with_opt (
    wlan_scan_params_v2_t t_wlan_scan_param )
```

Scan for wireless networks using options provided.

When this function is called, the WLAN Connection Manager starts scan for wireless networks. On completion of the scan the WLAN Connection Manager will call the specified callback function *cb*. The callback function can then retrieve the scan results by using the [wlan\\_get\\_scan\\_result\(\)](#) function.

**Note**

This function may only be called when the station interface is in the [WLAN\\_DISCONNECTED](#) or [WLAN\\_CONNECTED](#) state. Scanning is disabled in the [WLAN\\_CONNECTING](#) state.  
 This function will block until it can issue a scan request if called while another scan is in progress.

**Parameters**

in	<i>wlan_scan_param</i>	A <a href="#">wlan_scan_params_v2_t</a> structure holding a pointer to function that will be called to handle scan results when they are available, SSID of a wireless network, BSSID of a wireless network, number of channels with scan type information and number of probes.
----	------------------------	--

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_NOMEM if failed to allocated memory for [wlan\\_scan\\_params\\_v2\\_t](#) structure.  
 -WM\_E\_INVALID if *cb* scan result callack function pointer is NULL.  
 WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running or not in the [WLAN\\_DISCONNECTED](#) or [WLAN\\_CONNECTED](#) states.  
 -WM\_FAIL if an internal error has occurred and the system is unable to scan.

**5.7.3.28 wlan\_get\_scan\_result()**

```
int wlan_get_scan_result (
    unsigned int index,
    struct wlan_scan_result * res )
```



Retrieve a scan result.

This function may be called to retrieve scan results when the WLAN Connection Manager has finished scanning. It must be called from within the scan result callback (see [wlan\\_scan\(\)](#)) as scan results are valid only in that context. The callback argument 'count' provides the number of scan results that may be retrieved and [wlan\\_get\\_scan\\_result\(\)](#) may be used to retrieve scan results at *index* 0 through that number.

#### Note

This function may only be called in the context of the scan results callback.  
Calls to this function are synchronous.

#### Parameters

in	<i>index</i>	The scan result to retrieve.
out	<i>res</i>	A pointer to the <a href="#">wlan_scan_result</a> where the scan result information will be copied.

#### Returns

WM\_SUCCESS if successful.  
-WM\_E\_INVALID if *res* is NULL  
WLAN\_ERROR\_STATE if the WLAN Connection Manager was not running  
-WM\_FAIL if the scan result at *index* could not be retrieved (that is, *index* is out of range).

#### 5.7.3.29 wlan\_set\_ed\_mac\_mode()

```
int wlan_set_ed_mac_mode (
    wlan_ed_mac_ctrl_t wlan_ed_mac_ctrl )
```

Configure ED MAC mode in Wireless Firmware.

#### Note

When ed mac mode is enabled, Wireless Firmware will behave following way:

when background noise had reached -70dB or above, WiFi chipset/module should hold data transmitting until condition is removed. It is applicable for both 5GHz and 2.4GHz bands.

#### Parameters

in	<i>wlan_ed_mac_ctrl</i>	Struct with following parameters ed_ctrl_2g 0 - disable EU adaptivity for 2.4GHz band 1 - enable EU adaptivity for 2.4GHz band
----	-------------------------	--

ed\_offset\_2g 0 - Default Energy Detect threshold (Default: 0x9) offset value range: 0x80 to 0x7F

#### Note

If 5GH enabled then add following parameters

```

ed_ctrl_5g          0 - disable EU adaptivity for 5GHz band
                   1 - enable EU adaptivity for 5GHz band

ed_offset_5g        0 - Default Energy Detect threshold(Default: 0xC)
                   offset value range: 0x80 to 0x7F

```

**Returns**

WM\_SUCCESS if the call was successful.  
 -WM\_FAIL if failed.

**5.7.3.30 wlan\_get\_ed\_mac\_mode()**

```

int wlan_get_ed_mac_mode (
    wlan_ed_mac_ctrl_t * wlan_ed_mac_ctrl )

```

This API can be used to get current ED MAC MODE configuration.

**Parameters**

out	<i>wlan_ed_mac_ctrl</i>	A pointer to <a href="#">wlan_ed_mac_ctrl_t</a> with parameters mentioned in above set API.
-----	-------------------------	---

**Returns**

WM\_SUCCESS if the call was successful.  
 -WM\_FAIL if failed.

**5.7.3.31 wlan\_set\_cal\_data()**

```

void wlan_set_cal_data (
    uint8_t * cal_data,
    unsigned int cal_data_size )

```

Set wireless calibration data in WLAN firmware.

This function may be called to set wireless calibration data in firmware. This should be call before [wlan\\_init\(\)](#) function.

**Parameters**

in	<i>cal_data</i>	The calibration data buffer
in	<i>cal_data_size</i>	Size of calibration data buffer.

## 5.7.3.32 wlan\_set\_mac\_addr()

```
void wlan_set_mac_addr (
    uint8_t * mac )
```

Set wireless MAC Address in WLAN firmware.

This function may be called to set wireless MAC Address in firmware. This should be call before [wlan\\_init\(\)](#) function.

## Parameters

in	mac	The MAC Address in 6 byte array format like uint8_t mac[] = { 0x00, 0x50, 0x43, 0x21, 0x19, 0x6E};
----	-----	--

## 5.7.3.33 wlan\_configure\_listen\_interval()

```
void wlan_configure_listen_interval (
    int listen_interval )
```

Configure Listen interval of IEEE power save mode.

## Note

**Delivery Traffic Indication Message (DTIM):** It is a concept in 802.11 It is a time duration after which AP will send out buffered BROADCAST / MULTICAST data and stations connected to the AP should wakeup to take this broadcast / multicast data.

**Traffic Indication Map (TIM):** It is a bitmap which the AP sends with each beacon. The bitmap has one bit each for a station connected to AP.

Each station is recognized by an Association Id (AID). If AID is say 1 bit number 1 is set in the bitmap if unicast data is present with AP in its buffer for station with AID = 1 Ideally AP does not buffer any unicast data it just sends unicast data to the station on every beacon when station is not sleeping.

When broadcast data / multicast data is to be send AP sets bit 0 of TIM indicating broadcast / multicast.

The occurrence of DTIM is defined by AP.

Each beacon has a number indicating period at which DTIM occurs.

The number is expressed in terms of number of beacons.

This period is called DTIM Period / DTIM interval.

For example:

If AP has DTIM period = 3 the stations connected to AP have to wake up (if they are sleeping) to receive broadcast / multicast data on every third beacon.

Generic:

When DTIM period is X AP buffers broadcast data / multicast data for X beacons. Then it transmits the data no matter whether station is awake or not.

Listen interval:

This is time interval on station side which indicates when station will be awake to listen i.e. accept data.

Long listen interval:

It comes into picture when station sleeps (IEEEPS) and it does not want to wake up on every DTIM So station is not worried about broadcast data/multicast data in this case.

This should be a design decision what should be chosen Firmware suggests values which are about 3 times DTIM at the max to gain optimal usage and reliability.

In the IEEEPS power save mode, the WiFi firmware goes to sleep and periodically wakes up to check if the AP has any pending packets for it. A longer listen interval implies that the WiFi card stays in power save for a longer duration at the cost of additional delays while receiving data. Please note that choosing incorrect value for listen interval will causes poor response from device during data transfer. Actual listen interval selected by

firmware is equal to closest DTIM.

For e.g.:-

AP beacon period : 100 ms

AP DTIM period : 2

Application request value: 500ms

Actual listen interval = 400ms (This is the closest DTIM). Actual listen interval set will be a multiple of DTIM closest to but lower than the value provided by the application.

This API can be called before/after association. The configured listen interval will be used in subsequent association attempt.

#### Parameters

in	<i>listen_interval</i>	Listen interval as below 0 : Unchanged, -1 : Disable, 1-49: Value in beacon intervals, >= 50: Value in TUs
----	------------------------	--

#### 5.7.3.34 wlan\_configure\_null\_pkt\_interval()

```
void wlan_configure_null_pkt_interval (
    int time_in_secs )
```

Configure Null packet interval of IEEE power save mode.

#### Note

In IEEEPS station sends a NULL packet to AP to indicate that the station is alive and AP should not kick it off. If null packet is not send some APs may disconnect station which might lead to a loss of connectivity. The time is specified in seconds. Default value is 30 seconds.

This API should be called before configuring IEEEPS

#### Parameters

in	<i>time_in_secs</i>	: -1 Disables null packet transmission, 0 Null packet interval is unchanged, n Null packet interval in seconds.
----	---------------------	---

#### 5.7.3.35 wlan\_set\_antcfg()

```
int wlan_set_antcfg (
    uint32_t ant,
    uint16_t evaluate_time )
```

This API can be used to set the mode of Tx/Rx antenna. If SAD is enabled, this API can also used to set SAD antenna evaluate time interval(antenna mode must be antenna diversity when set SAD evaluate time interval).

## Parameters

in	<i>ant</i>	Antenna valid values are 1, 2 and 65535 1 : Tx/Rx antenna 1 2 : Tx/Rx antenna 2 0xFFFF: Tx/Rx antenna diversity
in	<i>evaluate_time</i>	SAD evaluate time interval, default value is 6s(0x1770).

## Returns

WM\_SUCCESS if successful.  
WLAN\_ERROR\_STATE if unsuccessful.

## 5.7.3.36 wlan\_get\_antcfg()

```
int wlan_get_antcfg (
    uint32_t * ant,
    uint16_t * evaluate_time )
```

This API can be used to get the mode of Tx/Rx antenna. If SAD is enabled, this API can also used to get SAD antenna evaluate time interval(antenna mode must be antenna diversity when set SAD evaluate time interval).

## Parameters

out	<i>ant</i>	pointer to antenna variable.
out	<i>evaluate_time</i>	pointer to evaluate_time variable for SAD.

## Returns

WM\_SUCCESS if successful.  
WLAN\_ERROR\_STATE if unsuccessful.

## 5.7.3.37 wlan\_get\_firmware\_version\_ext()

```
char* wlan_get_firmware_version_ext (
    void )
```

Get the wifi firmware version extension string.

## Note

This API does not allocate memory for pointer. It just returns pointer of WLCMGR internal static buffer. So no need to free the pointer by caller.

## Returns

wifi firmware version extension string pointer stored in WLCMGR

### 5.7.3.38 wlan\_version\_extended()

```
void wlan_version_extended (
    void )
```

Use this API to print wlan driver and firmware extended version.

### 5.7.3.39 wlan\_get\_tsf()

```
int wlan_get_tsf (
    uint32_t * tsf_high,
    uint32_t * tsf_low )
```

Use this API to get the TSF from Wi-Fi firmware.

#### Parameters

in	<i>tsf_high</i>	Pointer to store TSF higher 32bits.
in	<i>tsf_low</i>	Pointer to store TSF lower 32bits.

#### Returns

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

### 5.7.3.40 wlan\_ieee80211\_on()

```
int wlan_ieee80211_on (
    unsigned int wakeup_conditions )
```

Enable IEEE80211 with Host Sleep Configuration

When enabled, it opportunistically puts the wireless card into IEEE80211 mode. Before putting the Wireless card in power save this also sets the hostsleep configuration on the card as specified. This makes the card generate a wakeup for the processor if any of the wakeup conditions are met.

#### Parameters

in	<i>wakeup_conditions</i>	conditions to wake the host. This should be a logical OR of the conditions in <a href="#">wlan_wakeup_event_t</a> . Typically devices would want to wake up on <a href="#">WAKE_ON_ALL_BROADCAST</a> , <a href="#">WAKE_ON_UNICAST</a> , <a href="#">WAKE_ON_MAC_EVENT</a> , <a href="#">WAKE_ON_MULTICAST</a> , <a href="#">WAKE_ON_ARP_BROADCAST</a> , <a href="#">WAKE_ON_MGMT_FRAME</a> wnm_set 1: wnm is set. 0: wnm is not set. wnm_sleep_time: wnm sleep interval.(number of dtims)
----	--------------------------	--

**Returns**

WM\_SUCCESS if the call was successful.  
WLAN\_ERROR\_STATE if the call was made in a state where such an operation is illegal.  
-WM\_FAIL otherwise.

**Note**

This function should be used after station gets connected to a network.

**5.7.3.41 wlan\_ieeepps\_off()**

```
int wlan_ieeepps_off (  
    void )
```

Turn off IEEE Power Save mode.

**Note**

This call is asynchronous. The system will exit the power-save mode only when all requisite conditions are met.

**Returns**

WM\_SUCCESS if the call was successful.  
WLAN\_ERROR\_STATE if the call was made in a state where such an operation is illegal.  
-WM\_FAIL otherwise.

**5.7.3.42 wlan\_deepsleeps\_on()**

```
int wlan_deepsleeps_on (  
    void )
```

Turn on Deep Sleep Power Save mode.

**Note**

This call is asynchronous. The system will enter the power-save mode only when all requisite conditions are met. For example, wlan should be disconnected for this to work.

**Returns**

WM\_SUCCESS if the call was successful.  
WLAN\_ERROR\_STATE if the call was made in a state where such an operation is illegal.

#### 5.7.3.43 wlan\_deepsleepps\_off()

```
int wlan_deepsleepps_off (
    void )
```

Turn off Deep Sleep Power Save mode.

##### Note

This call is asynchronous. The system will exit the power-save mode only when all requisite conditions are met.

##### Returns

WM\_SUCCESS if the call was successful.

WLAN\_ERROR\_STATE if the call was made in a state where such an operation is illegal.

#### 5.7.3.44 wlan\_get\_beacon\_period()

```
uint16_t wlan_get_beacon_period (
    void )
```

Use this API to get the beacon period of associated BSS.

##### Returns

beacon\_period if operation is successful.

0 if command fails.

#### 5.7.3.45 wlan\_get\_dtim\_period()

```
uint8_t wlan_get_dtim_period (
    void )
```

Use this API to get the dtim period of associated BSS.

##### Returns

dtim\_period if operation is successful.

0 if DTIM IE Is not found in AP's Probe response.

##### Note

This API should not be called from WLAN event handler registered by application during [wlan\\_start](#).

#### 5.7.3.46 wlan\_get\_data\_rate()

```
int wlan_get_data_rate (
    wlan_ds_rate * ds_rate )
```

Use this API to get the current tx and rx rates along with bandwidth and guard interval information if rate is 11N.



## Parameters

in	<i>ds_rate</i>	A pointer to structure which will have tx, rx rate information along with bandwidth and guard interval information.
----	----------------	---

## Note

If rate is greater than 11 then it is 11N rate and from 12 MCS0 rate starts. The bandwidth mapping is like value 0 is for 20MHz, 1 is 40MHz, 2 is for 80MHz. The guard interval value zero means Long otherwise Short.

## Returns

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

## 5.7.3.47 wlan\_set\_pmfcfg()

```
int wlan_set_pmfcfg (
    uint8_t mfpc,
    uint8_t mfpr )
```

Use this API to set the set management frame protection parameters.

## Parameters

in	<i>mfpc</i>	Management Frame Protection Capable (MFPC) 1: Management Frame Protection Capable 0: Management Frame Protection not Capable
in	<i>mfpr</i>	Management Frame Protection Required (MFPR) 1: Management Frame Protection Required 0: Management Frame Protection Optional

## Note

Default setting is PMF not capable. mfpc = 0, mfpr = 1 is an invalid combination

## Returns

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

## 5.7.3.48 wlan\_get\_pmfcfg()

```
int wlan_get_pmfcfg (
    uint8_t * mfpc,
    uint8_t * mfpr )
```

Use this API to get the set management frame protection parameters.

## Parameters

out	<i>mfp</i>	Management Frame Protection Capable (MFPC) 1: Management Frame Protection Capable 0: Management Frame Protection not Capable
out	<i>mfp</i>	Management Frame Protection Required (MFPR) 1: Management Frame Protection Required 0: Management Frame Protection Optional

## Returns

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

## 5.7.3.49 wlan\_set\_packet\_filters()

```
int wlan_set_packet_filters (
    wlanflt_cfg_t * flt_cfg )
```

Use this API to set packet filters in Wi-Fi firmware.

Confidential

## Parameters

## Parameters

in	<i>flt_cfg</i>	<p>A pointer to structure which holds the the packet filters in same way as given below.</p> <p>MEF Configuration command</p> <pre> mefcfg={ Criteria: bit0-broadcast, bit1-unicast, bit3-multicast Criteria=2 Unicast frames are received during hostsleepmode NumEntries=1 Number of activated MEF entries mef_entry_0: example filters to match TCP destination port 80 send by 192.168.0.88 pkt or magic pkt. mef_entry_0={ mode: bit0-hostsleep mode, bit1-non hostsleep mode mode=1 HostSleep mode action: 0-discard and not wake host, 1-discard and wake host 3-allow and wake host action=3 Allow and Wake host filter_num=3 Number of filter RPN only support "&amp;&amp;" and "  " operator,space can not be removed between operator. RPN=Filter_0 &amp;&amp; Filter_1    Filter_2 Byte comparison filter's type is 0x41,Decimal comparison filter's type is 0x42, Bit comparison filter's type is 0x43 Filter_0 is decimal comparison filter, it always with type=0x42 Decimal filter always has type, pattern, offset, numbyte 4 field Filter_0 will match rx pkt with TCP destination port 80 Filter_0={ type=0x42 decimal comparison filter pattern=80 80 is the decimal constant to be compared offset=44 44 is the byte offset of the field in RX pkt to be compare numbyte=2 2 is the number of bytes of the field } Filter_1 is Byte comparison filter, it always with type=0x41 Byte filter always has type, byte, repeat, offset 4 filed Filter_1 will match rx pkt send by IP address 192.168.0.88 Filter_1={ type=0x41 Byte comparison filter repeat=1 1 copies of 'c0:a8:00:58' byte=c0:a8:00:58 'c0:a8:00:58' is the byte sequence constant with each byte in hex format, with ':' as delimiter between two byte. offset=34 34 is the byte offset of the equal length field of rx'd pkt. } Filter_2 is Magic packet, it will looking for 16 contiguous copies of '00:50:43:20:01:02' from the rx pkt's offset 14 Filter_2={ type=0x41 Byte comparison filter repeat=16 16 copies of '00:50:43:20:01:02' byte=00:50:43:20:01:02 # '00:50:43:20:01:02' is the byte sequence constant offset=14 14 is the byte offset of the equal length field of rx'd pkt. } } } Above filters can be set by filling values in following way in wlan_flt_cfg_t structure. wlan_flt_cfg_t flt_cfg; uint8_t byte_seq1[] = {0xc0, 0xa8, 0x00, 0x58}; uint8_t byte_seq2[] = {0x00, 0x50, 0x43, 0x20, 0x01, 0x02};  memset(&amp;flt_cfg, 0, sizeof(wlan_flt_cfg_t));  flt_cfg.criteria = 2; flt_cfg.nentries = 1;  flt_cfg.mef_entry.mode = 1; flt_cfg.mef_entry.action = 3;  flt_cfg.mef_entry.filter_num = 3; </pre>
		<p>Proprietary Information. Copyright © 2020 NXP</p>

---

**Parameters**

---

**Returns**

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

**5.7.3.50 wlan\_set\_auto\_arp()**

```
int wlan_set_auto_arp (  
    void )
```

Use this API to enable ARP Offload in Wi-Fi firmware

**Returns**

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

**5.7.3.51 wlan\_send\_host\_sleep()**

```
int wlan_send_host_sleep (  
    uint32_t wakeup_condition )
```

Use this API to configure host sleep params in Wi-Fi firmware.

**Returns**

WM\_SUCCESS if operation is successful.  
-WM\_FAIL if command fails.

**5.7.3.52 wlan\_get\_current\_bssid()**

```
int wlan_get_current_bssid (  
    uint8_t * bssid )
```

Use this API to get the BSSID of associated BSS.

**Parameters**

in	<i>bssid</i>	A pointer to array to store the BSSID.
----	--------------	--

**Returns**

WM\_SUCCESS if operation is successful.  
 -WM\_FAIL if command fails.

**5.7.3.53 wlan\_get\_current\_channel()**

```
uint8_t wlan_get_current_channel (
    void )
```

Use this API to get the channel number of associated BSS.

**Returns**

channel number if operation is successful.  
 0 if command fails.

**5.7.3.54 wlan\_get\_ps\_mode()**

```
int wlan_get_ps_mode (
    enum wlan_ps_mode * ps_mode )
```

Get station interface power save mode.

**Parameters**

out	<i>ps_mode</i>	A pointer to <a href="#">wlan_ps_mode</a> where station interface power save mode will be stored.
-----	----------------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_E\_INVALID if *ps\_mode* was NULL.

**5.7.3.55 wlan\_wlcmgr\_send\_msg()**

```
int wlan_wlcmgr_send_msg (
    enum wifi_event event,
    enum wifi_event_reason reason,
    void * data )
```

Send message to WLAN Connection Manager thread.

## Parameters

in	<i>event</i>	An event from <a href="#">wifi_event</a> .
in	<i>reason</i>	A reason code.
in	<i>data</i>	A pointer to data buffer associated with event.

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if failed.

## 5.7.3.56 wlan\_wfa\_basic\_cli\_init()

```
int wlan_wfa_basic_cli_init (  
    void )
```

Register WFA basic WLAN CLI commands

This function registers basic WLAN CLI commands like showing version information, MAC address

## Note

This function can only be called by the application after [wlan\\_init\(\)](#) called.

## Returns

WLAN\_ERROR\_NONE if the CLI commands were registered or  
WLAN\_ERROR\_ACTION if they were not registered (for example if this function was called while the CLI commands were already registered).

## 5.7.3.57 wlan\_basic\_cli\_init()

```
int wlan_basic_cli_init (  
    void )
```

Register basic WLAN CLI commands

This function registers basic WLAN CLI commands like showing version information, MAC address

## Note

This function can only be called by the application after [wlan\\_init\(\)](#) called.

This function gets called by [wlan\\_cli\\_init\(\)](#), hence only one function out of these two functions should be called in the application.

## Returns

WLAN\_ERROR\_NONE if the CLI commands were registered or  
WLAN\_ERROR\_ACTION if they were not registered (for example if this function was called while the CLI commands were already registered).

### 5.7.3.58 wlan\_cli\_init()

```
int wlan_cli_init (  
    void )
```

Register WLAN CLI commands.

Try to register the WLAN CLI commands with the CLI subsystem. This function is available for the application for use.

#### Note

This function can only be called by the application after [wlan\\_init\(\)](#) called.

This function internally calls [wlan\\_basic\\_cli\\_init\(\)](#), hence only one function out of these two functions should be called in the application.

#### Returns

WM\_SUCCESS if the CLI commands were registered or

-WM\_FAIL if they were not (for example if this function was called while the CLI commands were already registered).

### 5.7.3.59 wlan\_enhanced\_cli\_init()

```
int wlan_enhanced_cli_init (  
    void )
```

Register WLAN enhanced CLI commands.

Register the WLAN enhanced CLI commands like set or get tx-power, tx-datarate, tx-modulation etc with the CLI subsystem.

#### Note

This function can only be called by the application after [wlan\\_init\(\)](#) called.

#### Returns

WM\_SUCCESS if the CLI commands were registered or

-WM\_FAIL if they were not (for example if this function was called while the CLI commands were already registered).



### 5.7.3.60 wlan\_get\_uap\_supported\_max\_clients()

```
unsigned int wlan_get_uap_supported_max_clients (  
    void )
```

Get maximum number of WLAN firmware supported stations that will be allowed to connect to the uAP.

#### Returns

Maximum number of WLAN firmware supported stations.

#### Note

Get operation is allowed in any uAP state.

### 5.7.3.61 wlan\_get\_uap\_max\_clients()

```
int wlan_get_uap_max_clients (  
    unsigned int * max_sta_num )
```

Get current maximum number of stations that will be allowed to connect to the uAP.

#### Parameters

out	<i>max_sta_num</i>	A pointer to variable where current maximum number of stations of uAP interface will be stored.
-----	--------------------	---

#### Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

#### Note

Get operation is allowed in any uAP state.

### 5.7.3.62 wlan\_set\_uap\_max\_clients()

```
int wlan_set_uap_max_clients (  
    unsigned int max_sta_num )
```

Set maximum number of stations that will be allowed to connect to the uAP.

## Parameters

in	<i>max_sta_num</i>	Number of maximum stations for uAP.
----	--------------------	-------------------------------------

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

## Note

Set operation is not allowed in [WLAN\\_UAP\\_STARTED](#) state.

## 5.7.3.63 wlan\_set\_htcapinfo()

```
int wlan_set_htcapinfo (
    unsigned int htcapinfo )
```

This API can be used to configure some of parameters in HTCAPInfo IE (such as Short GI, Channel BW, and Green field support)

## Parameters

in	<i>htcapinfo</i>	<p>This is a bitmap and should be used as following</p> <ul style="list-style-type: none"> <li>Bit 29: Green field enable/disable</li> <li>Bit 26: Rx STBC Support enable/disable. (As we support single spatial stream only 1 bit is used for Rx STBC)</li> <li>Bit 25: Tx STBC support enable/disable.</li> <li>Bit 24: Short GI in 40 Mhz enable/disable</li> <li>Bit 23: Short GI in 20 Mhz enable/disable</li> <li>Bit 22: Rx LDPC enable/disable</li> <li>Bit 17: 20/40 Mhz enable/disable.</li> <li>Bit 8: Enable/disable 40Mhz Intolerant bit in ht capinfo.</li> </ul> <p>0 will reset this bit and 1 will set this bit in htcapinfo attached in assoc request. All others are reserved and should be set to 0.</p>
----	------------------	--

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

## 5.7.3.64 wlan\_set\_httxcfg()

```
int wlan_set_httxcfg (
    unsigned short httxcfg )
```

This API can be used to configure various 11n specific configuration for transmit (such as Short GI, Channel BW and Green field support)

Confidential

## Parameters

in	<i>httxcfg</i>	<p>This is a bitmap and should be used as following</p> <p>Bit 15-10: Reserved set to 0</p> <p>Bit 9-8: Rx STBC set to 0x01</p> <p>BIT9 BIT8 Description</p> <p>0 0 No spatial streams</p> <p>0 1 One spatial streams supported</p> <p>1 0 Reserved</p> <p>1 1 Reserved</p> <p>Bit 7: STBC enable/disable</p> <p>Bit 6: Short GI in 40 Mhz enable/disable</p> <p>Bit 5: Short GI in 20 Mhz enable/disable</p> <p>Bit 4: Green field enable/disable</p> <p>Bit 3-2: Reserved set to 1</p> <p>Bit 1: 20/40 Mhz enable/disable.</p> <p>Bit 0: LDPC enable/disable</p> <p>When Bit 1 is set then firmware could transmit in 20Mhz or 40Mhz based on rate adaptation. When this bit is reset then firmware will only transmit in 20Mhz.</p>
----	----------------	--

## Returns

WM\_SUCCESS if successful.  
 -WM\_FAIL if unsuccessful.

## 5.7.3.65 wlan\_set\_txratecfg()

```
int wlan_set_txratecfg (
    wlan_ds_rate ds_rate )
```

This API can be used to set the transmit data rate.

## Note

The data rate can be set only after association.

## Parameters

in	<i>ds_rate</i>	<p>struct contains following fields sub_command It should be WIFI_DS_RATE_CFG and rate_cfg should have following parameters.</p> <p>rate_format - This parameter specifies the data rate format used in this command</p> <p>0: LG 1: HT 2: VHT 0xff: Auto</p> <p>index - This parameter specifies the rate or MCS index</p> <p>If rate_format is 0 (LG),</p> <p>0 1 Mbps 1 2 Mbps 2 5.5 Mbps 3 11 Mbps 4 6 Mbps 5 9 Mbps 6 12 Mbps 7 18 Mbps 8 24 Mbps 9 36 Mbps 10 48 Mbps 11 54 Mbps</p> <p>If rate_format is 1 (HT),</p> <p>0 MCS0 1 MCS1 2 MCS2 3 MCS3 4 MCS4 5 MCS5 6 MCS6 7 MCS7</p> <p>If STREAM_2X2</p> <p>8 MCS8 9 MCS9 10 MCS10 11 MCS11 12 MCS12 13 MCS13 14 MCS14 15 MCS15</p> <p>If rate_format is 2 (VHT),</p> <p>0 MCS0 1 MCS1 2 MCS2 3 MCS3 4 MCS4 5 MCS5 6 MCS6 7 MCS7 8 MCS8 9 MCS9</p> <p>nss - This parameter specifies the NSS. It is valid only for VHT</p> <p>If rate_format is 2 (VHT),</p> <p>1 NSS1 2 NSS2</p>
----	----------------	--

**Returns**

WM\_SUCCESS if successful.  
 -WM\_FAIL if unsuccessful.

**5.7.3.66 wlan\_get\_txratecfg()**

```
int wlan_get_txratecfg (
    wlan_ds_rate * ds_rate )
```

This API can be used to get the transmit data rate.

**Parameters**

in	ds_rate	A pointer to wlan_ds_rate where Tx Rate configuration will be stored.
----	---------	---

**Returns**

WM\_SUCCESS if successful.  
 -WM\_FAIL if unsuccessful.

**5.7.3.67 wlan\_get\_sta\_tx\_power()**

```
int wlan_get_sta_tx_power (
    t_u32 * power_level )
```

Get Station interface transmit power

**Parameters**

out	power_level	Transmit power level.
-----	-------------	-----------------------

**Returns**

WM\_SUCCESS if successful.  
 -WM\_FAIL if unsuccessful.

**5.7.3.68 wlan\_set\_sta\_tx\_power()**

```
int wlan_set_sta_tx_power (
    t_u32 power_level )
```

Set Station interface transmit power

## Parameters

in	<i>power_level</i>	Transmit power level.
----	--------------------	-----------------------

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

## 5.7.3.69 wlan\_set\_wwsm\_txpwrlimit()

```
int wlan_set_wwsm_txpwrlimit (
    void )
```

Set World Wide Safe Mode Tx Power Limits

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

## 5.7.3.70 wlan\_get\_mgmt\_ie()

```
int wlan_get_mgmt_ie (
    enum wlan_bss_type bss_type,
    IEEEtypes_ElementId_t index,
    void * buf,
    unsigned int * buf_len )
```

Get Management IE for given BSS type (interface) and index.

## Parameters

in	<i>bss_type</i>	BSS Type of interface.
in	<i>index</i>	IE index.
out	<i>buf</i>	Buffer to store requested IE data.
out	<i>buf_len</i>	To store length of IE data.

## Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

### 5.7.3.71 wlan\_set\_mgmt\_ie()

```
int wlan_set_mgmt_ie (
    enum wlan_bss_type bss_type,
    IEEEtypes_ElementId_t id,
    void * buf,
    unsigned int buf_len )
```

Set Management IE for given BSS type (interface) and index.

#### Parameters

in	<i>bss_type</i>	BSS Type of interface.
in	<i>id</i>	Type/ID of Management IE.
in	<i>buf</i>	Buffer containing IE data.
in	<i>buf_len</i>	Length of IE data.

#### Returns

IE index if successful.  
-WM\_FAIL if unsuccessful.

### 5.7.3.72 wlan\_clear\_mgmt\_ie()

```
int wlan_clear_mgmt_ie (
    enum wlan_bss_type bss_type,
    IEEEtypes_ElementId_t index )
```

Clear Management IE for given BSS type (interface) and index.

#### Parameters

in	<i>bss_type</i>	BSS Type of interface.
in	<i>index</i>	IE index.

#### Returns

WM\_SUCCESS if successful.  
-WM\_FAIL if unsuccessful.

### 5.7.3.73 wlan\_get\_11d\_enable\_status()

```
bool wlan_get_11d_enable_status (
    void )
```

Get current status of 11d support.



**Returns**

true if 11d support is enabled by application.  
false if not enabled.

**5.7.3.74 wlan\_get\_current\_signal\_strength()**

```
int wlan_get_current_signal_strength (
    short * rssi,
    int * snr )
```

Get current RSSI and Signal to Noise ratio from WLAN firmware.

**Parameters**

in	<i>rssi</i>	A pointer to variable to store current RSSI
in	<i>snr</i>	A pointer to variable to store current SNR.

**Returns**

WM\_SUCCESS if successful.

**5.7.3.75 wlan\_get\_average\_signal\_strength()**

```
int wlan_get_average_signal_strength (
    short * rssi,
    int * snr )
```

Get average RSSI and Signal to Noise ratio from WLAN firmware.

**Parameters**

in	<i>rssi</i>	A pointer to variable to store current RSSI
in	<i>snr</i>	A pointer to variable to store current SNR.

**Returns**

WM\_SUCCESS if successful.

**5.7.3.76 wlan\_remain\_on\_channel()**

```
int wlan_remain_on_channel (
    const enum wlan_bss_type bss_type,
```

```

const bool status,
const uint8_t channel,
const uint32_t duration )

```

This API is used to set/cancel the remain on channel configuration.

#### Note

When status is false, channel and duration parameters are ignored.

#### Parameters

in	<i>bss_type</i>	The interface to set channel.
in	<i>status</i>	false : Cancel the remain on channel configuration true : Set the remain on channel configuration
in	<i>channel</i>	The channel to configure
in	<i>duration</i>	The duration for which to remain on channel in milliseconds.

#### Returns

WM\_SUCCESS on success or error code.

#### 5.7.3.77 wlan\_get\_otp\_user\_data()

```

int wlan_get_otp_user_data (
    uint8_t * buf,
    uint16_t len )

```

Get User Data from OTP Memory

#### Parameters

in	<i>buf</i>	Pointer to buffer where data will be stored
in	<i>len</i>	Number of bytes to read

#### Returns

WM\_SUCCESS if user data read operation is successful.  
 -WM\_E\_INVALID if buf is not valid or of insufficient size.  
 -WM\_FAIL if user data field is not present or command fails.

#### 5.7.3.78 wlan\_get\_cal\_data()

```

int wlan_get_cal_data (
    wlan_cal_data_t * cal_data )

```

Get calibration data from WLAN firmware

## Parameters

out	<i>cal_data</i>	Pointer to calibration data structure where calibration data and it's length will be stored.
-----	-----------------	--

## Returns

WM\_SUCCESS if cal data read operation is successful.  
-WM\_E\_INVALID if *cal\_data* is not valid.  
-WM\_FAIL if command fails.

## Note

The user of this API should free the allocated buffer for calibration data.

## 5.7.3.79 wlan\_set\_chanlist\_and\_txpwrlimit()

```
int wlan_set_chanlist_and_txpwrlimit (
    wlan_chanlist_t * chanlist,
    wlan_txpwrlimit_t * txpwrlimit )
```

Set the Channel List and TRPC channel configuration.

## Parameters

in	<i>chanlist</i>	A pointer to <a href="#">wlan_chanlist_t</a> Channel List configuration.
in	<i>txpwrlimit</i>	A pointer to <a href="#">wlan_txpwrlimit_t</a> TX PWR Limit configuration.

## Returns

WM\_SUCCESS on success, error otherwise.

## 5.7.3.80 wlan\_set\_chanlist()

```
int wlan_set_chanlist (
    wlan_chanlist_t * chanlist )
```

Set the Channel List configuration.

## Parameters

in	<i>chanlist</i>	A pointer to <a href="#">wlan_chanlist_t</a> Channel List configuration.
----	-----------------	--

**Returns**

WM\_SUCCESS on success, error otherwise.

**Note**

If Region Enforcement Flag is enabled in the OTP then this API will not take effect.

**5.7.3.81 wlan\_get\_chanlist()**

```
int wlan_get_chanlist (
    wlan_chanlist_t * chanlist )
```

Get the Channel List configuration.

**Parameters**

out	<i>chanlist</i>	A pointer to <a href="#">wlan_chanlist_t</a> Channel List configuration.
-----	-----------------	--

**Returns**

WM\_SUCCESS on success, error otherwise.

**Note**

The [wlan\\_chanlist\\_t](#) struct allocates memory for a maximum of 54 channels.

**5.7.3.82 wlan\_set\_txpwrlimit()**

```
int wlan_set_txpwrlimit (
    wlan_txpwrlimit_t * txpwrlimit )
```

Set the TRPC channel configuration.

**Parameters**

in	<i>txpwrlimit</i>	A pointer to <a href="#">wlan_txpwrlimit_t</a> TX PWR Limit configuration.
----	-------------------	--

**Returns**

WM\_SUCCESS on success, error otherwise.

## 5.7.3.83 wlan\_get\_txpwrlimit()

```
int wlan_get_txpwrlimit (
    wifi_SubBand_t subband,
    wifi_txpwrlimit_t * txpwrlimit )
```

Get the TRPC channel configuration.

## Parameters

in	subband	Where subband is: 0x00 2G subband (2.4G: channel 1-14) 0x10 5G subband0 (5G: channel 36,40,44,48, 52,56,60,64) 0x11 5G subband1 (5G: channel 100,104,108,112, 116,120,124,128, 132,136,140,144) 0x12 5G subband2 (5G: channel 149,153,157,161,165,172) 0x13 5G subband3 (5G: channel 183,184,185,187,188, 189, 192,196; 5G: channel 7,8,11,12,16,34)
out	txpwrlimit	A pointer to <a href="#">wlan_txpwrlimit_t</a> TX PWR Limit configuration structure where Wi-Fi firmware configuration will get copied.

## Returns

WM\_SUCCESS on success, error otherwise.

## Note

application can use [print\\_txpwrlimit](#) API to print the content of the txpwrlimit structure.

## 5.7.3.84 wlan\_set\_reassoc\_control()

```
void wlan_set_reassoc_control (
    bool reassoc_control )
```

Set Reassociation Control in WLAN Connection Manager

## Note

Reassociation is enabled by default in the WLAN Connection Manager.

## Parameters

in	reassoc_control	Reassociation enable/disable
----	-----------------	------------------------------

### 5.7.3.85 wlan\_uap\_set\_beacon\_period()

```
void wlan_uap_set_beacon_period (
    const uint16_t beacon_period )
```

API to set the beacon period of uAP

#### Parameters

in	<i>beacon_period</i>	Beacon period in TU (1 TU = 1024 micro seconds)
----	----------------------	---

#### Note

Please call this API before calling uAP start API.

### 5.7.3.86 wlan\_uap\_set\_bandwidth()

```
int wlan_uap_set_bandwidth (
    const uint8_t bandwidth )
```

API to set the bandwidth of uAP

#### Parameters

in	<i>Wi-Fi</i>	AP Bandwidth (20MHz/40MHz) 1: 20 MHz 2: 40 MHz
----	--------------	--

#### Returns

WM\_SUCCESS if successful otherwise failure.  
-WM\_FAIL if command fails.

#### Note

Please call this API before calling uAP start API.  
Default bandwidth setting is 40 MHz.

### 5.7.3.87 wlan\_uap\_set\_hidden\_ssid()

```
void wlan_uap_set_hidden_ssid (
    const bool bcast_ssid_ctl )
```

API to control SSID broadcast capability of uAP

This API enables/disables the SSID broadcast feature (also known as the hidden SSID feature). When broadcast SSID is enabled, the AP responds to probe requests from client stations that contain null SSID. When broadcast SSID is disabled, the AP does not respond to probe requests that contain null SSID and generates beacons that contain null SSID.

## Parameters

in	<i>bcast_ssid_ctl</i>	Broadcast SSID control if true SSID will be hidden otherwise it will be visible.
----	-----------------------	--

## Note

Please call this API before calling uAP start API.

## 5.7.3.88 wlan\_uap\_ctrl\_deauth()

```
void wlan_uap_ctrl_deauth (
    const bool enable )
```

API to control the deauth during uAP channel switch

## Parameters

in	<i>enable</i>	0 – Wi-Fi firmware will use default behaviour. 1 – Wi-Fi firmware will not send deauth packet when uap move to another channel.
----	---------------	---

## Note

Please call this API before calling uAP start API.

## 5.7.3.89 wlan\_uap\_set\_ecsa()

```
void wlan_uap_set_ecsa (
    void )
```

API to enable channel switch announcement functionality on uAP.

## Note

Please call this API before calling uAP start API. Also note that 11N should be enabled on uAP. The channel switch announcement IE is transmitted in 7 beacons before the channel switch, during a station connection attempt on a different channel with Ex-AP.

## 5.7.3.90 wlan\_uap\_set\_htcapinfo()

```
void wlan_uap_set_htcapinfo (
    const uint16_t ht_cap_info )
```

API to set the HT Capability Information of uAP

## Parameters

in	<i>ht_cap_info</i>	- This is a bitmap and should be used as following Bit 15: L Sig TxOP protection - reserved, set to 0 Bit 14: 40 MHz intolerant - reserved, set to 0 Bit 13: PSMP - reserved, set to 0 Bit 12: DSSS Cck40MHz mode Bit 11: Maximal AMSDU size - reserved, set to 0 Bit 10: Delayed BA - reserved, set to 0 Bits 9:8: Rx STBC - reserved, set to 0 Bit 7: Tx STBC - reserved, set to 0 Bit 6: Short GI 40 MHz Bit 5: Short GI 20 MHz Bit 4: GF preamble Bits 3:2: MIMO power save - reserved, set to 0 Bit 1: SuppChanWidth - set to 0 for 2.4 GHz band Bit 0: LDPC coding - reserved, set to 0
----	--------------------	---

## Note

Please call this API before calling uAP start API.

## 5.7.3.91 wlan\_uap\_set\_httxcfg()

```
void wlan_uap_set_httxcfg (
    unsigned short httxcfg )
```

This API can be used to configure various 11n specific configuration for transmit (such as Short GI, Channel BW and Green field support) for uAP interface.

## Parameters

in	<i>httxcfg</i>	This is a bitmap and should be used as following Bit 15-8: Reserved set to 0 Bit 7: STBC enable/disable Bit 6: Short GI in 40 Mhz enable/disable Bit 5: Short GI in 20 Mhz enable/disable Bit 4: Green field enable/disable Bit 3-2: Reserved set to 1 Bit 1: 20/40 Mhz enable disable. Bit 0: LDPC enable/disable When Bit 1 is set then firmware could transmit in 20Mhz or 40Mhz based on rate adaptation. When this bit is reset then firmware will only transmit in 20Mhz.
----	----------------	--

## Note

Please call this API before calling uAP start API.



#### 5.7.3.92 wlan\_sta\_ampdu\_tx\_enable()

```
void wlan_sta_ampdu_tx_enable (  
    void )
```

This API can be used to enable AMPDU support on the go when station is a transmitter.

##### Note

By default the station AMPDU TX support is on if configuration option is enabled in defconfig.

#### 5.7.3.93 wlan\_sta\_ampdu\_tx\_disable()

```
void wlan_sta_ampdu_tx_disable (  
    void )
```

This API can be used to disable AMPDU support on the go when station is a transmitter.

##### Note

By default the station AMPDU RX support is on if configuration option is enabled in defconfig.

#### 5.7.3.94 wlan\_sta\_ampdu\_rx\_enable()

```
void wlan_sta_ampdu_rx_enable (  
    void )
```

This API can be used to enable AMPDU support on the go when station is a receiver.

#### 5.7.3.95 wlan\_sta\_ampdu\_rx\_disable()

```
void wlan_sta_ampdu_rx_disable (  
    void )
```

This API can be used to disable AMPDU support on the go when station is a receiver.

#### 5.7.3.96 wlan\_uap\_set\_scan\_chan\_list()

```
void wlan_uap_set_scan_chan_list (  
    wifi_scan_chan_list_t scan_chan_list )
```

Set number of channels and channel number used during automatic channel selection of uAP.

## Parameters

in	<i>scan_chan_list</i>	A structure holding the number of channels and channel numbers.
----	-----------------------	---

## Note

Please call this API before uAP start API in order to set the user defined channels, otherwise it will have no effect. There is no need to call this API every time before uAP start, if once set same channel configuration will get used in all upcoming uAP start call. If user wish to change the channels at run time then it make sense to call this API before every uAP start API.

## 5.7.3.97 wlan\_set\_crypto\_RC4\_encrypt()

```
int wlan_set_crypto_RC4_encrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * KeyIV,
    const t_u16 KeyIVLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto RC4 algorithm encrypt command param.

## Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

## Returns

WM\_SUCCESS if successful otherwise failure.

## Note

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the encrypted data. The value of DataLength is updated to the encrypted data length. The length of the encrypted data is the same as the origin DataLength.

## 5.7.3.98 wlan\_set\_crypto\_RC4\_decrypt()

```
int wlan_set_crypto_RC4_decrypt (
    const t_u8 * Key,
```

```

    const t_u16 KeyLength,
    const t_u8 * KeyIV,
    const t_u16 KeyIVLength,
    t_u8 * Data,
    t_u16 * DataLength )

```

Set Crypto RC4 algorithm decrypt command param.

#### Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

#### Returns

WM\_SUCCESS if successful otherwise failure.

#### Note

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the decrypted data. The value of DataLength is updated to the decrypted data length. The length of the decrypted data is the same as the origin DataLength.

#### 5.7.3.99 wlan\_set\_crypto\_AES\_ECB\_encrypt()

```

int wlan_set_crypto_AES_ECB_encrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * KeyIV,
    const t_u16 KeyIVLength,
    t_u8 * Data,
    t_u16 * DataLength )

```

Set Crypto AES\_ECB algorithm encrypt command param.

#### Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the encrypted data. The value of DataLength is updated to the encrypted data length. The length of the encrypted data is the same as the origin DataLength.

**5.7.3.100 wlan\_set\_crypto\_AES\_ECB\_decrypt()**

```
int wlan_set_crypto_AES_ECB_decrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * KeyIV,
    const t_u16 KeyIVLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto AES\_ECB algorithm decrypt command param.

**Parameters**

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the decrypted data. The value of DataLength is updated to the decrypted data length. The length of the decrypted data is the same as the origin DataLength.

**5.7.3.101 wlan\_set\_crypto\_AES\_WRAP\_encrypt()**

```
int wlan_set_crypto_AES_WRAP_encrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
```

```

const t_u8 * KeyIV,
const t_u16 KeyIVLength,
t_u8 * Data,
t_u16 * DataLength )

```

Set Crypto AES\_WRAP algorithm encrypt command param.

#### Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

#### Returns

WM\_SUCCESS if successful otherwise failure.

#### Note

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the encrypted data. The value of DataLength is updated to the encrypted data length. The encrypted data is 8 bytes more than the original data. Therefore, the address pointed to by Data needs to reserve enough space.

#### 5.7.3.102 wlan\_set\_crypto\_AES\_WRAP\_decrypt()

```

int wlan_set_crypto_AES_WRAP_decrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * KeyIV,
    const t_u16 KeyIVLength,
    t_u8 * Data,
    t_u16 * DataLength )

```

Set Crypto AES\_WRAP algorithm decrypt command param.

#### Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>KeyIV</i>	KeyIV
in	<i>KeyIVLength</i>	The maximum keyIV length is 32.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the decrypted data. The value of DataLength is updated to the decrypted data length. The decrypted data is 8 bytes less than the original data.

**5.7.3.103 wlan\_set\_crypto\_AES\_CCMP\_encrypt()**

```
int wlan_set_crypto_AES_CCMP_encrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * AAD,
    const t_u16 AADLength,
    const t_u8 * Nonce,
    const t_u16 NonceLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto AES\_CCMP algorithm encrypt command param.

**Parameters**

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>AAD</i>	AAD
in	<i>AADLength</i>	The maximum AAD length is 32.
in	<i>Nonce</i>	Nonce
in	<i>NonceLength</i>	The maximum Nonce length is 14.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the encrypted data. The value of DataLength is updated to the encrypted data length. The encrypted data is 8 or 16 bytes more than the original data. Therefore, the address pointed to by Data needs to reserve enough space.

## 5.7.3.104 wlan\_set\_crypto\_AES\_CCMP\_decrypt()

```
int wlan_set_crypto_AES_CCMP_decrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * AAD,
    const t_u16 AADLength,
    const t_u8 * Nonce,
    const t_u16 NonceLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto AES\_CCMP algorithm decrypt command param.

## Parameters

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>AAD</i>	AAD
in	<i>AADLength</i>	The maximum AAD length is 32.
in	<i>Nonce</i>	Nonce
in	<i>NonceLength</i>	The maximum Nonce length is 14.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

## Returns

WM\_SUCCESS if successful otherwise failure.

## Note

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the decrypted data. The value of DataLength is updated to the decrypted data length. The decrypted data is 8 or 16 bytes less than the original data.

## 5.7.3.105 wlan\_set\_crypto\_AES\_GCMP\_encrypt()

```
int wlan_set_crypto_AES_GCMP_encrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * AAD,
    const t_u16 AADLength,
    const t_u8 * Nonce,
    const t_u16 NonceLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto AES\_GCMP algorithm encrypt command param.

**Parameters**

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>AAD</i>	AAD
in	<i>AADLength</i>	The maximum AAD length is 32.
in	<i>Nonce</i>	Nonce
in	<i>NonceLength</i>	The maximum Nonce length is 14.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.

**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the encrypted data. The value of DataLength is updated to the encrypted data length. The encrypted data is 16 bytes more than the original data. Therefore, the address pointed to by Data needs to reserve enough space.

**5.7.3.106 wlan\_set\_crypto\_AES\_GCMP\_decrypt()**

```
int wlan_set_crypto_AES_GCMP_decrypt (
    const t_u8 * Key,
    const t_u16 KeyLength,
    const t_u8 * AAD,
    const t_u16 AADLength,
    const t_u8 * Nonce,
    const t_u16 NonceLength,
    t_u8 * Data,
    t_u16 * DataLength )
```

Set Crypto AES\_CCMP algorithm decrypt command param.

**Parameters**

in	<i>Key</i>	key
in	<i>KeyLength</i>	The maximum key length is 32.
in	<i>AAD</i>	AAD
in	<i>AADLength</i>	The maximum AAD length is 32.
in	<i>Nonce</i>	Nonce
in	<i>NonceLength</i>	The maximum Nonce length is 14.
in	<i>Data</i>	Data
in	<i>DataLength</i>	The maximum Data length is 1300.



**Returns**

WM\_SUCCESS if successful otherwise failure.

**Note**

If the function returns WM\_SUCCESS, the data in the memory pointed to by Data is overwritten by the decrypted data. The value of DataLength is updated to the decrypted data length. The decrypted data is 16 bytes less than the original data.

**5.7.3.107 wlan\_send\_hostcmd()**

```
int wlan_send_hostcmd (
    void * cmd_buf,
    uint32_t cmd_buf_len,
    void * host_resp_buf,
    uint32_t resp_buf_len,
    uint32_t * reqd_resp_len )
```

This function sends the host command to f/w and copies back response to caller provided buffer in case of success. Response from firmware is not parsed by this function but just copied back to the caller buffer.

**Parameters**

in	<i>cmd_buf</i>	Buffer containing the host command with header
in	<i>cmd_buf_len</i>	length of valid bytes in cmd_buf
out	<i>resp_buf</i>	Caller provided buffer, in case of success command response is copied to this buffer Can be same as cmd_buf
in	<i>resp_buf_len</i>	resp_buf's allocated length
out	<i>reqd_resp_len</i>	length of valid bytes in response buffer if successful otherwise invalid.

**Returns**

WM\_SUCCESS in case of success.

WM\_E\_INBIG in case cmd\_buf\_len is bigger than the commands that can be handled by driver.

WM\_E\_INSMALL in case cmd\_buf\_len is smaller than the minimum length. Minimum length is atleast the length of command header. Please see Note for same.

WM\_E\_OUTBIG in case the resp\_buf\_len is not sufficient to copy response from firmware. reqd\_resp\_len is updated with the response size.

WM\_E\_INVALID in case cmd\_buf\_len and resp\_buf\_len have invalid values.

WM\_E\_NOMEM in case cmd\_buf, resp\_buf and reqd\_resp\_len are NULL

**Note**

Brief on the Command Header: Start 8 bytes of cmd\_buf should have these values set. Firmware would update resp\_buf with these 8 bytes at the start.

2 bytes : Command.

2 bytes : Size.

2 bytes : Sequence number.

2 bytes : Result.

Rest of buffer length is Command/Response Body.

## 5.7.4 Macro Documentation

### 5.7.4.1 ACTION\_GET

```
#define ACTION_GET (0U)
```

Action GET

### 5.7.4.2 ACTION\_SET

```
#define ACTION_SET (1)
```

Action SET

### 5.7.4.3 IEEEtypes\_SSID\_SIZE

```
#define IEEEtypes_SSID_SIZE 32U
```

Maximum SSID length

### 5.7.4.4 IEEEtypes\_ADDRESS\_SIZE

```
#define IEEEtypes_ADDRESS_SIZE 6
```

MAC Address length

### 5.7.4.5 WLAN\_RESCAN\_LIMIT

```
#define WLAN_RESCAN_LIMIT 5U
```

The number of times that the WLAN Connection Manager will look for a network before giving up.

### 5.7.4.6 WLAN\_RECONNECT\_LIMIT

```
#define WLAN_RECONNECT_LIMIT 5U
```

The number of times that the WLAN Connection Manager will attempt a reconnection with the network before giving up.

### 5.7.4.7 WLAN\_NETWORK\_NAME\_MIN\_LENGTH

```
#define WLAN_NETWORK_NAME_MIN_LENGTH 1U
```

The minimum length for network names, see [wlan\\_network](#). This must be between 1 and [WLAN\\_NETWORK\\_NAME\\_MAX\\_LENGTH](#).

#### 5.7.4.8 WLAN\_NETWORK\_NAME\_MAX\_LENGTH

```
#define WLAN_NETWORK_NAME_MAX_LENGTH 32U
```

The space reserved for storing network names, [wlan\\_network](#)

#### 5.7.4.9 WLAN\_PSK\_MIN\_LENGTH

```
#define WLAN_PSK_MIN_LENGTH 8U
```

The space reserved for storing PSK (password) phrases.

#### 5.7.4.10 WLAN\_MAX\_KNOWN\_NETWORKS

```
#define WLAN_MAX_KNOWN_NETWORKS CONFIG_WLAN_KNOWN_NETWORKS
```

The size of the list of known networks maintained by the WLAN Connection Manager

#### 5.7.4.11 WLAN\_PMK\_LENGTH

```
#define WLAN_PMK_LENGTH 32
```

Length of a pairwise master key (PMK). It's always 256 bits (32 Bytes)

#### 5.7.4.12 WLAN\_ERROR\_NONE

```
#define WLAN_ERROR_NONE 0
```

The operation was successful.

#### 5.7.4.13 WLAN\_ERROR\_PARAM

```
#define WLAN_ERROR_PARAM 1
```

The operation failed due to an error with one or more parameters.

#### 5.7.4.14 WLAN\_ERROR\_NOMEM

```
#define WLAN_ERROR_NOMEM 2
```

The operation could not be performed because there is not enough memory.

#### 5.7.4.15 WLAN\_ERROR\_STATE

```
#define WLAN_ERROR_STATE 3
```

The operation could not be performed in the current system state.

#### 5.7.4.16 WLAN\_ERROR\_ACTION

```
#define WLAN_ERROR_ACTION 4
```

The operation failed due to an internal error.

#### 5.7.4.17 WLAN\_ERROR\_PS\_ACTION

```
#define WLAN_ERROR_PS_ACTION 5
```

The operation to change power state could not be performed

#### 5.7.4.18 WLAN\_ERROR\_NOT\_SUPPORTED

```
#define WLAN_ERROR_NOT_SUPPORTED 6
```

The requested feature is not supported

### 5.7.5 Typedef Documentation

#### 5.7.5.1 wlan\_scan\_channel\_list\_t

```
typedef wifi_scan_channel_list_t wlan_scan_channel_list_t
```

Configuration for Wireless scan channel list from [wifi\\_scan\\_channel\\_list\\_t](#)

#### 5.7.5.2 wlan\_scan\_params\_v2\_t

```
typedef wifi_scan_params_v2_t wlan_scan_params_v2_t
```

Configuration for wireless scanning parameters v2 from [wifi\\_scan\\_params\\_v2\\_t](#)

#### 5.7.5.3 wlan\_cal\_data\_t

```
typedef wifi_cal_data_t wlan_cal_data_t
```

Configuration for Wireless Calibration data from [wifi\\_cal\\_data\\_t](#)

#### 5.7.5.4 wlanflt\_cfg\_t

```
typedef wififlt_cfg_t wlanflt_cfg_t
```

Configuration for Memory Efficient Filters in Wi-Fi firmware from [wififlt\\_cfg\\_t](#)

#### 5.7.5.5 wlan\_wowlan\_ptn\_cfg\_t

```
typedef wifi_wowlan_ptn_cfg_t wlan_wowlan_ptn_cfg_t
```

Configuration for wowlan pattern parameters from [wifi\\_wowlan\\_ptn\\_cfg\\_t](#)

#### 5.7.5.6 wlan\_tcp\_keep\_alive\_t

```
typedef wifi_tcp_keep_alive_t wlan_tcp_keep_alive_t
```

Configuration for TCP Keep alive parameters from [wifi\\_tcp\\_keep\\_alive\\_t](#)

#### 5.7.5.7 wlan\_ds\_rate

```
typedef wifi_ds_rate wlan_ds_rate
```

Configuration for TX Rate and Get data rate from [wifi\\_ds\\_rate](#)

#### 5.7.5.8 wlan\_ed\_mac\_ctrl\_t

```
typedef wifi_ed_mac_ctrl_t wlan_ed_mac_ctrl_t
```

Configuration for ED MAC Control parameters from [wifi\\_ed\\_mac\\_ctrl\\_t](#)

#### 5.7.5.9 wlan\_bandcfg\_t

```
typedef wifi_bandcfg_t wlan_bandcfg_t
```

Configuration for Band from [wifi\\_bandcfg\\_t](#)

#### 5.7.5.10 wlan\_cw\_mode\_ctrl\_t

```
typedef wifi_cw_mode_ctrl_t wlan_cw_mode_ctrl_t
```

Configuration for CW Mode parameters from [wifi\\_cw\\_mode\\_ctrl\\_t](#)

#### 5.7.5.11 wlan\_chanlist\_t

```
typedef wifi_chanlist_t wlan_chanlist_t
```

Configuration for Channel list from [wifi\\_chanlist\\_t](#)

#### 5.7.5.12 wlan\_txpwrlimit\_t

```
typedef wifi_txpwrlimit_t wlan_txpwrlimit_t
```

Configuration for TX Pwr Limit from [wifi\\_txpwrlimit\\_t](#)

### 5.7.6 Enumeration Type Documentation

#### 5.7.6.1 wm\_wlan\_errno

```
enum wm_wlan_errno
```

Enum for wlan errors

## Enumerator

WLAN_ERROR_FW_DNLD_FAILED	The Firmware download operation failed.
WLAN_ERROR_FW_NOT_READY	The Firmware ready register not set.
WLAN_ERROR_CARD_NOT_DETECTED	The WiFi card not found.
WLAN_ERROR_FW_NOT_DETECTED	The WiFi Firmware not found.
WLAN_BSSID_NOT_FOUND_IN_SCAN_LIST	BSSID not found in scan list

## 5.7.6.2 wlan\_event\_reason

```
enum wlan_event_reason
```

WLAN Connection Manager event reason

## Enumerator

WLAN_REASON_SUCCESS	The WLAN Connection Manager has successfully connected to a network and is now in the <a href="#">WLAN_CONNECTED</a> state.
WLAN_REASON_AUTH_SUCCESS	The WLAN Connection Manager has successfully authenticated to a network and is now in the <a href="#">WLAN_ASSOCIATED</a> state.
WLAN_REASON_CONNECT_FAILED	The WLAN Connection Manager failed to connect before actual connection attempt with AP due to incorrect wlan network profile. or The WLAN Connection Manager failed to reconnect to previously connected network and it is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_NETWORK_NOT_FOUND	The WLAN Connection Manager could not find the network that it was connecting to and it is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_NETWORK_AUTH_FAILED	The WLAN Connection Manager failed to authenticate with the network and is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_ADDRESS_SUCCESS	DHCP lease has been renewed.
WLAN_REASON_ADDRESS_FAILED	The WLAN Connection Manager failed to obtain an IP address or TCP stack configuration has failed or the IP address configuration was lost due to a DHCP error. The system is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_LINK_LOST	The WLAN Connection Manager has lost the link to the current network.
WLAN_REASON_CHAN_SWITCH	The WLAN Connection Manager has received the channel switch announcement from the current network.
WLAN_REASON_WPS_DISCONNECT	The WLAN Connection Manager has disconnected from the WPS network (or has canceled a connection attempt) by request and is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_USER_DISCONNECT	The WLAN Connection Manager has disconnected from the current network (or has canceled a connection attempt) by request and is now in the <a href="#">WLAN_DISCONNECTED</a> state.
WLAN_REASON_INITIALIZED	The WLAN Connection Manager is initialized and is ready for use. That is, it's now possible to scan or to connect to a network.

## Enumerator

WLAN_REASON_INITIALIZATION_FAILED	The WLAN Connection Manager has failed to initialize and is therefore not running. It is not possible to scan or to connect to a network. The WLAN Connection Manager should be stopped and started again via <a href="#">wlan_stop()</a> and <a href="#">wlan_start()</a> respectively.
WLAN_REASON_PS_ENTER	The WLAN Connection Manager has entered power save mode.
WLAN_REASON_PS_EXIT	The WLAN Connection Manager has exited from power save mode.
WLAN_REASON_UAP_SUCCESS	The WLAN Connection Manager has started uAP
WLAN_REASON_UAP_CLIENT_ASSOC	A wireless client has joined uAP's BSS network
WLAN_REASON_UAP_CLIENT DISSOC	A wireless client has left uAP's BSS network
WLAN_REASON_UAP_START_FAILED	The WLAN Connection Manager has failed to start uAP
WLAN_REASON_UAP_STOP_FAILED	The WLAN Connection Manager has failed to stop uAP
WLAN_REASON_UAP_STOPPED	The WLAN Connection Manager has stopped uAP

## 5.7.6.3 wlan\_wakeup\_event\_t

enum [wlan\\_wakeup\\_event\\_t](#)

Wakeup events for which wakeup will occur

## Enumerator

WAKE_ON_ALL_BROADCAST	Wakeup on broadcast
WAKE_ON_UNICAST	Wakeup on unicast
WAKE_ON_MAC_EVENT	Wakeup on MAC event
WAKE_ON_MULTICAST	Wakeup on multicast
WAKE_ON_ARP_BROADCAST	Wakeup on ARP broadcast
WAKE_ON_MGMT_FRAME	Wakeup on receiving a management frame

## 5.7.6.4 wlan\_connection\_state

enum [wlan\\_connection\\_state](#)

WLAN station/micro-AP/Wi-Fi Direct Connection/Status state

## Enumerator

WLAN_DISCONNECTED	The WLAN Connection Manager is not connected and no connection attempt is in progress. It is possible to connect to a network or scan.
WLAN_CONNECTING	The WLAN Connection Manager is not connected but it is currently attempting to connect to a network. It is not possible to scan at this time. It is possible to connect to a different network.

## Enumerator

WLAN_ASSOCIATED	The WLAN Connection Manager is not connected but associated.
WLAN_CONNECTED	The WLAN Connection Manager is connected. It is possible to scan and connect to another network at this time. Information about the current network configuration is available.
WLAN_UAP_STARTED	The WLAN Connection Manager has started uAP
WLAN_UAP_STOPPED	The WLAN Connection Manager has stopped uAP
WLAN_SCANNING	The WLAN Connection Manager is not connected and network scan is in progress.
WLAN_ASSOCIATING	The WLAN Connection Manager is not connected and network association is in progress.

## 5.7.6.5 wlan\_ps\_mode

```
enum wlan_ps_mode
```

Station Power save mode

## Enumerator

WLAN_ACTIVE	Active mode
WLAN_IEEE	IEEE power save mode
WLAN_DEEP_SLEEP	Deep sleep power save mode

## 5.7.6.6 wlan\_security\_type

```
enum wlan_security_type
```

Network security types

## Enumerator

WLAN_SECURITY_NONE	The network does not use security.
WLAN_SECURITY_WEP_OPEN	The network uses WEP security with open key.
WLAN_SECURITY_WEP_SHARED	The network uses WEP security with shared key.
WLAN_SECURITY_WPA	The network uses WPA security with PSK.
WLAN_SECURITY_WPA2	The network uses WPA2 security with PSK.
WLAN_SECURITY_WPA_WPA2_MIXED	The network uses WPA/WPA2 mixed security with PSK
WLAN_SECURITY_WILDCARD	The network can use any security method. This is often used when the user only knows the name and passphrase but not the security type.
WLAN_SECURITY_WPA3_SAE	The network uses WPA3 security with SAE. Also set the PMF settings using <a href="#">wlan_set_pmfcfg</a> API required for WPA3 SAE
WLAN_SECURITY_WPA2_WPA3_SAE_MIXED	The network uses WPA2/WPA3 SAE mixed security with PSK. This security mode is specific to uAP or SoftAP only



### 5.7.6.7 address\_types

enum [address\\_types](#)

Address types to be used by the element wlan\_ip\_config.addr\_type below

#### Enumerator

ADDR_TYPE_STATIC	static IP address
ADDR_TYPE_DHCP	Dynamic IP address
ADDR_TYPE_LLA	Link level address

## 5.8 wlan\_11d.h File Reference

WLAN module 11d API.

### 5.8.1 Function Documentation

#### 5.8.1.1 wlan\_enable\_11d()

```
static int wlan_enable_11d (  
    void ) [inline], [static]
```

wlan\_11d Wi-Fi Region Configuration By default, the SDK builds applications that are compliant with the US region configuration. This implies that the module obeys the US regulations for Wi-Fi transmissions on certified frequency bands. The SDK provides mechanism for configuring various region codes in the applications. This can be performed in one of the following two ways:

#### I) Specifying Country Code

In this method of configuration, the application defines up-front what is the country code that the device is going to be deployed in. Once configured the Wi-Fi firmware obeys the configured countries regulations. This configuration can be set by making a call to the [wlan\\_set\\_country\(\)](#) API. This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

For example: wlan\_set\_country(COUNTRY\_CN);

#### II) Using 802.11D

**Note**

The FCC does not allow the use of 802.11D in the US starting Jan 1, 2015. In this method of configuration, the Wi-Fi driver of the SDK will scan for Access Points in the vicinity and accordingly configure itself to operate in the available frequency bands. This configuration can be set by making a call to the [wlan\\_enable\\_11d\(\)](#) API. This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

For example: [wlan\\_enable\\_11d\(\)](#); Enable 11D support in WLAN Driver.

**Note**

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

Either this function or [wlan\\_set\\_country\(\)](#) should be used at a time. If both functions are called in the application, then WLAN Driver properties will be set as per the [wlan\\_set\\_country\(\)](#) function.

**Returns**

-WM\_FAIL if operation was failed.  
WM\_SUCCESS if operation was successful.

**5.8.1.2 wlan\_get\_country()**

```
static country\_code\_t wlan_get_country (  
    void ) [inline], [static]
```

Get country code from WLAN Driver.

**Note**

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

**Returns**

Country code. Refer to [country\\_code\\_t](#).

**5.8.1.3 wlan\_uap\_set\_country()**

```
static int wlan_uap_set_country (  
    country\_code\_t country ) [inline], [static]
```

Set country code in WLAN Driver.

**Note**

This API should be called after WLAN is initialized but before starting uAP interface.

Either this function or [wlan\\_enable\\_11d\(\)](#) should be used at a time. If both functions are called in the application, then WLAN Driver properties will be set as per the [wlan\\_uap\\_set\\_country\(\)](#) function.

## Parameters

in	<i>country</i>	Country code. Refer to <a href="#">country_code_t</a> .
----	----------------	---

## Returns

-WM\_FAIL if operation was failed.  
WM\_SUCCESS if operation was successful.

## 5.8.1.4 wlan\_set\_country()

```
static int wlan_set_country (
    country\_code\_t country ) [inline], [static]
```

Set country code in WLAN Driver.

## Note

This API should be called after WLAN is initialized but before making any connection attempts on station interface.

Either this function or [wlan\\_enable\\_11d\(\)](#) should be used at a time. If both functions are called in the application, then WLAN Driver properties will be set as per the [wlan\\_set\\_country\(\)](#) function.

## Parameters

in	<i>country</i>	Country code. Refer to <a href="#">country_code_t</a> .
----	----------------	---

## Returns

-WM\_FAIL if operation was failed.  
WM\_SUCCESS if operation was successful.

## 5.8.1.5 wlan\_set\_domain\_params()

```
static int wlan_set_domain_params (
    wifi\_domain\_param\_t * dp ) [inline], [static]
```

## wlan\_11d\_custom Custom Wi-Fi Region Configuration

Ideally applications should use either [wlan\\_enable\\_11d\(\)](#) or [wlan\\_set\\_country\(\)](#) APIs to have standard 802.11d functionality as per regulations of Wi-Fi transmissions on certified frequency bands.

But If application wants to configure custom 802.11d configurations then wlan\_set\_domain\_params API can be used for that.

If applications just want to set a particular region then [wlan\\_set\\_region\\_code\(\)](#) API can be used for the purpose.

Supported region code values are given in wlan\_11d.c file.

Sets the domain parameters for the uAP.

**Note**

This API should be called after WLAN is initialized but before starting uAP

To use this API you will need to fill up the structure [wifi\\_domain\\_param\\_t](#) with correct parameters.

**Note**

This API should be called after WLAN is initialized but before making any connection attempts on station interface.

The below section lists all the arrays that can be passed individually or in combination to the API [wlan\\_set\\_domain\\_params\(\)](#). These are the sub band sets to be part of the Country Info IE in the uAP beacon. One of them is to be selected according to your region. Please have a look at the example given in the documentation below for reference.

Supported Country Codes: "US" : USA, "CA" : Canada, "SG" : Singapore, "EU" : Europe, "AU" : Australia, "KR" : Republic of Korea, "CN" : China, "FR" : France, "JP" : Japan

```
Region : US(US) or Canada(CA) or Singapore(SG) 2.4 GHz
wifi_sub_band_set_t subband_US_CA_SG_2_4_GHz[] = {
    {1, 11, 20}
};
```

```
Region: Europe(EU), Australia(AU), Republic of Korea(KR),
China(CN) 2.4 GHz
wifi_sub_band_set_t subband_EU_AU_KR_CN_2_4GHz[] = {
    {1, 13, 20}
};
```

```
Region: France(FR) 2.4 GHz
wifi_sub_band_set_t subband_FR_2_4GHz[] = {
    {1, 9, 20},
    {10, 4, 10}
};
```

```
Region: Japan(JP) 2.4 GHz
wifi_sub_band_set_t subband_JP_2_4GHz[] = {
    {1, 14, 20},
};
```

```
Region: Constrained 2.4 Ghz
wifi_sub_band_set_t subband_CS_2_4GHz[] = {
    {1, 9, 20},
    {10, 2, 10}
};
```

```
Region: US(US) or Singapore(SG) 5 GHz
wifi_sub_band_set_t subband_US_SG_5GHz[] = {
    {36, 1, 20},
    {40, 1, 20},
    {44, 1, 20},
    {48, 1, 20},
    {52, 1, 20},
    {56, 1, 20},
    {60, 1, 20},
    {64, 1, 20},
    {100, 1, 20},
    {104, 1, 20},
    {108, 1, 20},
    {112, 1, 20},
    {116, 1, 20},
    {120, 1, 20},
    {124, 1, 20},
    {128, 1, 20},
    {132, 1, 20},
    {136, 1, 20},
    {140, 1, 20},
    {149, 1, 20},
    {153, 1, 20},
    {157, 1, 20},
    {161, 1, 20},
    {165, 1, 20}
};
```

```
Region: Canada(CA) 5 GHz
```

```
wifi_sub_band_set_t subband_CA_5GHz[] = {
    {36, 1, 20},
    {40, 1, 20},
    {44, 1, 20},
    {48, 1, 20},
    {52, 1, 20},
    {56, 1, 20},
    {60, 1, 20},
    {64, 1, 20},
    {100, 1, 20},
    {104, 1, 20},
    {108, 1, 20},
    {112, 1, 20},
    {116, 1, 20},
    {132, 1, 20},
    {136, 1, 20},
    {140, 1, 20},
    {149, 1, 20},
    {153, 1, 20},
    {157, 1, 20},
    {161, 1, 20},
    {165, 1, 20}
};
```

Region: Europe/ETSI(EU), Australia(AU), Republic of Korea(KR) 5 GHz

```
wifi_sub_band_set_t subband_EU_AU_KR_5GHz[] = {
    {36, 1, 20},
    {40, 1, 20},
    {44, 1, 20},
    {48, 1, 20},
    {52, 1, 20},
    {56, 1, 20},
    {60, 1, 20},
    {64, 1, 20},
    {100, 1, 20},
    {104, 1, 20},
    {108, 1, 20},
    {112, 1, 20},
    {116, 1, 20},
    {120, 1, 20},
    {124, 1, 20},
    {128, 1, 20},
    {132, 1, 20},
    {136, 1, 20},
    {140, 1, 20}
};
```

Region: China(CN) 5 GHz

```
wifi_sub_band_set_t subband_CN_5GHz[] = {
    {149, 1, 33},
    {153, 1, 33},
    {157, 1, 33},
    {161, 1, 33},
    {165, 1, 33}
};
```

Region: France(FR) 5 GHz

```
wifi_sub_band_set_t subband_FR_5GHz[] = {
    {36, 1, 20},
    {40, 1, 20},
    {44, 1, 20},
    {48, 1, 20},
    {52, 1, 20},
    {56, 1, 20},
    {60, 1, 20},
    {64, 1, 20},
    {100, 1, 20},
    {104, 1, 20},
    {108, 1, 20},
    {112, 1, 20},
    {116, 1, 20},
    {120, 1, 20},
    {124, 1, 20},
    {128, 1, 20},
    {132, 1, 20},
    {136, 1, 20},
    {140, 1, 20},
    {149, 1, 20},
    {153, 1, 20},
    {157, 1, 20},
    {161, 1, 20},
    {165, 1, 20}
};
```

Region: Japan(JP) 5 GHz

```
wifi_sub_band_set_t subband_JP_5_GHz[] = {
    {8, 1, 23},
```

```

{12, 1, 23},
{16, 1, 23},
{36, 1, 23},
{40, 1, 23},
{44, 1, 23},
{48, 1, 23},
{52, 1, 23},
{56, 1, 23},
{60, 1, 23},
{64, 1, 23},
{100, 1, 23},
{104, 1, 23},
{108, 1, 23},
{112, 1, 23},
{116, 1, 23},
{120, 1, 23},
{124, 1, 23},
{128, 1, 23},
{132, 1, 23},
{136, 1, 23},
{140, 1, 23}
};

\code
// We will be using the KR 2.4 and 5 GHz bands for this example

int nr_sb = (sizeof(subband_EU_AU_KR_CN_2_4GHz)
+ sizeof(subband_EU_AU_KR_5GHz))
/ sizeof(wifi_sub_band_set_t);

// We already have space for first sub band info entry in
// wifi_domain_param_t
wifi_domain_param_t *dp = os_mem_alloc(sizeof(
    wifi_domain_param_t) +
    (sizeof(wifi_sub_band_set_t) * (nr_sb - 1)));

// COUNTRY_CODE_LEN is 3. Add extra ' ' as country code is 2 characters
(void)memcpy(dp->country_code, "KR ", COUNTRY_CODE_LEN);

dp->no_of_sub_band = nr_sb;
(void)memcpy(&dp->sub_band[0], &subband_EU_AU_KR_CN_2_4GHz[0],
    1 * sizeof(wifi_sub_band_set_t));
(void)memcpy(&dp->sub_band[1], &subband_EU_AU_KR_5GHz,
    (nr_sb - 1) * sizeof(wifi_sub_band_set_t));

wlan_set_domain_params(dp);
os_mem_free(dp);

```

#### Parameters

in	<i>dp</i>	The wifi domain parameters
----	-----------	----------------------------

#### Returns

-WM\_E\_INVALID if invalid parameters were passed.  
WM\_SUCCESS if operation was successful.

#### 5.8.1.6 wlan\_set\_region\_code()

```

static int wlan_set_region_code (
    uint32_t region_code ) [inline], [static]

```

Set 11D region code.

#### Parameters

in	<i>region_code</i>	11D region code to set.
----	--------------------	-------------------------

**Returns**

-WM\_FAIL if operation was failed.  
WM\_SUCCESS if operation was successful.

**5.8.1.7 wlan\_11d\_country\_index\_2\_string()**

```
const uint8_t* wlan_11d_country_index_2_string (  
    int country )
```

Get country string from country code

This function converts country index to country string

**Parameters**

in	<i>country</i>	Country index
----	----------------	---------------

**Returns**

Country string

**5.9 wlan\_tests.h File Reference**

WLAN Connection Manager Tests.

**5.9.1 Function Documentation****5.9.1.1 print\_txpwrlimit()**

```
void print_txpwrlimit (  
    wlan_txpwrlimit_t txpwrlimit )
```

Print the TX PWR Limit table received from Wi-Fi firmware

**Parameters**

in	<i>txpwrlimit</i>	A <a href="#">wlan_txpwrlimit_t</a> struct holding the the TX PWR Limit table received from Wi-Fi firmware.
----	-------------------	---

## 5.10 wm\_net.h File Reference

Network Abstraction Layer.

### 5.10.1 Detailed Description

This provides the calls related to the network layer. The SDK uses lwIP as the network stack.

Here we document the network utility functions provided by the SDK. The detailed lwIP API documentation can be found at: [http://lwip.wikia.com/wiki/Application\\_API\\_layers](http://lwip.wikia.com/wiki/Application_API_layers)

### 5.10.2 Function Documentation

#### 5.10.2.1 net\_dhcp\_hostname\_set()

```
int net_dhcp_hostname_set (
    char * hostname )
```

Set hostname for network interface

##### Parameters

in	<i>hostname</i>	Hostname to be set.
----	-----------------	---------------------

##### Note

NULL is a valid value for hostname.

##### Returns

WM\_SUCESS

#### 5.10.2.2 net\_stop\_dhcp\_timer()

```
void net_stop_dhcp_timer (
    void )
```

Deactivate the dhcp timer

#### 5.10.2.3 net\_socket\_blocking()

```
static int net_socket_blocking (
    int sock,
    int state ) [inline], [static]
```

Set socket blocking option as on or off



## Parameters

in	<i>sock</i>	socket number to be set for blocking option.
in	<i>state</i>	set blocking on or off

## Returns

WM\_SUCESS otherwise standard LWIP error codes.

## 5.10.2.4 net\_get\_sock\_error()

```
static int net_get_sock_error (  
    int sock ) [inline], [static]
```

Get error number from provided socket

## Parameters

in	<i>sock</i>	socket number to get error number.
----	-------------	------------------------------------

## Returns

error number.

## 5.10.2.5 net\_inet\_aton()

```
static uint32_t net_inet_aton (  
    const char * cp ) [inline], [static]
```

Converts Internet host address from the IPv4 dotted-decimal notation into binary form (in network byte order)

## Parameters

in	<i>cp</i>	IPv4 host address in dotted-decimal notation.
----	-----------	---

## Returns

IPv4 address in binary form

### 5.10.2.6 net\_gethostbyname()

```
static int net_gethostbyname (
    const char * cp,
    struct hostent ** hentry ) [inline], [static]
```

Get network host entry

#### Parameters

in	<i>cp</i>	Hostname or an IPv4 address in the standard dot notation.
in	<i>hentry</i>	Pointer to pointer of host entry structure.

#### Note

This function is not thread safe. If thread safety is required please use `lwip_getaddrinfo()` - `lwip_freeaddrinfo()` combination.

#### Returns

WM\_SUCCESS if operation successful.  
-WM\_FAIL if operation fails.

### 5.10.2.7 net\_inet\_ntoa()

```
static void net_inet_ntoa (
    unsigned long addr,
    char * cp ) [inline], [static]
```

Converts Internet host address in network byte order to a string in IPv4 dotted-decimal notation

#### Parameters

in	<i>addr</i>	IP address in network byte order.
out	<i>cp</i>	buffer in which IPv4 dotted-decimal string is returned.

### 5.10.2.8 net\_is\_ip\_or\_ipv6()

```
static bool net_is_ip_or_ipv6 (
    const uint8_t * buffer ) [inline], [static]
```

Check whether buffer is IPv4 or IPV6 packet type

**Parameters**

in	<i>buffer</i>	pointer to buffer where packet to be checked located.
----	---------------	---

**Returns**

true if buffer packet type matches with IPv4 or IPv6, false otherwise.

**5.10.2.9 net\_sock\_to\_interface()**

```
void* net_sock_to_interface (
    int sock )
```

Get interface handle from socket descriptor

Given a socket descriptor this API returns which interface it is bound with.

**Parameters**

in	<i>sock</i>	socket descriptor
----	-------------	-------------------

**Returns**

[out] interface handle

**5.10.2.10 net\_wlan\_init()**

```
int net_wlan_init (
    void )
```

Initialize TCP/IP networking stack

**Returns**

WM\_SUCCESS on success  
-WM\_FAIL otherwise

**5.10.2.11 net\_wlan\_deinit()**

```
int net_wlan_deinit (
    void )
```

Deinitialize TCP/IP networking stack

**Returns**

WM\_SUCCESS on success  
-WM\_FAIL otherwise

#### 5.10.2.12 net\_get\_sta\_handle()

```
void* net_get_sta_handle (
    void )
```

Get station interface handle

Some APIs require the interface handle to be passed to them. The handle can be retrieved using this API.

##### Returns

station interface handle

#### 5.10.2.13 net\_get\_uap\_handle()

```
void* net_get_uap_handle (
    void )
```

Get micro-AP interface handle

Some APIs require the interface handle to be passed to them. The handle can be retrieved using this API.

##### Returns

micro-AP interface handle

#### 5.10.2.14 net\_interface\_up()

```
void net_interface_up (
    void * intrfc_handle )
```

Take interface up

Change interface state to up. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

##### Parameters

in	<i>intrfc_handle</i>	interface handle
----	----------------------	------------------

##### Returns

void

## 5.10.2.15 net\_interface\_down()

```
void net_interface_down (
    void * intrfc_handle )
```

Take interface down

Change interface state to down. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

## Parameters

in	<i>intrfc_handle</i>	interface handle
----	----------------------	------------------

## Returns

void

## 5.10.2.16 net\_interface\_dhcp\_stop()

```
void net_interface_dhcp_stop (
    void * intrfc_handle )
```

Stop DHCP client on given interface

Stop the DHCP client on given interface state. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

## Parameters

in	<i>intrfc_handle</i>	interface handle
----	----------------------	------------------

## Returns

void

## 5.10.2.17 net\_configure\_address()

```
int net_configure_address (
    struct wlan_ip_config * addr,
    void * intrfc_handle )
```

Configure IP address for interface

## Parameters

in	<i>addr</i>	Address that needs to be configured.
in	<i>intrfc_handle</i>	Handle for network interface to be configured.

**Returns**

WM\_SUCCESS on success or an error code.

**5.10.2.18 net\_configure\_dns()**

```
void net_configure_dns (
    struct wlan_ip_config * ip,
    enum wlan_bss_role role )
```

Configure DNS server address

**Parameters**

in	<i>ip</i>	IP address of the DNS server to set
in	<i>role</i>	Network wireless BSS Role

**5.10.2.19 net\_get\_if\_addr()**

```
int net_get_if_addr (
    struct wlan_ip_config * addr,
    void * intrfc_handle )
```

Get interface IP Address in [wlan\\_ip\\_config](#)

This function will get the IP address of a given interface. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

**Parameters**

out	<i>addr</i>	<a href="#">wlan_ip_config</a>
in	<i>intrfc_handle</i>	interface handle

**Returns**

WM\_SUCCESS on success or error code.

**5.10.2.20 net\_get\_if\_name()**

```
int net_get_if_name (
    char * if_name,
    void * intrfc_handle )
```

Get interface Name string containing name and number

This function will get the string containing name and number for given interface. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

## Parameters

out	<i>if_name</i>	interface name pointer
in	<i>intrfc_handle</i>	interface handle

## Returns

WM\_SUCCESS on success or error code.

## 5.10.2.21 net\_get\_if\_ip\_addr()

```
int net_get_if_ip_addr (
    uint32_t * ip,
    void * intrfc_handle )
```

## Get interface IP Address

This function will get the IP Address of a given interface. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

## Parameters

out	<i>ip</i>	ip address pointer
in	<i>intrfc_handle</i>	interface handle

## Returns

WM\_SUCCESS on success or error code.

## 5.10.2.22 net\_get\_if\_ip\_mask()

```
int net_get_if_ip_mask (
    uint32_t * nm,
    void * intrfc_handle )
```

## Get interface IP Subnet-Mask

This function will get the Subnet-Mask of a given interface. Use [net\\_get\\_sta\\_handle\(\)](#), [net\\_get\\_uap\\_handle\(\)](#) to get interface handle.

## Parameters

in	<i>mask</i>	Subnet Mask pointer
in	<i>intrfc_handle</i>	interface

**Returns**

WM\_SUCCESS on success or error code.

**5.10.2.23 net\_ipv4stack\_init()**

```
void net_ipv4stack_init (  
    void )
```

Initialize the network stack

This function initializes the network stack. This function is called by [wlan\\_start\(\)](#).

Applications may optionally call this function directly: if they wish to use the networking stack (loopback interface) without the wlan functionality. if they wish to initialize the networking stack even before wlan comes up.

**Note**

This function may safely be called multiple times.

**5.10.2.24 net\_stat()**

```
void net_stat (  
    void )
```

Display network statistics

**5.11 wm\_os.h File Reference**

OS Abstraction Layer.

**5.11.1 Detailed Description**

The OS abstraction layer provides wrapper APIs over some of the commonly used OS primitives. Since the behaviour and semantics of the various OSes differs widely, some abstraction APIs require a specific handling as listed below.



### 5.11.2 Usage

The OS abstraction layer provides the following types of primitives:

- Thread: Create or delete a thread using `os_thread_create()` or `os_thread_delete()`. Block a thread using `os_thread_sleep()`. Complete a thread's execution using `os_thread_self_complete()`.
- Message Queue: Create or delete a message queue using `os_queue_create()` or `os_queue_delete()`. Send a message using `os_queue_send()` and received a message using `os_queue_rcv()`.
- Mutex: Create or delete a mutex using `os_mutex_create()` or `os_mutex_delete()`. Acquire a mutex using `os_mutex_get()` and release it using `os_mutex_put()`.
- Semaphores: Create or delete a semaphore using `os_semaphore_create()` / `os_semaphore_create_counting()` or `os_semaphore_delete`. Acquire a semaphore using `os_semaphore_get()` and release it using `os_semaphore_put()`.
- Timers: Create or delete a timer using `os_timer_create()` or `os_timer_delete()`. Change the timer using `os_timer_change()`. Activate or de-activate the timer using `os_timer_activate()` or `os_timer_deactivate()`. Reset a timer using `os_timer_reset()`.
- Dynamic Memory Allocation: Dynamically allocate memory using `os_mem_alloc()`, `os_mem_calloc()` and free it using `os_mem_free()`.

### 5.11.3 Function Documentation

#### 5.11.3.1 `os_ticks_get()`

```
unsigned os_ticks_get (  
    void )
```

Get current OS tick counter value

##### Returns

32 bit value of ticks since boot-up

#### 5.11.3.2 `os_get_timestamp()`

```
unsigned int os_get_timestamp (  
    void )
```

Returns time in micro-secs since bootup

##### Note

The value returned will wrap around after sometime and caller is expected to guard itself against this.

##### Returns

Time in micro-secs since bootup

### 5.11.3.3 `os_msec_to_ticks()`

```
uint32_t os_msec_to_ticks (
    uint32_t msec )
```

Convert milliseconds to OS ticks

This function converts the given millisecond value to the number of OS ticks.

This is useful as functions like [os\\_thread\\_sleep\(\)](#) accept only ticks as input.

#### Parameters

in	<i>msec</i>	Milliseconds
----	-------------	--------------

#### Returns

Number of OS ticks corresponding to msec

### 5.11.3.4 `os_ticks_to_msec()`

```
unsigned long os_ticks_to_msec (
    unsigned long ticks )
```

Convert ticks to milliseconds

This function converts the given ticks value to milliseconds. This is useful as some functions, like [os\\_ticks\\_get\(\)](#), return values in units of OS ticks.

#### Parameters

in	<i>ticks</i>	OS ticks
----	--------------	----------

#### Returns

Number of milliseconds corresponding to ticks

### 5.11.3.5 `os_thread_create()`

```
int os_thread_create (
    os_thread_t * thandle,
    const char * name,
    void(*) (os_thread_arg_t arg) main_func,
    void * arg,
    os_thread_stack_t * stack,
    int prio )
```

### Create new thread

This function starts a new thread. The new thread starts execution by invoking `main_func()`. The parameter `arg` is passed as the sole argument of `main_func()`.

After finishing execution, the new thread should either call:

- `os_thread_self_complete()` to suspend itself OR
- `os_thread_delete()` to delete itself

Failing to do this and just returning from `main_func()` will result in undefined behavior.

#### Parameters

out	<i>thandle</i>	Pointer to a thread handle
in	<i>name</i>	Name of the new thread. A copy of this string will be made by the OS for itself. The maximum name length is defined by the macro <code>configMAX_TASK_NAME_LEN</code> in FreeRTOS header file . Any name length above it will be truncated.
in	<i>main_func</i>	Function pointer to new thread function
in	<i>arg</i>	The sole argument passed to <code>main_func()</code>
in	<i>stack</i>	A pointer to initialized object of type <code>os_thread_stack_t</code> . The object should be created and initialized using <code>os_thread_stack_define()</code> .
in	<i>prio</i>	The priority of the new thread. One value among <code>OS_PRIO_0</code> , <code>OS_PRIO_1</code> , <code>OS_PRIO_2</code> , <code>OS_PRIO_3</code> and <code>OS_PRIO_4</code> should be passed. <code>OS_PRIO_0</code> represents the highest priority and <code>OS_PRIO_4</code> represents the lowest priority.

#### Returns

WM\_SUCCESS if thread was created successfully  
-WM\_FAIL if thread creation failed

#### 5.11.3.6 os\_thread\_delete()

```
int os_thread_delete (
    os_thread_t * thandle )
```

### Terminate a thread

This function deletes a thread. The task being deleted will be removed from all ready, blocked, suspended and event lists.

#### Parameters

in	<i>thandle</i>	Pointer to the thread handle of the thread to be deleted. If self deletion is required NULL should be passed.
----	----------------	---

**Returns**

WM\_SUCCESS if operation success  
 -WM\_FAIL if operation fails

**5.11.3.7 os\_thread\_sleep()**

```
void os_thread_sleep (
    uint32_t ticks )
```

Sleep for specified number of OS ticks

This function causes the calling thread to sleep and block for the given number of OS ticks. The actual time that the task remains blocked depends on the tick rate. The function [os\\_msec\\_to\\_ticks\(\)](#) is provided to convert from real-time to ticks.

Any other thread can wake up this task specifically using the API [os\\_thread\\_wait\\_abort\(\)](#)

**Parameters**

in	<i>ticks</i>	Number of ticks to sleep
----	--------------	--------------------------

**Returns**

0 If slept for given ticks or more  
 Positive value if woken up before given ticks.

**Note**

The value returned is amount of ticks left before the task was to be originally scheduled to be woken up. So if sleep was for 10 ticks and the task is woken up after 8 ticks then 2 will be returned.

**5.11.3.8 os\_thread\_self\_complete()**

```
void os_thread_self_complete (
    os_thread_t * thandle )
```

Suspend the given thread

- The function [os\\_thread\\_self\\_complete\(\)](#) will **permanently** suspend the given thread. Passing NULL will suspend the current thread. This function never returns.
- The thread continues to consume system resources. To delete the thread the function [os\\_thread\\_delete\(\)](#) needs to be called separately.

## Parameters

in	<i>thandle</i>	Pointer to thread handle
----	----------------	--------------------------

## 5.11.3.9 os\_queue\_create()

```
int os_queue_create (
    os_queue_t * qhandle,
    const char * name,
    int msgsize,
    os_queue_pool_t * poolname )
```

Create an OS queue

This function creates a new queue instance. This allocates the storage required by the new queue and returns a handle for the queue.

## Parameters

out	<i>qhandle</i>	Pointer to the handle of the newly created queue
in	<i>name</i>	String specifying the name of the queue
in	<i>msgsize</i>	The number of bytes each item in the queue will require. Items are queued by copy, not by reference, so this is the number of bytes that will be copied for each posted item. Each item on the queue must be the same size.
in	<i>poolname</i>	The object of the type <a href="#">os_queue_pool_t</a> . The helper macro <a href="#">os_queue_pool_define()</a> helps to define this object.

## Returns

WM\_SUCCESS if queue creation was successful  
 -WM\_FAIL if queue creation failed

## 5.11.3.10 os\_queue\_send()

```
int os_queue_send (
    os_queue_t * qhandle,
    const void * msg,
    unsigned long wait )
```

Post an item to the back of the queue.

This function posts an item to the back of a queue. The item is queued by copy, not by reference. This function can also be called from an interrupt service routine.

## Parameters

in	<i>qhandle</i>	Pointer to the handle of the queue
----	----------------	------------------------------------

## Parameters

in	<i>msg</i>	A pointer to the item that is to be placed on the queue. The size of the items the queue will hold was defined when the queue was created, so this many bytes will be copied from msg into the queue storage area.
in	<i>wait</i>	The maximum amount of time, in OS ticks, the task should block waiting for space to become available on the queue, should it already be full. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

## Returns

WM\_SUCCESS if send operation was successful  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL if send operation failed

## 5.11.3.11 os\_queue\_rcv()

```
int os_queue_rcv (
    os_queue_t * qhandle,
    void * msg,
    unsigned long wait )
```

Receive an item from queue

This function receives an item from a queue. The item is received by copy so a buffer of adequate size must be provided. The number of bytes copied into the buffer was defined when the queue was created.

## Parameters

in	<i>qhandle</i>	Pointer to handle of the queue
out	<i>msg</i>	Pointer to the buffer into which the received item will be copied. The size of the items in the queue was defined when the queue was created. This pointer should point to a buffer as many bytes in size.
in	<i>wait</i>	The maximum amount of time, in OS ticks, the task should block waiting for messages to arrive on the queue, should it already be empty. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

## Returns

WM\_SUCCESS if receive operation was successful  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL if receive operation failed

## Note

This function must not be used in an interrupt service routine.

#### 5.11.3.12 os\_queue\_delete()

```
int os_queue_delete (
    os_queue_t * qhandle )
```

##### Delete queue

This function deletes a queue. It frees all the memory allocated for storing of items placed on the queue.

##### Parameters

in	<i>qhandle</i>	Pointer to handle of the queue to be deleted.
----	----------------	---

##### Returns

Currently always returns WM\_SUCCESS

#### 5.11.3.13 os\_queue\_get\_msgs\_waiting()

```
int os_queue_get_msgs_waiting (
    os_queue_t * qhandle )
```

Return the number of messages stored in queue.

##### Parameters

in	<i>qhandle</i>	Pointer to handle of the queue to be queried.
----	----------------	---

##### Returns

Number of items in the queue  
-WM\_E\_INVALID if invalid parameters are passed

#### 5.11.3.14 os\_setup\_idle\_function()

```
int os_setup_idle_function (
    void(*) (void) func )
```

##### Setup idle function

This function sets up a callback function which will be called whenever the system enters the idle thread context.

##### Parameters

in	<i>func</i>	The callback function
----	-------------	-----------------------

**Returns**

WM\_SUCCESS on success  
-WM\_FAIL on error

**5.11.3.15 os\_setup\_tick\_function()**

```
int os_setup_tick_function (
    void(*) (void) func )
```

**Setup tick function**

This function sets up a callback function which will be called on every SysTick interrupt.

**Parameters**

in	<i>func</i>	The callback function
----	-------------	-----------------------

**Returns**

WM\_SUCCESS on success  
-WM\_FAIL on error

**5.11.3.16 os\_remove\_idle\_function()**

```
int os_remove_idle_function (
    void(*) (void) func )
```

**Remove idle function**

This function removes an idle callback function that was registered previously using [os\\_setup\\_idle\\_function\(\)](#).

**Parameters**

in	<i>func</i>	The callback function
----	-------------	-----------------------

**Returns**

WM\_SUCCESS on success  
-WM\_FAIL on error

**5.11.3.17 os\_remove\_tick\_function()**

```
int os_remove_tick_function (
    void(*) (void) func )
```



Remove tick function

This function removes a tick callback function that was registered previously using [os\\_setup\\_tick\\_function\(\)](#).

#### Parameters

in	<i>func</i>	Callback function
----	-------------	-------------------

#### Returns

WM\_SUCCESS on success  
-WM\_FAIL on error

#### 5.11.3.18 os\_mutex\_create()

```
int os_mutex_create (
    os_mutex_t * mhandle,
    const char * name,
    int flags )
```

Create mutex

This function creates a mutex.

#### Parameters

out	<i>mhandle</i>	Pointer to a mutex handle
in	<i>name</i>	Name of the mutex
in	<i>flags</i>	Priority inheritance selection. Valid options are <a href="#">OS_MUTEX_INHERIT</a> or <a href="#">OS_MUTEX_NO_INHERIT</a> .

#### Note

Currently non-inheritance in mutex is not supported.

#### Returns

WM\_SUCCESS on success  
-WM\_FAIL on error

#### 5.11.3.19 os\_mutex\_get()

```
int os_mutex_get (
    os_mutex_t * mhandle,
    unsigned long wait )
```

Acquire mutex

This function acquires a mutex. Only one thread can acquire a mutex at any given time. If already acquired the callers will be blocked for the specified time duration.

## Parameters

in	<i>mhandle</i>	Pointer to mutex handle
in	<i>wait</i>	The maximum amount of time, in OS ticks, the task should block waiting for the mutex to be acquired. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

## Returns

WM\_SUCCESS when mutex is acquired  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL on failure

## 5.11.3.20 os\_mutex\_put()

```
int os_mutex_put (
    os_mutex_t * mhandle )
```

## Release mutex

This function releases a mutex previously acquired using [os\\_mutex\\_get\(\)](#).

## Note

The mutex should be released from the same thread context from which it was acquired. If you wish to acquire and release in different contexts, please use [os\\_semaphore\\_get\(\)](#) and [os\\_semaphore\\_put\(\)](#) variants.

## Parameters

in	<i>mhandle</i>	Pointer to the mutex handle
----	----------------	-----------------------------

## Returns

WM\_SUCCESS when mutex is released  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL on failure

## 5.11.3.21 os\_recursive\_mutex\_create()

```
int os_recursive_mutex_create (
    os_mutex_t * mhandle,
    const char * name )
```

## Create recursive mutex

This function creates a recursive mutex. A mutex used recursively can be 'get' repeatedly by the owner. The mutex doesn't become available again until the owner has called [os\\_recursive\\_mutex\\_put\(\)](#) for each successful 'get' request.

**Note**

This type of mutex uses a priority inheritance mechanism so a task 'get'ing a mutex MUST ALWAYS 'put' the mutex back once no longer required.

**Parameters**

out	<i>mhandle</i>	Pointer to a mutex handle
in	<i>name</i>	Name of the mutex as NULL terminated string

**Returns**

WM\_SUCCESS on success  
 -WM\_EINVAL on invalid parameter.  
 -WM\_FAIL on error

**5.11.3.22 os\_recursive\_mutex\_get()**

```
int os_recursive_mutex_get (
    os_mutex_t * mhandle,
    unsigned long wait )
```

Get recursive mutex

This function recursively obtains, or 'get's, a mutex. The mutex must have previously been created using a call to [os\\_recursive\\_mutex\\_create\(\)](#).

**Parameters**

in	<i>mhandle</i>	Pointer to mutex handle obtained from <a href="#">os_recursive_mutex_create()</a> .
in	<i>wait</i>	The maximum amount of time, in OS ticks, the task should block waiting for the mutex to be acquired. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait for portMAX_DELAY (0xffffffff) or return immediately.

**Returns**

WM\_SUCCESS when recursive mutex is acquired  
 -WM\_FAIL on failure

**5.11.3.23 os\_recursive\_mutex\_put()**

```
int os_recursive_mutex_put (
    os_mutex_t * mhandle )
```

Put recursive mutex

This function recursively releases, or 'give's, a mutex. The mutex must have previously been created using a call to [os\\_recursive\\_mutex\\_create\(\)](#)

**Parameters**

in	<i>mhandle</i>	Pointer to the mutex handle
----	----------------	-----------------------------

**Returns**

WM\_SUCCESS when mutex is released  
-WM\_FAIL on failure

**5.11.3.24 os\_mutex\_delete()**

```
int os_mutex_delete (
    os_mutex_t * mhandle )
```

**Delete mutex**

This function deletes a mutex.

**Parameters**

in	<i>mhandle</i>	Pointer to the mutex handle
----	----------------	-----------------------------

**Note**

A mutex should not be deleted if other tasks are blocked on it.

**Returns**

WM\_SUCCESS on success

**5.11.3.25 os\_event\_notify\_get()**

```
int os_event_notify_get (
    unsigned long wait_time )
```

**Wait for task notification**

This function waits for task notification from other task or interrupt context. This is similar to binary semaphore, but uses less RAM and much faster than semaphore mechanism

**Parameters**

in	<i>wait_time</i>	Timeout specified in no. of OS ticks
----	------------------	--------------------------------------

**Returns**

WM\_SUCCESS when notification is successful  
 -WM\_FAIL on failure or timeout

**5.11.3.26 os\_event\_notify\_put()**

```
int os_event_notify_put (
    os_thread_t task )
```

**Give task notification**

This function gives task notification so that waiting task can be unblocked. This is similar to binary semaphore, but uses less RAM and much faster than semaphore mechanism

**Parameters**

in	<i>task</i>	Task handle to be notified
----	-------------	----------------------------

**Returns**

WM\_SUCCESS when notification is successful  
 -WM\_FAIL on failure or timeout

**5.11.3.27 os\_semaphore\_create()**

```
int os_semaphore_create (
    os_semaphore_t * mhandle,
    const char * name )
```

**Create binary semaphore**

This function creates a binary semaphore. A binary semaphore can be acquired by only one entity at a given time.

**Parameters**

out	<i>mhandle</i>	Pointer to a semaphore handle
in	<i>name</i>	Name of the semaphore

**Returns**

WM\_SUCCESS on success  
 -WM\_FAIL on error

### 5.11.3.28 os\_semaphore\_create\_counting()

```
int os_semaphore_create_counting (
    os_semaphore_t * mhandle,
    const char * name,
    unsigned long maxcount,
    unsigned long initcount )
```

#### Create counting semaphore

This function creates a counting semaphore. A counting semaphore can be acquired 'count' number of times at a given time.

#### Parameters

out	<i>mhandle</i>	Pointer to a semaphore handle
in	<i>name</i>	Name of the semaphore
in	<i>maxcount</i>	The maximum count value that can be reached. When the semaphore reaches this value it can no longer be 'put'
in	<i>initcount</i>	The count value assigned to the semaphore when it is created. For e.g. If '0' is passed, then <a href="#">os_semaphore_get()</a> will block until some other thread does an <a href="#">os_semaphore_put()</a> .

#### Returns

WM\_SUCCESS on success  
-WM\_FAIL on error

### 5.11.3.29 os\_semaphore\_get()

```
int os_semaphore_get (
    os_semaphore_t * mhandle,
    unsigned long wait )
```

#### Acquire semaphore

This function acquires a semaphore. At a given time, a binary semaphore can be acquired only once, while a counting semaphore can be acquired as many as 'count' number of times. Once this condition is reached, the other callers of this function will be blocked for the specified time duration.

#### Parameters

in	<i>mhandle</i>	Pointer to a semaphore handle
in	<i>wait</i>	The maximum amount of time, in OS ticks, the task should block waiting for the semaphore to be acquired. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

**Returns**

WM\_SUCCESS when semaphore is acquired  
-WM\_EINVAL if invalid parameters are passed  
-WM\_FAIL on failure

**5.11.3.30 os\_semaphore\_put()**

```
int os_semaphore_put (
    os_semaphore_t * mhandle )
```

Release semaphore

This function releases a semaphore previously acquired using [os\\_semaphore\\_get\(\)](#).

**Note**

This function can also be called from interrupt-context.

**Parameters**

in	<i>mhandle</i>	Pointer to a semaphore handle
----	----------------	-------------------------------

**Returns**

WM\_SUCCESS when semaphore is released  
-WM\_EINVAL if invalid parameters are passed  
-WM\_FAIL on failure

**5.11.3.31 os\_semaphore\_getcount()**

```
int os_semaphore_getcount (
    os_semaphore_t * mhandle )
```

Get semaphore count

This function returns the current value of a semaphore.

**Parameters**

in	<i>mhandle</i>	Pointer to a semaphore handle
----	----------------	-------------------------------

**Returns**

current value of the semaphore

### 5.11.3.32 os\_semaphore\_delete()

```
int os_semaphore_delete (
    os_semaphore_t * mhandle )
```

Delete a semaphore

This function deletes the semaphore.

#### Parameters

in	<i>mhandle</i>	Pointer to a semaphore handle
----	----------------	-------------------------------

#### Note

Do not delete a semaphore that has tasks blocked on it (tasks that are in the Blocked state waiting for the semaphore to become available)

#### Returns

WM\_SUCCESS on success

### 5.11.3.33 os\_rwlock\_create()

```
int os_rwlock_create (
    os_rw_lock_t * plock,
    const char * mutex_name,
    const char * lock_name )
```

Create reader-writer lock

This function creates a reader-writer lock.

#### Parameters

in	<i>lock</i>	Pointer to a reader-writer lock handle
in	<i>mutex_name</i>	Name of the mutex
in	<i>lock_name</i>	Name of the lock

#### Returns

WM\_SUCCESS on success

-WM\_FAIL on error



5.11.3.34 `os_rwlock_delete()`

```
void os_rwlock_delete (
    os_rwlock_t * lock )
```

Delete a reader-write lock

This function deletes a reader-writer lock.

## Parameters

in	<i>lock</i>	Pointer to the reader-writer lock handle
----	-------------	--

5.11.3.35 `os_rwlock_write_lock()`

```
int os_rwlock_write_lock (
    os_rwlock_t * lock,
    unsigned int wait_time )
```

Acquire writer lock

This function acquires a writer lock. While readers can acquire the lock on a sharing basis, writers acquire the lock in an exclusive manner.

## Parameters

in	<i>lock</i>	Pointer to the reader-writer lock handle
in	<i>wait_time</i>	The maximum amount of time, in OS ticks, the task should block waiting for the lock to be acquired. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

## Returns

WM\_SUCCESS on success  
-WM\_FAIL on error

5.11.3.36 `os_rwlock_write_unlock()`

```
void os_rwlock_write_unlock (
    os_rwlock_t * lock )
```

Release writer lock

This function releases a writer lock previously acquired using [os\\_rwlock\\_write\\_lock\(\)](#).

## Parameters

in	<i>lock</i>	Pointer to the reader-writer lock handle
----	-------------	--

5.11.3.37 `os_rwlock_read_lock()`

```
int os_rwlock_read_lock (
    os_rwlock_t * lock,
    unsigned int wait_time )
```

## Acquire reader lock

This function acquires a reader lock. While readers can acquire the lock on a sharing basis, writers acquire the lock in an exclusive manner.

## Parameters

in	<i>lock</i>	pointer to the reader-writer lock handle
in	<i>wait_time</i>	The maximum amount of time, in OS ticks, the task should block waiting for the lock to be acquired. The function <a href="#">os_msec_to_ticks()</a> can be used to convert from real-time to OS ticks. The special values <a href="#">OS_WAIT_FOREVER</a> and <a href="#">OS_NO_WAIT</a> are provided to respectively wait infinitely or return immediately.

## Returns

WM\_SUCCESS on success  
-WM\_FAIL on error

5.11.3.38 `os_rwlock_read_unlock()`

```
int os_rwlock_read_unlock (
    os_rwlock_t * lock )
```

## Release reader lock

This function releases a reader lock previously acquired using [os\\_rwlock\\_read\\_lock\(\)](#).

## Parameters

in	<i>lock</i>	pointer to the reader-writer lock handle
----	-------------	--

## Returns

WM\_SUCCESS if unlock operation successful.  
-WM\_FAIL if unlock operation failed.

## 5.11.3.39 os\_timer\_create()

```
int os_timer_create (
    os_timer_t * timer_t,
    const char * name,
    os_timer_tick ticks,
    void(*) (os_timer_arg_t) call_back,
    void * cb_arg,
    os_timer_reload_t reload,
    os_timer_activate_t activate )
```

Create timer

This function creates a timer.

## Parameters

out	<i>timer_t</i>	Pointer to the timer handle
in	<i>name</i>	Name of the timer
in	<i>ticks</i>	Period in ticks
in	<i>call_back</i>	Timer expire callback function
in	<i>cb_arg</i>	Timer callback data
in	<i>reload</i>	Reload Options, valid values include <a href="#">OS_TIMER_ONE_SHOT</a> or <a href="#">OS_TIMER_PERIODIC</a> .
in	<i>activate</i>	Activate Options, valid values include <a href="#">OS_TIMER_AUTO_ACTIVATE</a> or <a href="#">OS_TIMER_NO_ACTIVATE</a>

## Returns

WM\_SUCCESS if timer created successfully

-WM\_FAIL if timer creation fails

## 5.11.3.40 os\_timer\_activate()

```
int os_timer_activate (
    os_timer_t * timer_t )
```

Activate timer

This function activates (or starts) a timer that was previously created using [os\\_timer\\_create\(\)](#). If the timer had already started and was already in the active state, then this call is equivalent to [os\\_timer\\_reset\(\)](#).

## Parameters

in	<i>timer_t</i>	Pointer to a timer handle
----	----------------	---------------------------

**Returns**

WM\_SUCCESS if timer activated successfully  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL if timer fails to activate

**5.11.3.41 os\_timer\_change()**

```
int os_timer_change (
    os_timer_t * timer_t,
    os_timer_tick ntime,
    os_timer_tick block_time )
```

**Change timer period**

This function changes the period of a timer that was previously created using os\_time\_create(). This function changes the period of an active or dormant state timer.

**Parameters**

in	<i>timer_t</i>	Pointer to a timer handle
in	<i>ntime</i>	Time in ticks after which the timer will expire
in	<i>block_time</i>	This option is currently not supported

**Returns**

WM\_SUCCESS on success  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL on failure

**5.11.3.42 os\_timer\_is\_running()**

```
bool os_timer_is_running (
    os_timer_t * timer_t )
```

**Check the timer active state**

This function checks if the timer is in the active or dormant state. A timer is in the dormant state if (a) it has been created but not started, or (b) it has expired and a one-shot timer.

**Parameters**

in	<i>timer_t</i>	Pointer to a timer handle
----	----------------	---------------------------

**Returns**

true if timer is active  
false if time is not active

**5.11.3.43 os\_timer\_get\_context()**

```
void* os_timer_get_context (
    os_timer_t * timer_t )
```

**Get the timer context**

This function helps to retrieve the timer context i.e. 'cb\_arg' passed to [os\\_timer\\_create\(\)](#).

**Parameters**

in	<i>timer_t</i>	Pointer to timer handle. The timer handle is received in the timer callback.
----	----------------	--

**Returns**

The timer context i.e. the callback argument passed to [os\\_timer\\_create\(\)](#).

**5.11.3.44 os\_timer\_reset()**

```
int os_timer_reset (
    os_timer_t * timer_t )
```

**Reset timer**

This function resets a timer that was previously created using [os\\_timer\\_create\(\)](#). If the timer had already been started and was already in the active state, then this call will cause the timer to re-evaluate its expiry time so that it is relative to when [os\\_timer\\_reset\(\)](#) was called. If the timer was in the dormant state then this call behaves in the same way as [os\\_timer\\_activate\(\)](#).

**Parameters**

in	<i>timer_t</i>	Pointer to a timer handle
----	----------------	---------------------------

**Returns**

WM\_SUCCESS on success  
-WM\_EINVAL if invalid parameters are passed  
-WM\_FAIL on failure

5.11.3.45 `os_timer_deactivate()`

```
int os_timer_deactivate (
    os_timer_t * timer_t )
```

Deactivate timer

This function deactivates (or stops) a timer that was previously started.

## Parameters

in	<i>timer_t</i>	handle populated by <a href="#">os_timer_create()</a>
----	----------------	---

## Returns

WM\_SUCCESS on success  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL on failure

5.11.3.46 `os_timer_delete()`

```
int os_timer_delete (
    os_timer_t * timer_t )
```

Delete timer

This function deletes a timer.

## Parameters

in	<i>timer_t</i>	Pointer to a timer handle
----	----------------	---------------------------

## Returns

WM\_SUCCESS on success  
 -WM\_E\_INVALID if invalid parameters are passed  
 -WM\_FAIL on failure

5.11.3.47 `os_mem_alloc()`

```
void* os_mem_alloc (
    size_t size )
```

Allocate memory

This function allocates memory dynamically.

**Parameters**

in	size	Size of the memory to be allocated
----	------	------------------------------------

**Returns**

Pointer to the allocated memory  
NULL if allocation fails

**5.11.3.48 os\_mem\_calloc()**

```
void* os_mem_calloc (
    size_t size )
```

Allocate memory and zero it

This function allocates memory dynamically and sets the memory contents to zero.

**Parameters**

in	size	Size of the memory to be allocated
----	------	------------------------------------

**Returns**

Pointer to the allocated memory  
NULL if allocation fails

**5.11.3.49 os\_mem\_free()**

```
void os_mem_free (
    void * ptr )
```

**Free Memory**

This function frees dynamically allocated memory using any of the dynamic allocation primitives.

**Parameters**

in	ptr	Pointer to the memory to be freed
----	-----	-----------------------------------

**5.11.3.50 os\_disable\_all\_interrupts()**

```
void os_disable_all_interrupts (
    void )
```

Disables all interrupts at NVIC level

**5.11.3.51 os\_enable\_all\_interrupts()**

```
void os_enable_all_interrupts (
    void )
```

Enable all interrupts at NVIC level

**5.11.4 Macro Documentation****5.11.4.1 os\_thread\_relinquish**

```
#define os_thread_relinquish( ) taskYIELD()
```

Get the current value of free running microsecond counter

**Note**

This will wraparound after CNTMAX and the caller is expected to take care of this.

**Returns**

The current value of microsecond counter. Force a context switch

**5.11.4.2 os\_ticks\_to\_unblock**

```
#define os_ticks_to_unblock( ) xTaskGetUnblockTime()
```

Get ticks to next thread wakeup

**5.11.4.3 os\_thread\_stack\_define**

```
#define os_thread_stack_define(
    stackname,
    stacksize ) os_thread_stack_t stackname = {(stacksize) / (sizeof(portSTACK_TY↵
    PE)) }
```

Helper macro to define the stack size (in bytes) before a new thread is created using the function [os\\_thread\\_create\(\)](#).



#### 5.11.4.4 os\_queue\_pool\_define

```
#define os_queue_pool_define(  
    poolname,  
    poolsize ) os_queue_pool_t poolname = {poolsize};
```

Define OS Queue pool

This macro helps define the name and size of the queue to be created using the function [os\\_queue\\_create\(\)](#).

#### 5.11.4.5 OS\_WAIT\_FOREVER

```
#define OS_WAIT_FOREVER portMAX_DELAY
```

Wait Forever

#### 5.11.4.6 OS\_NO\_WAIT

```
#define OS_NO_WAIT 0
```

Do Not Wait

#### 5.11.4.7 OS\_MUTEX\_INHERIT

```
#define OS_MUTEX_INHERIT 1
```

Priority Inheritance Enabled

#### 5.11.4.8 OS\_MUTEX\_NO\_INHERIT

```
#define OS_MUTEX_NO_INHERIT 0
```

Priority Inheritance Disabled

#### 5.11.4.9 os\_get\_runtime\_stats

```
#define os_get_runtime_stats(  
    __buff__ ) vTaskGetRunTimeStats(__buff__)
```

Get ASCII formatted run time statistics

Please ensure that your buffer is big enough for the formatted data to fit. Failing to do this may cause memory data corruption.

### 5.11.5 Typedef Documentation

### 5.11.5.1 cb\_fn

```
typedef int (* cb_fn) (os_rw_lock_t *plock, unsigned int wait_time)
```

This is prototype of reader callback

## 5.11.6 Enumeration Type Documentation

### 5.11.6.1 os\_timer\_reload\_t

```
enum os_timer_reload_t
```

OS Timer reload Options

Enumerator

OS_TIMER_ONE_SHOT	Create one shot timer. Timer will be in the dormant state after it expires.
OS_TIMER_PERIODIC	Create a periodic timer. Timer will auto-reload after it expires.

### 5.11.6.2 os\_timer\_activate\_t

```
enum os_timer_activate_t
```

OS Timer Activate Options

Enumerator

OS_TIMER_AUTO_ACTIVATE	Start the timer on creation.
OS_TIMER_NO_ACTIVATE	Do not start the timer on creation.

## 5.12 wm\_utils.h File Reference

Utility functions.

### 5.12.1 Detailed Description

Collection of some common helper functions

## 5.12.2 Function Documentation

### 5.12.2.1 hex2bin()

```
static unsigned int hex2bin (
    const uint8_t * ibuf,
    uint8_t * obuf,
    unsigned max_olen ) [inline], [static]
```

Convert a given hex string to a equivalent binary representation.

E.g. If your input string of 4 bytes is {'F', 'F', 'F', 'F'} the output string will be of 2 bytes {255, 255} or to put the same in other way {0xFF, 0xFF}

Note that hex2bin is not the same as strtoul as the latter will properly return the integer in the correct machine binary format viz. little endian. hex2bin however does only in-place like replacement of two ASCII characters to one binary number taking 1 byte in memory.

#### Parameters

in	<i>ibuf</i>	input buffer
out	<i>obuf</i>	output buffer
in	<i>max_olen</i>	Maximum output buffer length

#### Returns

length of the binary string

### 5.12.2.2 bin2hex()

```
void bin2hex (
    uint8_t * src,
    char * dest,
    unsigned int src_len,
    unsigned int dest_len )
```

Convert given binary array to equivalent hex representation.

#### Parameters

in	<i>src</i>	Input buffer
out	<i>dest</i>	Output buffer
in	<i>src_len</i>	Length of the input buffer
in	<i>dest_len</i>	Length of the output buffer

**Returns**

void

**5.12.2.3 random\_register\_handler()**

```
int random_register_handler (
    random_hdlr_t func )
```

Register a random entropy generator handler

This API allows applications to register their own random entropy generator handlers that will be internally used by [get\\_random\\_sequence\(\)](#) to add even more randomization to the byte stream generated by it.

**Parameters**

in	func	Function pointer of type <a href="#">random_hdlr_t</a>
----	------	--

**Returns**

WM\_SUCCESS if successful  
-WM\_E\_NOSPC if there is no space available for additional handlers

**5.12.2.4 random\_unregister\_handler()**

```
int random_unregister_handler (
    random_hdlr_t func )
```

Un-register a random entropy generator handler

This API can be used to un-register a handler registered using [random\\_register\\_handler\(\)](#)

**Parameters**

in	func	Function pointer of type <a href="#">random_hdlr_t</a> used during registering
----	------	--

**Returns**

WM\_SUCCESS if successful  
-WM\_E\_INVALID if the passed pointer is invalid

## 5.12.2.5 random\_register\_seed\_handler()

```
int random_register_seed_handler (
    random_hdlr_t func )
```

Register a random seed generator handler

For getting better random numbers, the initial seed (ideally required only once on every boot) should also be random. This API allows applications to register their own seed generators. Applications can use any logic such that a different seed is generated every time. A sample seed generator which uses a combination of DAC (generating random noise) and ADC (that internally samples the random noise) along with the flash id has already been provided. Please have a look at [sample\\_initialise\\_random\\_seed\(\)](#).

The seed generator handler is called only once by the [get\\_random\\_sequence\(\)](#) function. Applications can also explicitly initialize the seed by calling [random\\_initialize\\_seed\(\)](#) after registering a handler.

## Parameters

in	func	Function pointer of type <a href="#">random_hdlr_t</a>
----	------	--

## Returns

WM\_SUCCESS if successful  
 -WM\_E\_NOSPC if there is no space available for additional handlers

## 5.12.2.6 random\_unregister\_seed\_handler()

```
int random_unregister_seed_handler (
    random_hdlr_t func )
```

Un-register a random seed generator handler

This API can be used to un-register a handler registered using [random\\_register\\_seed\\_handler\(\)](#)

## Parameters

in	func	Function pointer of type <a href="#">random_hdlr_t</a> used during registering
----	------	--

## Returns

WM\_SUCCESS if successful  
 -WM\_E\_INVALID if the passed pointer is invalid

## 5.12.2.7 random\_initialize\_seed()

```
void random_initialize_seed (
    void )
```

Initialize the random number generator's seed

The [get\\_random\\_sequence\(\)](#) uses a random number generator that is initialized with a seed when [get\\_random\\_sequence\(\)](#) is called for the first time. The handlers registered using [random\\_register\\_seed\\_handler\(\)](#) are used to generate the seed. If an application wants to explicitly initialize the seed, this API can be used. The seed will then not be re-initialized in [get\\_random\\_sequence\(\)](#).

#### 5.12.2.8 sample\_initialise\_random\_seed()

```
uint32_t sample_initialise_random_seed (
    void )
```

Sample random seed generator

This is a sample random seed generator handler that can be registered using [random\\_register\\_seed\\_handler\(\)](#) to generate a random seed. This uses a combination of DAC (generating random noise) and ADC (that internally samples the random noise) along with the flash id to generate a seed. It is recommended to register this handler and immediately call [random\\_initialize\\_seed\(\)](#) before executing any other application code, especially if the application is going to use ADC/DAC for its own purpose.

#### Returns

Random seed

#### 5.12.2.9 get\_random\_sequence()

```
void get_random_sequence (
    void * buf,
    unsigned int size )
```

Generate random sequence of bytes

This function generates random sequence of bytes in the user provided buffer.

#### Parameters

out	<i>buf</i>	The buffer to be populated with random data
in	<i>size</i>	The number of bytes of the random sequence required

#### 5.12.2.10 strdup()

```
char* strdup (
    const char * s )
```

Returns a pointer to a new string which is a duplicate of the input string *s*. Memory for the new string is obtained allocated by the function.

It is caller's responsibility to free the memory after its use.

## Parameters

in	s	Pointer to string to be duplicated
----	---	------------------------------------

## Returns

Pointer to newly allocated string which is duplicate of input string  
 NULL on error

## 5.12.2.11 soft\_crc32()

```
uint32_t soft_crc32 (
    const void * data__,
    int data_size,
    uint32_t crc )
```

Calculate CRC32 using software algorithm

## Precondition

soft\_crc32\_init()

[soft\\_crc32\(\)](#) allows the user to calculate CRC32 values of arbitrary sized buffers across multiple calls.

## Parameters

in	<i>data__</i>	Input buffer over which CRC32 is calculated.
in	<i>data_size</i>	Length of the input buffer.
in	<i>crc</i>	Previous CRC32 value used as starting point for given buffer calculation.

## Returns

Calculated CRC32 value

## 5.12.2.12 fill\_sequential\_pattern()

```
void fill_sequential_pattern (
    void * buffer,
    int size,
    uint8_t first_byte )
```

Fill the given buffer with a sequential pattern starting from given byte.

For example, if the 'first\_byte' is 0x45 and buffer size of 5 then buffer will be set to {0x45, 0x46, 0x47, 0x48, 0x49}

**Parameters**

in	<i>buffer</i>	The pattern will be set to this buffer.
in	<i>size</i>	Number of pattern bytes to be written to the buffer.
in	<i>first_byte</i>	This is the value of first byte in the sequential pattern.

**Returns**

void

**5.12.2.13 verify\_sequential\_pattern()**

```
bool verify_sequential_pattern (
    const void * buffer,
    int size,
    uint8_t first_byte )
```

Verify if the the given buffer has a sequential pattern starting from given byte.

For example, if the 'first\_byte' is 0x45 and buffer size of 5 then buffer will be verified for presence of {0x45, 0x46, 0x47, 0x48, 0x49}

**Parameters**

in	<i>buffer</i>	The pattern will be verified from this buffer.
in	<i>size</i>	Number of pattern bytes to be verified from the buffer.
in	<i>first_byte</i>	This is the value of first byte in the sequential pattern.

**Returns**

'true' If verification successful.  
 'false' If verification fails.

**5.12.3 Macro Documentation****5.12.3.1 dump\_hex**

```
#define dump_hex(
    ... )
```

**Value:**

```
do
{
} while (0)
```



### 5.12.3.2 dump\_hex\_ascii

```
#define dump_hex_ascii(  
    ... )
```

**Value:**

```
do  
{  
    while (0)
```

### 5.12.3.3 dump\_ascii

```
#define dump_ascii(  
    ... )
```

**Value:**

```
do  
{  
    while (0)
```

### 5.12.3.4 print\_ascii

```
#define print_ascii(  
    ... )
```

**Value:**

```
do  
{  
    while (0)
```

### 5.12.3.5 dump\_json

```
#define dump_json(  
    ... )
```

**Value:**

```
do  
{  
    while (0)
```

## 5.12.4 Typedef Documentation

### 5.12.4.1 random\_hdlr\_t

```
typedef uint32_t(* random_hdlr_t) (void)
```

Function prototype for a random entropy/seed generator

**Returns**

a 32bit random number

# Index

`_wifi_set_mac_addr`  
    wifi.h, 79

`ACTION_GET`  
    wlan.h, 142

`ACTION_SET`  
    wlan.h, 142

`action`  
    wifi\_mef\_entry\_t, 27

`addr1`  
    wifi\_mgmt\_frame\_t, 30

`addr2`  
    wifi\_mgmt\_frame\_t, 30

`addr3`  
    wifi\_mgmt\_frame\_t, 30

`addr4`  
    wifi\_mgmt\_frame\_t, 30

`addr_type`  
    ipv4\_config, 8

`address`  
    ipv4\_config, 8

`address_types`  
    wlan.h, 149

`ant_mode`  
    wifi\_antcfg\_t, 11

`avg_tbtt_offset`  
    wifi\_tbtt\_offset\_t, 47

`BSS_TYPE_STA`  
    wifi-decl.h, 70

`BSS_TYPE_UAP`  
    wifi-decl.h, 70

`band`  
    wifi\_scan\_result, 42

`bandcfg`  
    wifi\_remain\_on\_channel\_t, 33

`bcn_nf_avg`  
    wifi\_rssi\_info\_t, 36

`bcn_nf_last`  
    wifi\_rssi\_info\_t, 36

`bcn_rssi_avg`  
    wifi\_rssi\_info\_t, 36

`bcn_rssi_last`  
    wifi\_rssi\_info\_t, 36

`bcn_snr_avg`  
    wifi\_rssi\_info\_t, 36

`bcn_snr_last`  
    wifi\_rssi\_info\_t, 35

`beacon_period`  
    wifi\_scan\_result, 41

    wlan\_network, 57

    wlan\_scan\_result, 62

`bin2hex`  
    wm\_utils.h, 191

`bssid`  
    wifi\_scan\_params\_v2\_t, 39

    wifi\_scan\_result, 40

    wlan\_network, 55

    wlan\_scan\_result, 60

`bssid_specific`  
    wlan\_network, 56

`byte_seq`  
    wifi\_mef\_filter\_t, 28

`cb`  
    wifi\_scan\_params\_v2\_t, 39

`cb_fn`  
    wm\_os.h, 189

`ccmp`  
    wlan\_cipher, 53

`chan_desc`  
    wifi\_txpwrlimit\_config\_t, 50

`chan_freq`  
    wifi\_chan\_info\_t, 14

`chan_info`  
    wifi\_chanlist\_t, 16

`chan_list`  
    wifi\_scan\_params\_v2\_t, 39

`chan_num`  
    wifi\_chan\_info\_t, 14

    wifi\_channel\_desc\_t, 17

`chan_number`  
    wifi\_chan\_scan\_param\_set\_t, 15

    wifi\_scan\_chan\_list\_t, 37

    wifi\_scan\_channel\_list\_t, 38

`chan_scan_param`  
    wifi\_chan\_list\_param\_set\_t, 15

`chan_width`  
    wifi\_channel\_desc\_t, 17

`chanInfo`  
    wifi\_cw\_mode\_ctrl\_t, 18

`Channel`  
    wifi\_scan\_result, 41

`channel`  
    wifi\_cw\_mode\_ctrl\_t, 18

    wifi\_remain\_on\_channel\_t, 33

    wlan\_network, 55

    wlan\_scan\_result, 60

`channel_specific`  
    wlan\_network, 56

- cli.h, 63
  - cli\_init, 64
  - cli\_register\_command, 63
  - cli\_register\_commands, 64
  - cli\_stop, 64
  - cli\_unregister\_command, 64
  - cli\_unregister\_commands, 65
- cli\_command, 7
  - function, 7
  - help, 7
  - name, 7
- cli\_init
  - cli.h, 64
- cli\_register\_command
  - cli.h, 63
- cli\_register\_commands
  - cli.h, 64
- cli\_stop
  - cli.h, 64
- cli\_unregister\_command
  - cli.h, 64
- cli\_unregister\_commands
  - cli.h, 65
- config\_bands
  - wifi\_bandcfg\_t, 12
- count
  - wifi\_sta\_list\_t, 45
- country\_code
  - wifi\_domain\_param\_t, 21
- country\_code\_t
  - wifi.h, 83
- criteria
  - wififlt\_cfg\_t, 24
- current\_channel
  - wifi\_rf\_channel\_t, 34
- current\_level
  - wifi\_tx\_power\_t, 49
- data
  - wifi\_cal\_data\_t, 13
- data\_len
  - wifi\_cal\_data\_t, 13
- data\_nf\_avg
  - wifi\_rssi\_info\_t, 35
- data\_nf\_last
  - wifi\_rssi\_info\_t, 35
- data\_rate
  - wifi\_ds\_rate, 22
- data\_rssi\_avg
  - wifi\_rssi\_info\_t, 35
- data\_rssi\_last
  - wifi\_rssi\_info\_t, 35
- data\_snr\_avg
  - wifi\_rssi\_info\_t, 36
- data\_snr\_last
  - wifi\_rssi\_info\_t, 36
- dhcp-server.h, 65
  - dhcp\_enable\_dns\_server, 66
  - dhcp\_get\_ip\_from\_mac, 67
  - dhcp\_server\_lease\_timeout, 67
  - dhcp\_server\_start, 66
  - dhcp\_server\_stop, 67
  - dhcp\_stat, 68
  - dhcpcd\_cli\_init, 66
  - wm\_dhcpcd\_errno, 68
- dhcp\_enable\_dns\_server
  - dhcp-server.h, 66
- dhcp\_get\_ip\_from\_mac
  - dhcp-server.h, 67
- dhcp\_server\_lease\_timeout
  - dhcp-server.h, 67
- dhcp\_server\_start
  - dhcp-server.h, 66
- dhcp\_server\_stop
  - dhcp-server.h, 67
- dhcp\_stat
  - dhcp-server.h, 68
- dhcpcd\_cli\_init
  - dhcp-server.h, 66
- dns1
  - ipv4\_config, 9
- dns2
  - ipv4\_config, 9
- dst\_ip
  - wifi\_nat\_keep\_alive\_t, 31
  - wifi\_tcp\_keep\_alive\_t, 48
- dst\_mac
  - wifi\_nat\_keep\_alive\_t, 31
  - wifi\_tcp\_keep\_alive\_t, 48
- dst\_port
  - wifi\_nat\_keep\_alive\_t, 32
- dst\_tcp\_port
  - wifi\_tcp\_keep\_alive\_t, 48
- dtim\_period
  - wifi\_scan\_result, 41
  - wlan\_network, 57
  - wlan\_scan\_result, 62
- dump\_ascii
  - wm\_utils.h, 197
- dump\_hex
  - wm\_utils.h, 196
- dump\_hex\_ascii
  - wm\_utils.h, 196
- dump\_json
  - wm\_utils.h, 197
- duration\_id
  - wifi\_mgmt\_frame\_t, 30
- ed\_ctrl\_2g
  - wifi\_ed\_mac\_ctrl\_t, 23
- ed\_offset\_2g
  - wifi\_ed\_mac\_ctrl\_t, 23
- enable
  - wifi\_tcp\_keep\_alive\_t, 47
- evaluate\_time
  - wifi\_antcfg\_t, 11
- fill\_sequential\_pattern

- wm\_utils.h, 195
- filter\_item
  - wifi\_mef\_entry\_t, 27
- filter\_num
  - wifi\_mef\_entry\_t, 27
- first\_chan
  - wifi\_sub\_band\_set\_t, 45
- flags
  - wifi\_auto\_reconnect\_config\_t, 12
- frame\_ctrl\_flags
  - wifi\_mgmt\_frame\_t, 30
- frame\_type
  - wifi\_mgmt\_frame\_t, 30
- frm\_len
  - wifi\_mgmt\_frame\_t, 29
- function
  - cli\_command, 7
- fw\_bands
  - wifi\_bandcfg\_t, 12
- get\_random\_sequence
  - wm\_utils.h, 194
- gw
  - ipv4\_config, 8
- help
  - cli\_command, 7
- hex2bin
  - wm\_utils.h, 191
- IEEEtypes\_ADDRESS\_SIZE
  - wlan.h, 142
- IEEEtypes\_SSID\_SIZE
  - wlan.h, 142
- interval
  - wifi\_nat\_keep\_alive\_t, 31
  - wifi\_tcp\_keep\_alive\_t, 48
- ip
  - wlan\_network, 56
- iperf.h, 69
  - iperf\_cli\_deinit, 69
  - iperf\_cli\_init, 69
- iperf\_cli\_deinit
  - iperf.h, 69
- iperf\_cli\_init
  - iperf.h, 69
- ipv4
  - wlan\_ip\_config, 54
- ipv4\_config, 8
  - addr\_type, 8
  - address, 8
  - dns1, 9
  - dns2, 9
  - gw, 8
  - netmask, 9
- is\_ibss\_bit\_set
  - wifi\_scan\_result, 40
- is\_pmf\_required
  - wifi\_scan\_result, 42
- wlan\_network\_security, 58
- is\_sta\_connected
  - wlan.h, 96
- is\_sta\_ipv4\_connected
  - wlan.h, 96
- is\_uap\_started
  - wlan.h, 96
- MLAN\_MAX\_PASS\_LENGTH
  - wifi-decl.h, 70
- MLAN\_MAX\_SSID\_LENGTH
  - wifi-decl.h, 70
- MLAN\_MAX\_VER\_STR\_LEN
  - wifi-decl.h, 70
- mac
  - wifi\_mac\_addr\_t, 26
  - wifi\_sta\_info\_t, 44
- mask\_seq
  - wifi\_mef\_filter\_t, 29
- max\_keep\_alives
  - wifi\_tcp\_keep\_alive\_t, 48
- max\_power
  - wifi\_tx\_power\_t, 49
- max\_scan\_time
  - wifi\_chan\_scan\_param\_set\_t, 15
- max\_tbtt\_offset
  - wifi\_tbtt\_offset\_t, 46
- max\_tx\_pwr
  - wifi\_sub\_band\_set\_t, 46
- mcstCipher
  - wlan\_network\_security, 58
- mef\_entry
  - wififlt\_cfg\_t, 24
- mfpc
  - wlan\_network\_security, 59
- mfpr
  - wlan\_network\_security, 59
- min\_power
  - wifi\_tx\_power\_t, 49
- min\_scan\_time
  - wifi\_chan\_scan\_param\_set\_t, 15
- min\_tbtt\_offset
  - wifi\_tbtt\_offset\_t, 46
- mod\_group
  - wifi\_txpwrlimit\_entry\_t, 51
- mode
  - wifi\_cw\_mode\_ctrl\_t, 18
  - wifi\_mef\_entry\_t, 26
- name
  - cli\_command, 7
  - wlan\_network, 55
- nentries
  - wififlt\_cfg\_t, 24
- net\_configure\_address
  - wm\_net.h, 161
- net\_configure\_dns
  - wm\_net.h, 162
- net\_dhcp\_hostname\_set

wm\_net.h, 156  
net\_get\_if\_addr  
    wm\_net.h, 162  
net\_get\_if\_ip\_addr  
    wm\_net.h, 163  
net\_get\_if\_ip\_mask  
    wm\_net.h, 163  
net\_get\_if\_name  
    wm\_net.h, 162  
net\_get\_sock\_error  
    wm\_net.h, 157  
net\_get\_sta\_handle  
    wm\_net.h, 159  
net\_get\_uap\_handle  
    wm\_net.h, 160  
net\_gethostbyname  
    wm\_net.h, 157  
net\_inet\_aton  
    wm\_net.h, 157  
net\_inet\_ntoa  
    wm\_net.h, 158  
net\_interface\_dhcp\_stop  
    wm\_net.h, 161  
net\_interface\_down  
    wm\_net.h, 160  
net\_interface\_up  
    wm\_net.h, 160  
net\_ipv4stack\_init  
    wm\_net.h, 164  
net\_is\_ip\_or\_ipv6  
    wm\_net.h, 158  
net\_sock\_to\_interface  
    wm\_net.h, 159  
net\_socket\_blocking  
    wm\_net.h, 156  
net\_stat  
    wm\_net.h, 164  
net\_stop\_dhcp\_timer  
    wm\_net.h, 156  
net\_wlan\_deinit  
    wm\_net.h, 159  
net\_wlan\_init  
    wm\_net.h, 159  
netmask  
    ipv4\_config, 9  
no\_of\_chan  
    wifi\_sub\_band\_set\_t, 46  
no\_of\_channels  
    wifi\_chan\_list\_param\_set\_t, 14  
no\_of\_sub\_band  
    wifi\_domain\_param\_t, 21  
num\_byte\_seq  
    wifi\_mef\_filter\_t, 28  
num\_bytes  
    wifi\_mef\_filter\_t, 28  
num\_channels  
    wifi\_scan\_params\_v2\_t, 39  
num\_chans  
    wifi\_chanlist\_t, 16  
    wifi\_txpwrlimit\_t, 52  
num\_mask\_seq  
    wifi\_mef\_filter\_t, 29  
num\_mod\_grps  
    wifi\_txpwrlimit\_config\_t, 50  
num\_of\_chan  
    wifi\_scan\_chan\_list\_t, 37  
num\_probes  
    wifi\_scan\_params\_v2\_t, 39  
OS\_MUTEX\_INHERIT  
    wm\_os.h, 189  
OS\_MUTEX\_NO\_INHERIT  
    wm\_os.h, 189  
OS\_NO\_WAIT  
    wm\_os.h, 189  
OS\_WAIT\_FOREVER  
    wm\_os.h, 189  
offset  
    wifi\_mef\_filter\_t, 28  
os\_disable\_all\_interrupts  
    wm\_os.h, 187  
os\_enable\_all\_interrupts  
    wm\_os.h, 188  
os\_event\_notify\_get  
    wm\_os.h, 176  
os\_event\_notify\_put  
    wm\_os.h, 177  
os\_get\_runtime\_stats  
    wm\_os.h, 189  
os\_get\_timestamp  
    wm\_os.h, 165  
os\_mem\_alloc  
    wm\_os.h, 186  
os\_mem\_calloc  
    wm\_os.h, 187  
os\_mem\_free  
    wm\_os.h, 187  
os\_msec\_to\_ticks  
    wm\_os.h, 165  
os\_mutex\_create  
    wm\_os.h, 173  
os\_mutex\_delete  
    wm\_os.h, 176  
os\_mutex\_get  
    wm\_os.h, 173  
os\_mutex\_put  
    wm\_os.h, 174  
os\_queue\_create  
    wm\_os.h, 169  
os\_queue\_delete  
    wm\_os.h, 170  
os\_queue\_get\_msgs\_waiting  
    wm\_os.h, 171  
os\_queue\_pool\_define  
    wm\_os.h, 188  
os\_queue\_pool\_t, 9  
size, 9

- os\_queue\_rcv
  - wm\_os.h, 170
- os\_queue\_send
  - wm\_os.h, 169
- os\_recursive\_mutex\_create
  - wm\_os.h, 174
- os\_recursive\_mutex\_get
  - wm\_os.h, 175
- os\_recursive\_mutex\_put
  - wm\_os.h, 175
- os\_remove\_idle\_function
  - wm\_os.h, 172
- os\_remove\_tick\_function
  - wm\_os.h, 172
- os\_rwlock\_create
  - wm\_os.h, 180
- os\_rwlock\_delete
  - wm\_os.h, 180
- os\_rwlock\_read\_lock
  - wm\_os.h, 182
- os\_rwlock\_read\_unlock
  - wm\_os.h, 182
- os\_rwlock\_write\_lock
  - wm\_os.h, 181
- os\_rwlock\_write\_unlock
  - wm\_os.h, 181
- os\_semaphore\_create
  - wm\_os.h, 177
- os\_semaphore\_create\_counting
  - wm\_os.h, 177
- os\_semaphore\_delete
  - wm\_os.h, 179
- os\_semaphore\_get
  - wm\_os.h, 178
- os\_semaphore\_getcount
  - wm\_os.h, 179
- os\_semaphore\_put
  - wm\_os.h, 179
- os\_setup\_idle\_function
  - wm\_os.h, 171
- os\_setup\_tick\_function
  - wm\_os.h, 172
- os\_thread\_create
  - wm\_os.h, 166
- os\_thread\_delete
  - wm\_os.h, 167
- os\_thread\_relinquish
  - wm\_os.h, 188
- os\_thread\_self\_complete
  - wm\_os.h, 168
- os\_thread\_sleep
  - wm\_os.h, 168
- os\_thread\_stack\_define
  - wm\_os.h, 188
- os\_thread\_stack\_t, 10
  - size, 10
- os\_ticks\_get
  - wm\_os.h, 165
- os\_ticks\_to\_msec
  - wm\_os.h, 166
- os\_ticks\_to\_unblock
  - wm\_os.h, 188
- os\_timer\_activate
  - wm\_os.h, 183
- os\_timer\_activate\_t
  - wm\_os.h, 190
- os\_timer\_change
  - wm\_os.h, 184
- os\_timer\_create
  - wm\_os.h, 183
- os\_timer\_deactivate
  - wm\_os.h, 185
- os\_timer\_delete
  - wm\_os.h, 186
- os\_timer\_get\_context
  - wm\_os.h, 185
- os\_timer\_is\_running
  - wm\_os.h, 184
- os\_timer\_reload\_t
  - wm\_os.h, 190
- os\_timer\_reset
  - wm\_os.h, 185
- param
  - wifi\_ds\_rate, 22
- passive\_scan\_or\_radar\_detect
  - wifi\_chan\_info\_t, 14
- password
  - wlan\_network\_security, 58
- password\_len
  - wlan\_network\_security, 58
- pattern
  - wifi\_mef\_filter\_t, 28
- payload
  - wifi\_mgmt\_frame\_t, 31
- phtcap\_ie\_present
  - wifi\_scan\_result, 42
- phtinfo\_ie\_present
  - wifi\_scan\_result, 42
- pktLength
  - wifi\_cw\_mode\_ctrl\_t, 18
- pmk
  - wlan\_network\_security, 59
- pmk\_valid
  - wlan\_network\_security, 59
- power\_mgmt\_status
  - wifi\_sta\_info\_t, 44
- print\_ascii
  - wm\_utils.h, 197
- print\_txpwrlimit
  - wlan\_tests.h, 155
- psk
  - wlan\_network\_security, 58
- psk\_len
  - wlan\_network\_security, 58
- RSSI

- wifi\_scan\_result, 41
- random\_hdlr\_t
  - wm\_utils.h, 197
- random\_initialize\_seed
  - wm\_utils.h, 193
- random\_register\_handler
  - wm\_utils.h, 192
- random\_register\_seed\_handler
  - wm\_utils.h, 192
- random\_unregister\_handler
  - wm\_utils.h, 192
- random\_unregister\_seed\_handler
  - wm\_utils.h, 193
- rate
  - wifi\_rate\_cfg\_t, 32
- rate\_cfg
  - wifi\_ds\_rate, 22
- rate\_format
  - wifi\_rate\_cfg\_t, 32
- rate\_index
  - wifi\_rate\_cfg\_t, 32
- rateInfo
  - wifi\_cw\_mode\_ctrl\_t, 19
- reconnect\_counter
  - wifi\_auto\_reconnect\_config\_t, 11
- reconnect\_interval
  - wifi\_auto\_reconnect\_config\_t, 11
- remain\_period
  - wifi\_remain\_on\_channel\_t, 34
- remove
  - wifi\_remain\_on\_channel\_t, 33
- repeat
  - wifi\_mef\_filter\_t, 28
- reset
  - wifi\_tcp\_keep\_alive\_t, 47
- rf\_type
  - wifi\_rf\_channel\_t, 34
- role
  - wlan\_network, 55
  - wlan\_scan\_result, 61
- rpn
  - wifi\_mef\_entry\_t, 27
- rsn\_mcstCipher
  - wifi\_scan\_result, 42
- rsn\_ucstCipher
  - wifi\_scan\_result, 42
- rsi
  - wifi\_sta\_info\_t, 44
  - wlan\_scan\_result, 62
- rsvd
  - wlan\_cipher, 53
- rx\_data\_rate
  - wifi\_data\_rate\_t, 19
- rx\_ht\_bw
  - wifi\_data\_rate\_t, 20
- rx\_ht\_gi
  - wifi\_data\_rate\_t, 20
- rx\_mcs\_index
  - wifi\_data\_rate\_t, 20
- rx\_rate\_format
  - wifi\_data\_rate\_t, 20
- sample\_initialise\_random\_seed
  - wm\_utils.h, 194
- scan\_time
  - wifi\_scan\_channel\_list\_t, 38
- scan\_type
  - wifi\_scan\_channel\_list\_t, 38
- security
  - wlan\_network, 56
- security\_specific
  - wlan\_network, 56
- seq\_ctl
  - wifi\_mgmt\_frame\_t, 30
- seq\_no
  - wifi\_tcp\_keep\_alive\_t, 48
- size
  - os\_queue\_pool\_t, 9
  - os\_thread\_stack\_t, 10
- soft\_crc32
  - wm\_utils.h, 195
- src\_tcp\_port
  - wifi\_tcp\_keep\_alive\_t, 48
- ssid
  - wifi\_scan\_params\_v2\_t, 39
  - wifi\_scan\_result, 40
  - wlan\_network, 55
  - wlan\_scan\_result, 60
- ssid\_len
  - wifi\_scan\_result, 41
  - wlan\_scan\_result, 60
- ssid\_specific
  - wlan\_network, 56
- start\_freq
  - wifi\_channel\_desc\_t, 17
- status
  - wifi\_remain\_on\_channel\_t, 33
- strdup
  - wm\_utils.h, 194
- sub\_band
  - wifi\_domain\_param\_t, 21
- sub\_command
  - wifi\_ds\_rate, 22
- subband
  - wifi\_txpwrlimit\_t, 52
- timeout
  - wifi\_tcp\_keep\_alive\_t, 48
- tkip
  - wlan\_cipher, 53
- trans\_bssid
  - wifi\_scan\_result, 43
  - wlan\_scan\_result, 62
- trans\_mode
  - wifi\_scan\_result, 43
- trans\_ssid
  - wifi\_scan\_result, 43

- wlan\_scan\_result, 62
- trans\_ssid\_len
  - wifi\_scan\_result, 43
  - wlan\_scan\_result, 62
- tx\_data\_rate
  - wifi\_data\_rate\_t, 19
- tx\_ht\_bw
  - wifi\_data\_rate\_t, 20
- tx\_ht\_gi
  - wifi\_data\_rate\_t, 20
- tx\_mcs\_index
  - wifi\_data\_rate\_t, 20
- tx\_power
  - wifi\_txpwrlimit\_entry\_t, 51
- tx\_rate\_format
  - wifi\_data\_rate\_t, 20
- txPower
  - wifi\_cw\_mode\_ctrl\_t, 18
- txpwrlimit\_config
  - wifi\_txpwrlimit\_t, 52
- txpwrlimit\_entry
  - wifi\_txpwrlimit\_config\_t, 50
- type
  - wifi\_mef\_filter\_t, 28
  - wlan\_network, 55
  - wlan\_network\_security, 57
  - wlan\_scan\_result, 61
- ucstCipher
  - wlan\_network\_security, 58
- verify\_sequential\_pattern
  - wm\_utils.h, 196
- version\_str
  - wifi\_fw\_version\_ext\_t, 25
  - wifi\_fw\_version\_t, 25
- version\_str\_sel
  - wifi\_fw\_version\_ext\_t, 25
- WLAN\_ERROR\_ACTION
  - wlan.h, 143
- WLAN\_ERROR\_NOMEM
  - wlan.h, 143
- WLAN\_ERROR\_NONE
  - wlan.h, 143
- WLAN\_ERROR\_NOT\_SUPPORTED
  - wlan.h, 144
- WLAN\_ERROR\_PARAM
  - wlan.h, 143
- WLAN\_ERROR\_PS\_ACTION
  - wlan.h, 144
- WLAN\_ERROR\_STATE
  - wlan.h, 143
- WLAN\_MAX\_KNOWN\_NETWORKS
  - wlan.h, 143
- WLAN\_NETWORK\_NAME\_MAX\_LENGTH
  - wlan.h, 142
- WLAN\_NETWORK\_NAME\_MIN\_LENGTH
  - wlan.h, 142
- WLAN\_PMK\_LENGTH
  - wlan.h, 143
- WLAN\_PSK\_MIN\_LENGTH
  - wlan.h, 143
- WLAN\_RECONNECT\_LIMIT
  - wlan.h, 142
- WLAN\_RESCAN\_LIMIT
  - wlan.h, 142
- WPA\_WPA2\_WEP
  - wifi\_scan\_result, 41
- wep
  - wlan\_scan\_result, 61
- wep104
  - wlan\_cipher, 53
- wep40
  - wlan\_cipher, 53
- wifi-decl.h, 70
  - BSS\_TYPE\_STA, 70
  - BSS\_TYPE\_UAP, 70
  - MLAN\_MAX\_PASS\_LENGTH, 70
  - MLAN\_MAX\_SSID\_LENGTH, 70
  - MLAN\_MAX\_VER\_STR\_LEN, 70
  - wifi\_SubBand\_t, 70
  - wifi\_frame\_type\_t, 71
- wifi.h, 71
  - \_wifi\_set\_mac\_addr, 79
  - country\_code\_t, 83
  - wifi\_add\_mcast\_filter, 79
  - wifi\_deinit, 72
  - wifi\_deregister\_amsdu\_data\_input\_callback, 74
  - wifi\_deregister\_data\_input\_callback, 73
  - wifi\_enable\_11d\_support, 82
  - wifi\_get\_device\_firmware\_version\_ext, 76
  - wifi\_get\_device\_mac\_addr, 75
  - wifi\_get\_ipv4\_multicast\_mac, 80
  - wifi\_get\_last\_cmd\_sent\_ms, 76
  - wifi\_get\_region\_code, 81
  - wifi\_get\_scan\_result, 77
  - wifi\_get\_scan\_result\_count, 78
  - wifi\_get\_uap\_channel, 82
  - wifi\_init, 71
  - wifi\_init\_fcc, 72
  - wifi\_low\_level\_output, 74
  - wifi\_register\_amsdu\_data\_input\_callback, 73
  - wifi\_register\_data\_input\_callback, 73
  - wifi\_register\_event\_queue, 76
  - wifi\_remove\_mcast\_filter, 80
  - wifi\_set\_cal\_data, 78
  - wifi\_set\_mac\_addr, 79
  - wifi\_set\_packet\_retry\_count, 74
  - wifi\_set\_region\_code, 81
  - wifi\_sta\_ampdu\_rx\_disable, 75
  - wifi\_sta\_ampdu\_rx\_enable, 75
  - wifi\_sta\_ampdu\_tx\_disable, 75
  - wifi\_sta\_ampdu\_tx\_enable, 75
  - wifi\_uap\_bss\_sta\_list, 78
  - wifi\_unregister\_event\_queue, 77
  - wifi\_update\_last\_cmd\_sent\_ms, 76



- wifi\_SubBand\_t
  - wifi-decl.h, 70
- wifi\_add\_mcast\_filter
  - wifi.h, 79
- wifi\_antcfg\_t, 10
  - ant\_mode, 11
  - evaluate\_time, 11
- wifi\_auto\_reconnect\_config\_t, 11
  - flags, 12
  - reconnect\_counter, 11
  - reconnect\_interval, 11
- wifi\_bandcfg\_t, 12
  - config\_bands, 12
  - fw\_bands, 12
- wifi\_cal\_data\_t, 13
  - data, 13
  - data\_len, 13
- wifi\_chan\_info\_t, 13
  - chan\_freq, 14
  - chan\_num, 14
  - passive\_scan\_or\_radar\_detect, 14
- wifi\_chan\_list\_param\_set\_t, 14
  - chan\_scan\_param, 15
  - no\_of\_channels, 14
- wifi\_chan\_scan\_param\_set\_t, 15
  - chan\_number, 15
  - max\_scan\_time, 15
  - min\_scan\_time, 15
- wifi\_chanlist\_t, 16
  - chan\_info, 16
  - num\_chans, 16
- wifi\_channel\_desc\_t, 17
  - chan\_num, 17
  - chan\_width, 17
  - start\_freq, 17
- wifi\_cw\_mode\_ctrl\_t, 18
  - chanInfo, 18
  - channel, 18
  - mode, 18
  - pktLength, 18
  - rateInfo, 19
  - txPower, 18
- wifi\_data\_rate\_t, 19
  - rx\_data\_rate, 19
  - rx\_ht\_bw, 20
  - rx\_ht\_gi, 20
  - rx\_mcs\_index, 20
  - rx\_rate\_format, 20
  - tx\_data\_rate, 19
  - tx\_ht\_bw, 20
  - tx\_ht\_gi, 20
  - tx\_mcs\_index, 20
  - tx\_rate\_format, 20
- wifi\_deinit
  - wifi.h, 72
- wifi\_deregister\_amsdu\_data\_input\_callback
  - wifi.h, 74
- wifi\_deregister\_data\_input\_callback
  - wifi.h, 73
- wifi\_domain\_param\_t, 21
  - country\_code, 21
  - no\_of\_sub\_band, 21
  - sub\_band, 21
- wifi\_ds\_rate, 22
  - data\_rate, 22
  - param, 22
  - rate\_cfg, 22
  - sub\_command, 22
- wifi\_ed\_mac\_ctrl\_t, 23
  - ed\_ctrl\_2g, 23
  - ed\_offset\_2g, 23
- wifi\_enable\_11d\_support
  - wifi.h, 82
- wifi\_event
  - wifi\_events.h, 84
- wifi\_event\_reason
  - wifi\_events.h, 85
- wifi\_events.h, 84
  - wifi\_event, 84
  - wifi\_event\_reason, 85
  - wifi\_wakeup\_event\_t, 86
  - wlan\_bss\_role, 86
  - wlan\_bss\_type, 85
- wififlt\_cfg\_t, 24
  - criteria, 24
  - mef\_entry, 24
  - nentries, 24
- wifi\_frame\_type\_t
  - wifi-decl.h, 71
- wifi\_fw\_version\_ext\_t, 24
  - version\_str, 25
  - version\_str\_sel, 25
- wifi\_fw\_version\_t, 25
  - version\_str, 25
- wifi\_get\_device\_firmware\_version\_ext
  - wifi.h, 76
- wifi\_get\_device\_mac\_addr
  - wifi.h, 75
- wifi\_get\_ipv4\_multicast\_mac
  - wifi.h, 80
- wifi\_get\_last\_cmd\_sent\_ms
  - wifi.h, 76
- wifi\_get\_region\_code
  - wifi.h, 81
- wifi\_get\_scan\_result
  - wifi.h, 77
- wifi\_get\_scan\_result\_count
  - wifi.h, 78
- wifi\_get\_uap\_channel
  - wifi.h, 82
- wifi\_init
  - wifi.h, 71
- wifi\_init\_fcc
  - wifi.h, 72
- wifi\_low\_level\_output
  - wifi.h, 74

- wifi\_mac\_addr\_t, 26
  - mac, 26
- wifi\_mef\_entry\_t, 26
  - action, 27
  - filter\_item, 27
  - filter\_num, 27
  - mode, 26
  - rpn, 27
- wifi\_mef\_filter\_t, 27
  - byte\_seq, 28
  - mask\_seq, 29
  - num\_byte\_seq, 28
  - num\_bytes, 28
  - num\_mask\_seq, 29
  - offset, 28
  - pattern, 28
  - repeat, 28
  - type, 28
- wifi\_mgmt\_frame\_t, 29
  - addr1, 30
  - addr2, 30
  - addr3, 30
  - addr4, 30
  - duration\_id, 30
  - frame\_ctrl\_flags, 30
  - frame\_type, 30
  - frm\_len, 29
  - payload, 31
  - seq\_ctl, 30
- wifi\_nat\_keep\_alive\_t, 31
  - dst\_ip, 31
  - dst\_mac, 31
  - dst\_port, 32
  - interval, 31
- wifi\_rate\_cfg\_t, 32
  - rate, 32
  - rate\_format, 32
  - rate\_index, 32
- wifi\_register\_amsdu\_data\_input\_callback
  - wifi.h, 73
- wifi\_register\_data\_input\_callback
  - wifi.h, 73
- wifi\_register\_event\_queue
  - wifi.h, 76
- wifi\_remain\_on\_channel\_t, 33
  - bandcfg, 33
  - channel, 33
  - remain\_period, 34
  - remove, 33
  - status, 33
- wifi\_remove\_mcast\_filter
  - wifi.h, 80
- wifi\_rf\_channel\_t, 34
  - current\_channel, 34
  - rf\_type, 34
- wifi\_rssi\_info\_t, 35
  - bcn\_nf\_avg, 36
  - bcn\_nf\_last, 36
  - bcn\_rssi\_avg, 36
  - bcn\_rssi\_last, 36
  - bcn\_snr\_avg, 36
  - bcn\_snr\_last, 35
  - data\_nf\_avg, 35
  - data\_nf\_last, 35
  - data\_rssi\_avg, 35
  - data\_rssi\_last, 35
  - data\_snr\_avg, 36
  - data\_snr\_last, 36
- wifi\_scan\_chan\_list\_t, 37
  - chan\_number, 37
  - num\_of\_chan, 37
- wifi\_scan\_channel\_list\_t, 38
  - chan\_number, 38
  - scan\_time, 38
  - scan\_type, 38
- wifi\_scan\_params\_v2\_t, 38
  - bssid, 39
  - cb, 39
  - chan\_list, 39
  - num\_channels, 39
  - num\_probes, 39
  - ssid, 39
- wifi\_scan\_result, 40
  - band, 42
  - beacon\_period, 41
  - bssid, 40
  - Channel, 41
  - dtim\_period, 41
  - is\_ibss\_bit\_set, 40
  - is\_pmf\_required, 42
  - phtcap\_ie\_present, 42
  - phtinfo\_ie\_present, 42
  - RSSI, 41
  - rsn\_mcstCipher, 42
  - rsn\_ucstCipher, 42
  - ssid, 40
  - ssid\_len, 41
  - trans\_bssid, 43
  - trans\_mode, 43
  - trans\_ssid, 43
  - trans\_ssid\_len, 43
  - WPA\_WPA2\_WEP, 41
  - wmm\_ie\_present, 42
  - wpa2\_entp\_IE\_exist, 43
  - wpa\_mcstCipher, 41
  - wpa\_ucstCipher, 41
  - wps\_IE\_exist, 43
  - wps\_session, 43
- wifi\_set\_cal\_data
  - wifi.h, 78
- wifi\_set\_mac\_addr
  - wifi.h, 79
- wifi\_set\_packet\_retry\_count
  - wifi.h, 74
- wifi\_set\_region\_code
  - wifi.h, 81

- wifi\_sta\_ampdu\_rx\_disable
  - wifi.h, 75
- wifi\_sta\_ampdu\_rx\_enable
  - wifi.h, 75
- wifi\_sta\_ampdu\_tx\_disable
  - wifi.h, 75
- wifi\_sta\_ampdu\_tx\_enable
  - wifi.h, 75
- wifi\_sta\_info\_t, 44
  - mac, 44
  - power\_mgmt\_status, 44
  - rsi, 44
- wifi\_sta\_list\_t, 45
  - count, 45
- wifi\_sub\_band\_set\_t, 45
  - first\_chan, 45
  - max\_tx\_pwr, 46
  - no\_of\_chan, 46
- wifi\_tbt\_offset\_t, 46
  - avg\_tbt\_offset, 47
  - max\_tbt\_offset, 46
  - min\_tbt\_offset, 46
- wifi\_tcp\_keep\_alive\_t, 47
  - dst\_ip, 48
  - dst\_mac, 48
  - dst\_tcp\_port, 48
  - enable, 47
  - interval, 48
  - max\_keep\_alives, 48
  - reset, 47
  - seq\_no, 48
  - src\_tcp\_port, 48
  - timeout, 48
- wifi\_tx\_power\_t, 49
  - current\_level, 49
  - max\_power, 49
  - min\_power, 49
- wifi\_txpwrlimit\_config\_t, 50
  - chan\_desc, 50
  - num\_mod\_grps, 50
  - txpwrlimit\_entry, 50
- wifi\_txpwrlimit\_entry\_t, 50
  - mod\_group, 51
  - tx\_power, 51
- wifi\_txpwrlimit\_t, 51
  - num\_chans, 52
  - subband, 52
  - txpwrlimit\_config, 52
- wifi\_uap\_bss\_sta\_list
  - wifi.h, 78
- wifi\_unregister\_event\_queue
  - wifi.h, 77
- wifi\_update\_last\_cmd\_sent\_ms
  - wifi.h, 76
- wifi\_wakeup\_event\_t
  - wifi\_events.h, 86
- wlan.h, 86
  - ACTION\_GET, 142
  - ACTION\_SET, 142
  - address\_types, 149
  - IEEEtypes\_ADDRESS\_SIZE, 142
  - IEEEtypes\_SSID\_SIZE, 142
  - is\_sta\_connected, 96
  - is\_sta\_ipv4\_connected, 96
  - is\_uap\_started, 96
  - WLAN\_ERROR\_ACTION, 143
  - WLAN\_ERROR\_NOMEM, 143
  - WLAN\_ERROR\_NONE, 143
  - WLAN\_ERROR\_NOT\_SUPPORTED, 144
  - WLAN\_ERROR\_PARAM, 143
  - WLAN\_ERROR\_PS\_ACTION, 144
  - WLAN\_ERROR\_STATE, 143
  - WLAN\_MAX\_KNOWN\_NETWORKS, 143
  - WLAN\_NETWORK\_NAME\_MAX\_LENGTH, 142
  - WLAN\_NETWORK\_NAME\_MIN\_LENGTH, 142
  - WLAN\_PMK\_LENGTH, 143
  - WLAN\_PSK\_MIN\_LENGTH, 143
  - WLAN\_RECONNECT\_LIMIT, 142
  - WLAN\_RESCAN\_LIMIT, 142
  - wlan\_add\_network, 89
  - wlan\_bandcfg\_t, 145
  - wlan\_basic\_cli\_init, 115
  - wlan\_cal\_data\_t, 144
  - wlan\_chanlist\_t, 145
  - wlan\_clear\_mgmt\_ie, 124
  - wlan\_cli\_init, 115
  - wlan\_configure\_listen\_interval, 103
  - wlan\_configure\_null\_pkt\_interval, 104
  - wlan\_connect, 90
  - wlan\_connection\_state, 147
  - wlan\_cw\_mode\_ctrl\_t, 145
  - wlan\_deepsleeps\_off, 107
  - wlan\_deepsleeps\_on, 107
  - wlan\_deinit, 88
  - wlan\_disconnect, 91
  - wlan\_ds\_rate, 145
  - wlan\_ed\_mac\_ctrl\_t, 145
  - wlan\_enhanced\_cli\_init, 116
  - wlan\_event\_reason, 146
  - wlanflt\_cfg\_t, 144
  - wlan\_get\_11d\_enable\_status, 124
  - wlan\_get\_address, 93
  - wlan\_get\_antcfg, 105
  - wlan\_get\_average\_signal\_strength, 125
  - wlan\_get\_beacon\_period, 108
  - wlan\_get\_cal\_data, 126
  - wlan\_get\_chanlist, 128
  - wlan\_get\_connection\_state, 98
  - wlan\_get\_current\_bssid, 113
  - wlan\_get\_current\_channel, 114
  - wlan\_get\_current\_network, 95
  - wlan\_get\_current\_signal\_strength, 125
  - wlan\_get\_current\_uap\_network, 95
  - wlan\_get\_data\_rate, 108
  - wlan\_get\_dtim\_period, 108
  - wlan\_get\_ed\_mac\_mode, 102

- wlan\_get\_firmware\_version\_ext, 105
- wlan\_get\_mac\_address, 93
- wlan\_get\_mgmt\_ie, 123
- wlan\_get\_network, 96
- wlan\_get\_network\_byname, 97
- wlan\_get\_network\_count, 98
- wlan\_get\_otp\_user\_data, 126
- wlan\_get\_pmfcfg, 109
- wlan\_get\_ps\_mode, 114
- wlan\_get\_scan\_result, 100
- wlan\_get\_sta\_tx\_power, 122
- wlan\_get\_tsf, 106
- wlan\_get\_txpwrlimit, 128
- wlan\_get\_txratecfg, 122
- wlan\_get\_uap\_address, 94
- wlan\_get\_uap\_channel, 94
- wlan\_get\_uap\_connection\_state, 99
- wlan\_get\_uap\_max\_clients, 117
- wlan\_get\_uap\_supported\_max\_clients, 116
- wlan\_ieee80211\_off, 107
- wlan\_ieee80211\_on, 106
- wlan\_init, 87
- wlan\_initialize\_uap\_network, 89
- wlan\_ps\_mode, 148
- wlan\_remain\_on\_channel, 125
- wlan\_remove\_network, 90
- wlan\_scan, 99
- wlan\_scan\_channel\_list\_t, 144
- wlan\_scan\_params\_v2\_t, 144
- wlan\_scan\_with\_opt, 100
- wlan\_security\_type, 148
- wlan\_send\_host\_sleep, 113
- wlan\_send\_hostcmd, 141
- wlan\_set\_antcfg, 104
- wlan\_set\_auto\_arp, 113
- wlan\_set\_cal\_data, 102
- wlan\_set\_chanlist, 127
- wlan\_set\_chanlist\_and\_txpwrlimit, 127
- wlan\_set\_crypto\_AES\_CCMP\_decrypt, 138
- wlan\_set\_crypto\_AES\_CCMP\_encrypt, 138
- wlan\_set\_crypto\_AES\_ECB\_decrypt, 136
- wlan\_set\_crypto\_AES\_ECB\_encrypt, 135
- wlan\_set\_crypto\_AES\_GCMP\_decrypt, 140
- wlan\_set\_crypto\_AES\_GCMP\_encrypt, 139
- wlan\_set\_crypto\_AES\_WRAP\_decrypt, 137
- wlan\_set\_crypto\_AES\_WRAP\_encrypt, 136
- wlan\_set\_crypto\_RC4\_decrypt, 134
- wlan\_set\_crypto\_RC4\_encrypt, 134
- wlan\_set\_ed\_mac\_mode, 101
- wlan\_set\_htcapinfo, 118
- wlan\_set\_httxcf, 118
- wlan\_set\_mac\_addr, 102
- wlan\_set\_mgmt\_ie, 123
- wlan\_set\_packet\_filters, 110
- wlan\_set\_pmfcfg, 109
- wlan\_set\_reassoc\_control, 129
- wlan\_set\_sta\_tx\_power, 122
- wlan\_set\_txpwrlimit, 128
- wlan\_set\_txratecfg, 120
- wlan\_set\_uap\_max\_clients, 117
- wlan\_set\_wwsm\_txpwrlimit, 123
- wlan\_sta\_ampdu\_rx\_disable, 133
- wlan\_sta\_ampdu\_rx\_enable, 133
- wlan\_sta\_ampdu\_tx\_disable, 133
- wlan\_sta\_ampdu\_tx\_enable, 132
- wlan\_start, 87
- wlan\_start\_network, 91
- wlan\_stop, 88
- wlan\_stop\_network, 92
- wlan\_tcp\_keep\_alive\_t, 145
- wlan\_txpwrlimit\_t, 145
- wlan\_uap\_ctrl\_deauth, 131
- wlan\_uap\_set\_bandwidth, 130
- wlan\_uap\_set\_beacon\_period, 129
- wlan\_uap\_set\_ecsa, 131
- wlan\_uap\_set\_hidden\_ssid, 130
- wlan\_uap\_set\_htcapinfo, 131
- wlan\_uap\_set\_httxcf, 132
- wlan\_uap\_set\_scan\_chan\_list, 133
- wlan\_version\_extended, 105
- wlan\_wakeup\_event\_t, 147
- wlan\_wfa\_basic\_cli\_init, 115
- wlan\_wlcmgr\_send\_msg, 114
- wlan\_wowlan\_ptn\_cfg\_t, 144
- wm\_wlan\_errno, 145
- wlan\_11d.h, 149
- wlan\_11d\_country\_index\_2\_string, 155
- wlan\_enable\_11d, 149
- wlan\_get\_country, 150
- wlan\_set\_country, 151
- wlan\_set\_domain\_params, 151
- wlan\_set\_region\_code, 154
- wlan\_uap\_set\_country, 150
- wlan\_11d\_country\_index\_2\_string
- wlan\_11d.h, 155
- wlan\_add\_network
- wlan.h, 89
- wlan\_bandcfg\_t
- wlan.h, 145
- wlan\_basic\_cli\_init
- wlan.h, 115
- wlan\_bss\_role
- wifi\_events.h, 86
- wlan\_bss\_type
- wifi\_events.h, 85
- wlan\_cal\_data\_t
- wlan.h, 144
- wlan\_chanlist\_t
- wlan.h, 145
- wlan\_cipher, 52
- ccmp, 53
- rsvd, 53
- tkip, 53
- wep104, 53
- wep40, 53
- wlan\_clear\_mgmt\_ie

wlan.h, 124  
wlan\_cli\_init  
    wlan.h, 115  
wlan\_configure\_listen\_interval  
    wlan.h, 103  
wlan\_configure\_null\_pkt\_interval  
    wlan.h, 104  
wlan\_connect  
    wlan.h, 90  
wlan\_connection\_state  
    wlan.h, 147  
wlan\_cw\_mode\_ctrl\_t  
    wlan.h, 145  
wlan\_deepsleeps\_off  
    wlan.h, 107  
wlan\_deepsleeps\_on  
    wlan.h, 107  
wlan\_deinit  
    wlan.h, 88  
wlan\_disconnect  
    wlan.h, 91  
wlan\_ds\_rate  
    wlan.h, 145  
wlan\_ed\_mac\_ctrl\_t  
    wlan.h, 145  
wlan\_enable\_11d  
    wlan\_11d.h, 149  
wlan\_enhanced\_cli\_init  
    wlan.h, 116  
wlan\_event\_reason  
    wlan.h, 146  
wlanflt\_cfg\_t  
    wlan.h, 144  
wlan\_get\_11d\_enable\_status  
    wlan.h, 124  
wlan\_get\_address  
    wlan.h, 93  
wlan\_get\_antcfg  
    wlan.h, 105  
wlan\_get\_average\_signal\_strength  
    wlan.h, 125  
wlan\_get\_beacon\_period  
    wlan.h, 108  
wlan\_get\_cal\_data  
    wlan.h, 126  
wlan\_get\_chanlist  
    wlan.h, 128  
wlan\_get\_connection\_state  
    wlan.h, 98  
wlan\_get\_country  
    wlan\_11d.h, 150  
wlan\_get\_current\_bssid  
    wlan.h, 113  
wlan\_get\_current\_channel  
    wlan.h, 114  
wlan\_get\_current\_network  
    wlan.h, 95  
wlan\_get\_current\_signal\_strength  
    wlan.h, 125  
wlan\_get\_current\_uap\_network  
    wlan.h, 95  
wlan\_get\_data\_rate  
    wlan.h, 108  
wlan\_get\_dtim\_period  
    wlan.h, 108  
wlan\_get\_ed\_mac\_mode  
    wlan.h, 102  
wlan\_get\_firmware\_version\_ext  
    wlan.h, 105  
wlan\_get\_mac\_address  
    wlan.h, 93  
wlan\_get\_mgmt\_ie  
    wlan.h, 123  
wlan\_get\_network  
    wlan.h, 96  
wlan\_get\_network\_byname  
    wlan.h, 97  
wlan\_get\_network\_count  
    wlan.h, 98  
wlan\_get\_otp\_user\_data  
    wlan.h, 126  
wlan\_get\_pmfcfg  
    wlan.h, 109  
wlan\_get\_ps\_mode  
    wlan.h, 114  
wlan\_get\_scan\_result  
    wlan.h, 100  
wlan\_get\_sta\_tx\_power  
    wlan.h, 122  
wlan\_get\_tsf  
    wlan.h, 106  
wlan\_get\_txpwrlimit  
    wlan.h, 128  
wlan\_get\_txratecfg  
    wlan.h, 122  
wlan\_get\_uap\_address  
    wlan.h, 94  
wlan\_get\_uap\_channel  
    wlan.h, 94  
wlan\_get\_uap\_connection\_state  
    wlan.h, 99  
wlan\_get\_uap\_max\_clients  
    wlan.h, 117  
wlan\_get\_uap\_supported\_max\_clients  
    wlan.h, 116  
wlan\_ieeeeps\_off  
    wlan.h, 107  
wlan\_ieeeeps\_on  
    wlan.h, 106  
wlan\_init  
    wlan.h, 87  
wlan\_initialize\_uap\_network  
    wlan.h, 89  
wlan\_ip\_config, 53  
    ipv4, 54  
wlan\_network, 54

- beacon\_period, 57
- bssid, 55
- bssid\_specific, 56
- channel, 55
- channel\_specific, 56
- dtim\_period, 57
- ip, 56
- name, 55
- role, 55
- security, 56
- security\_specific, 56
- ssid, 55
- ssid\_specific, 56
- type, 55
- wlan\_network\_security, 57
  - is\_pmf\_required, 58
  - mcstCipher, 58
  - mfpc, 59
  - mfpr, 59
  - password, 58
  - password\_len, 58
  - pmk, 59
  - pmk\_valid, 59
  - psk, 58
  - psk\_len, 58
  - type, 57
  - ucstCipher, 58
- wlan\_ps\_mode
  - wlan.h, 148
- wlan\_remain\_on\_channel
  - wlan.h, 125
- wlan\_remove\_network
  - wlan.h, 90
- wlan\_scan
  - wlan.h, 99
- wlan\_scan\_channel\_list\_t
  - wlan.h, 144
- wlan\_scan\_params\_v2\_t
  - wlan.h, 144
- wlan\_scan\_result, 60
  - beacon\_period, 62
  - bssid, 60
  - channel, 60
  - dtim\_period, 62
  - role, 61
  - rsi, 62
  - ssid, 60
  - ssid\_len, 60
  - trans\_bssid, 62
  - trans\_ssid, 62
  - trans\_ssid\_len, 62
  - type, 61
  - wep, 61
  - wmm, 61
  - wpa, 61
  - wpa2, 61
  - wpa2\_entp, 61
  - wpa3\_sae, 62
- wlan\_scan\_with\_opt
  - wlan.h, 100
- wlan\_security\_type
  - wlan.h, 148
- wlan\_send\_host\_sleep
  - wlan.h, 113
- wlan\_send\_hostcmd
  - wlan.h, 141
- wlan\_set\_antcfg
  - wlan.h, 104
- wlan\_set\_auto\_arp
  - wlan.h, 113
- wlan\_set\_cal\_data
  - wlan.h, 102
- wlan\_set\_chanlist
  - wlan.h, 127
- wlan\_set\_chanlist\_and\_txpwrlimit
  - wlan.h, 127
- wlan\_set\_country
  - wlan\_11d.h, 151
- wlan\_set\_crypto\_AES\_CCMP\_decrypt
  - wlan.h, 138
- wlan\_set\_crypto\_AES\_CCMP\_encrypt
  - wlan.h, 138
- wlan\_set\_crypto\_AES\_ECB\_decrypt
  - wlan.h, 136
- wlan\_set\_crypto\_AES\_ECB\_encrypt
  - wlan.h, 135
- wlan\_set\_crypto\_AES\_GCMP\_decrypt
  - wlan.h, 140
- wlan\_set\_crypto\_AES\_GCMP\_encrypt
  - wlan.h, 139
- wlan\_set\_crypto\_AES\_WRAP\_decrypt
  - wlan.h, 137
- wlan\_set\_crypto\_AES\_WRAP\_encrypt
  - wlan.h, 136
- wlan\_set\_crypto\_RC4\_decrypt
  - wlan.h, 134
- wlan\_set\_crypto\_RC4\_encrypt
  - wlan.h, 134
- wlan\_set\_domain\_params
  - wlan\_11d.h, 151
- wlan\_set\_ed\_mac\_mode
  - wlan.h, 101
- wlan\_set\_htcapinfo
  - wlan.h, 118
- wlan\_set\_httxcfg
  - wlan.h, 118
- wlan\_set\_mac\_addr
  - wlan.h, 102
- wlan\_set\_mgmt\_ie
  - wlan.h, 123
- wlan\_set\_packet\_filters
  - wlan.h, 110
- wlan\_set\_pmfcfg
  - wlan.h, 109
- wlan\_set\_reassoc\_control
  - wlan.h, 129

- wlan\_set\_region\_code
  - wlan\_11d.h, 154
- wlan\_set\_sta\_tx\_power
  - wlan.h, 122
- wlan\_set\_txpwrlimit
  - wlan.h, 128
- wlan\_set\_txratecfg
  - wlan.h, 120
- wlan\_set\_uap\_max\_clients
  - wlan.h, 117
- wlan\_set\_wwsm\_txpwrlimit
  - wlan.h, 123
- wlan\_sta\_ampdu\_rx\_disable
  - wlan.h, 133
- wlan\_sta\_ampdu\_rx\_enable
  - wlan.h, 133
- wlan\_sta\_ampdu\_tx\_disable
  - wlan.h, 133
- wlan\_sta\_ampdu\_tx\_enable
  - wlan.h, 132
- wlan\_start
  - wlan.h, 87
- wlan\_start\_network
  - wlan.h, 91
- wlan\_stop
  - wlan.h, 88
- wlan\_stop\_network
  - wlan.h, 92
- wlan\_tcp\_keep\_alive\_t
  - wlan.h, 145
- wlan\_tests.h, 155
  - print\_txpwrlimit, 155
- wlan\_txpwrlimit\_t
  - wlan.h, 145
- wlan\_uap\_ctrl\_deauth
  - wlan.h, 131
- wlan\_uap\_set\_bandwidth
  - wlan.h, 130
- wlan\_uap\_set\_beacon\_period
  - wlan.h, 129
- wlan\_uap\_set\_country
  - wlan\_11d.h, 150
- wlan\_uap\_set\_ecsa
  - wlan.h, 131
- wlan\_uap\_set\_hidden\_ssid
  - wlan.h, 130
- wlan\_uap\_set\_htcapinfo
  - wlan.h, 131
- wlan\_uap\_set\_httxcfgr
  - wlan.h, 132
- wlan\_uap\_set\_scan\_chan\_list
  - wlan.h, 133
- wlan\_version\_extended
  - wlan.h, 105
- wlan\_wakeup\_event\_t
  - wlan.h, 147
- wlan\_wfa\_basic\_cli\_init
  - wlan.h, 115
- wlan\_wlcmgr\_send\_msg
  - wlan.h, 114
- wlan\_wowlan\_ptn\_cfg\_t
  - wlan.h, 144
- wm\_dhcpd\_errno
  - dhcp-server.h, 68
- wm\_net.h, 156
  - net\_configure\_address, 161
  - net\_configure\_dns, 162
  - net\_dhcp\_hostname\_set, 156
  - net\_get\_if\_addr, 162
  - net\_get\_if\_ip\_addr, 163
  - net\_get\_if\_ip\_mask, 163
  - net\_get\_if\_name, 162
  - net\_get\_sock\_error, 157
  - net\_get\_sta\_handle, 159
  - net\_get\_uap\_handle, 160
  - net\_gethostbyname, 157
  - net\_inet\_aton, 157
  - net\_inet\_ntoa, 158
  - net\_interface\_dhcp\_stop, 161
  - net\_interface\_down, 160
  - net\_interface\_up, 160
  - net\_ipv4stack\_init, 164
  - net\_is\_ip\_or\_ipv6, 158
  - net\_sock\_to\_interface, 159
  - net\_socket\_blocking, 156
  - net\_stat, 164
  - net\_stop\_dhcp\_timer, 156
  - net\_wlan\_deinit, 159
  - net\_wlan\_init, 159
- wm\_os.h, 164
  - cb\_fn, 189
  - OS\_MUTEX\_INHERIT, 189
  - OS\_MUTEX\_NO\_INHERIT, 189
  - OS\_NO\_WAIT, 189
  - OS\_WAIT\_FOREVER, 189
  - os\_disable\_all\_interrupts, 187
  - os\_enable\_all\_interrupts, 188
  - os\_event\_notify\_get, 176
  - os\_event\_notify\_put, 177
  - os\_get\_runtime\_stats, 189
  - os\_get\_timestamp, 165
  - os\_mem\_alloc, 186
  - os\_mem\_calloc, 187
  - os\_mem\_free, 187
  - os\_msec\_to\_ticks, 165
  - os\_mutex\_create, 173
  - os\_mutex\_delete, 176
  - os\_mutex\_get, 173
  - os\_mutex\_put, 174
  - os\_queue\_create, 169
  - os\_queue\_delete, 170
  - os\_queue\_get\_msgs\_waiting, 171
  - os\_queue\_pool\_define, 188
  - os\_queue\_recv, 170
  - os\_queue\_send, 169
  - os\_recursive\_mutex\_create, 174

- os\_recursive\_mutex\_get, 175
- os\_recursive\_mutex\_put, 175
- os\_remove\_idle\_function, 172
- os\_remove\_tick\_function, 172
- os\_rwlock\_create, 180
- os\_rwlock\_delete, 180
- os\_rwlock\_read\_lock, 182
- os\_rwlock\_read\_unlock, 182
- os\_rwlock\_write\_lock, 181
- os\_rwlock\_write\_unlock, 181
- os\_semaphore\_create, 177
- os\_semaphore\_create\_counting, 177
- os\_semaphore\_delete, 179
- os\_semaphore\_get, 178
- os\_semaphore\_getcount, 179
- os\_semaphore\_put, 179
- os\_setup\_idle\_function, 171
- os\_setup\_tick\_function, 172
- os\_thread\_create, 166
- os\_thread\_delete, 167
- os\_thread\_relinquish, 188
- os\_thread\_self\_complete, 168
- os\_thread\_sleep, 168
- os\_thread\_stack\_define, 188
- os\_ticks\_get, 165
- os\_ticks\_to\_msec, 166
- os\_ticks\_to\_unblock, 188
- os\_timer\_activate, 183
- os\_timer\_activate\_t, 190
- os\_timer\_change, 184
- os\_timer\_create, 183
- os\_timer\_deactivate, 185
- os\_timer\_delete, 186
- os\_timer\_get\_context, 185
- os\_timer\_is\_running, 184
- os\_timer\_reload\_t, 190
- os\_timer\_reset, 185
- wm\_utils.h, 190
  - bin2hex, 191
  - dump\_ascii, 197
  - dump\_hex, 196
  - dump\_hex\_ascii, 196
  - dump\_json, 197
  - fill\_sequential\_pattern, 195
  - get\_random\_sequence, 194
  - hex2bin, 191
  - print\_ascii, 197
  - random\_hdlr\_t, 197
  - random\_initialize\_seed, 193
  - random\_register\_handler, 192
  - random\_register\_seed\_handler, 192
  - random\_unregister\_handler, 192
  - random\_unregister\_seed\_handler, 193
  - sample\_initialise\_random\_seed, 194
  - soft\_crc32, 195
  - strdup, 194
  - verify\_sequential\_pattern, 196
- wm\_wlan\_errno
  - wlan.h, 145
  - wmm
    - wlan\_scan\_result, 61
  - wmm\_ie\_present
    - wifi\_scan\_result, 42
  - wpa
    - wlan\_scan\_result, 61
  - wpa2
    - wlan\_scan\_result, 61
  - wpa2\_entp
    - wlan\_scan\_result, 61
  - wpa2\_entp\_IE\_exist
    - wifi\_scan\_result, 43
  - wpa3\_sae
    - wlan\_scan\_result, 62
  - wpa\_mcstCipher
    - wifi\_scan\_result, 41
  - wpa\_ucstCipher
    - wifi\_scan\_result, 41
  - wps\_IE\_exist
    - wifi\_scan\_result, 43
  - wps\_session
    - wifi\_scan\_result, 43