



WIFI Reference Manual

C API Reference

© NXP Semiconductors, 2008-2020

Generated by Doxygen 1.8.18

1 Main Page	1
1.1 Introduction	1
1.1.1 Developer Documentation	1
2 Data Structure Index	3
2.1 Data Structures	3
3 File Index	7
3.1 File List	7
4 Data Structure Documentation	9
4.1 _Cipher_t Struct Reference	9
4.1.1 Field Documentation	9
4.1.1.1 none	9
4.1.1.2 wep40	10
4.1.1.3 wep104	10
4.1.1.4 tkip	10
4.1.1.5 ccmp	10
4.1.1.6 aes_128_cmac	10
4.1.1.7 gcmp	10
4.1.1.8 sms4	10
4.1.1.9 gcmp_256	10
4.1.1.10 ccmp_256	11
4.1.1.11 rsvd	11
4.1.1.12 bip_gmac_128	11
4.1.1.13 bip_gmac_256	11
4.1.1.14 bip_cmac_256	11
4.1.1.15 gtk_not_used	11
4.1.1.16 rsvd2	11
4.2 _SecurityMode_t Struct Reference	12
4.2.1 Field Documentation	12
4.2.1.1 noRsn	12
4.2.1.2 wepStatic	12
4.2.1.3 wepDynamic	12
4.2.1.4 wpa	13
4.2.1.5 wpaNone	13
4.2.1.6 wpa2	13
4.2.1.7 wpa2_sha256	13
4.2.1.8 owe	13
4.2.1.9 wpa3_sae	13
4.2.1.10 wpa2_entp	13
4.2.1.11 wpa2_entp_sha256	13
4.2.1.12 ft_1x	14

4.2.1.13 ft_1x_sha384	14
4.2.1.14 ft_psk	14
4.2.1.15 ft_sae	14
4.2.1.16 wpa3_1x_sha256	14
4.2.1.17 wpa3_1x_sha384	14
4.2.1.18 rsvd	14
4.3 BandConfig_t Struct Reference	15
4.3.1 Field Documentation	15
4.3.1.1 chanBand	15
4.3.1.2 chanWidth	15
4.3.1.3 chan2Offset	15
4.3.1.4 scanMode	15
4.4 ChanBandInfo_t Struct Reference	15
4.4.1 Field Documentation	16
4.4.1.1 bandConfig	16
4.4.1.2 chanNum	16
4.5 cli_command Struct Reference	16
4.5.1 Detailed Description	16
4.5.2 Field Documentation	16
4.5.2.1 name	16
4.5.2.2 help	16
4.5.2.3 function	17
4.6 csi_local_buff_statu Struct Reference	17
4.6.1 Member Function Documentation	17
4.6.1.1 OSA_SEMAPHORE_HANDLE_DEFINE()	17
4.6.2 Field Documentation	17
4.6.2.1 write_index	17
4.6.2.2 read_index	18
4.6.2.3 valid_data_cnt	18
4.7 Event_Radar_Detected_Info Struct Reference	18
4.7.1 Field Documentation	18
4.7.1.1 detect_count	18
4.7.1.2 reg_domain	18
4.7.1.3 main_det_type	19
4.7.1.4 pw_chirp_type	19
4.7.1.5 pw_chirp_idx	19
4.7.1.6 pw_value	19
4.7.1.7 pri_radar_type	19
4.7.1.8 pri_binCnt	19
4.7.1.9 binCounter	19
4.7.1.10 numDfsRecords	19
4.7.1.11 dfsRecordHdrs	20

4.7.1.12 reallyPassed	20
4.8 ipv4_config Struct Reference	20
4.8.1 Detailed Description	20
4.8.2 Field Documentation	20
4.8.2.1 addr_type	20
4.8.2.2 address	20
4.8.2.3 gw	21
4.8.2.4 netmask	21
4.8.2.5 dns1	21
4.8.2.6 dns2	21
4.9 ipv6_config Struct Reference	21
4.9.1 Detailed Description	21
4.9.2 Field Documentation	21
4.9.2.1 address	22
4.9.2.2 addr_type	22
4.9.2.3 addr_state	22
4.10 net_ip_config Struct Reference	22
4.10.1 Detailed Description	22
4.10.2 Field Documentation	22
4.10.2.1 ipv6	22
4.10.2.2 ipv6_count	23
4.10.2.3 ipv4	23
4.11 net_ipv4_config Struct Reference	23
4.11.1 Detailed Description	23
4.11.2 Field Documentation	23
4.11.2.1 addr_type	23
4.11.2.2 address	24
4.11.2.3 gw	24
4.11.2.4 netmask	24
4.11.2.5 dns1	24
4.11.2.6 dns2	24
4.12 net_ipv6_config Struct Reference	24
4.12.1 Detailed Description	24
4.12.2 Field Documentation	25
4.12.2.1 address	25
4.12.2.2 addr_type	25
4.12.2.3 addr_state	25
4.13 osa_rw_lock_t Struct Reference	25
4.13.1 Member Function Documentation	25
4.13.1.1 OSA_MUTEX_HANDLE_DEFINE() [1/2]	26
4.13.1.2 OSA_MUTEX_HANDLE_DEFINE() [2/2]	26
4.13.1.3 OSA_SEMAPHORE_HANDLE_DEFINE()	26

4.13.2 Field Documentation	26
4.13.2.1 reader_cb	26
4.13.2.2 reader_count	26
4.14 test_cfg_param_t Struct Reference	26
4.14.1 Field Documentation	27
4.14.1.1 name	27
4.14.1.2 offset	27
4.14.1.3 len	27
4.14.1.4 notes	27
4.15 test_cfg_table_t Struct Reference	27
4.15.1 Field Documentation	27
4.15.1.1 name	28
4.15.1.2 data	28
4.15.1.3 len	28
4.15.1.4 param_list	28
4.15.1.5 param_num	28
4.16 tx_ampdu_prot_mode_para Struct Reference	28
4.16.1 Detailed Description	28
4.16.2 Field Documentation	29
4.16.2.1 mode	29
4.17 txrate_setting Struct Reference	29
4.17.1 Detailed Description	29
4.17.2 Field Documentation	29
4.17.2.1 preamble	29
4.17.2.2 bandwidth	29
4.17.2.3 shortGI	30
4.17.2.4 stbc	30
4.17.2.5 dcm	30
4.17.2.6 adv_coding	30
4.17.2.7 doppler	30
4.17.2.8 max_pkttext	30
4.17.2.9 reserved	30
4.18 wifi_11ax_config_t Struct Reference	31
4.18.1 Detailed Description	31
4.18.2 Field Documentation	31
4.18.2.1 band	31
4.18.2.2 id	31
4.18.2.3 len	31
4.18.2.4 ext_id	31
4.18.2.5 he_mac_cap	32
4.18.2.6 he_phy_cap	32
4.18.2.7 he_txrx_mcs_support	32

4.18.2.8 val	32
4.19 wifi_antcfg_t Struct Reference	32
4.19.1 Detailed Description	32
4.19.2 Field Documentation	32
4.19.2.1 ant_mode	33
4.19.2.2 evaluate_time	33
4.19.2.3 current_antenna	33
4.20 wifi_auto_reconnect_config_t Struct Reference	33
4.20.1 Detailed Description	33
4.20.2 Field Documentation	33
4.20.2.1 reconnect_counter	33
4.20.2.2 reconnect_interval	34
4.20.2.3 flags	34
4.21 wifi_bandcfg_t Struct Reference	34
4.21.1 Detailed Description	34
4.21.2 Field Documentation	34
4.21.2.1 config_bands	34
4.21.2.2 fw_bands	34
4.22 wifi_btwt_config_t Struct Reference	35
4.22.1 Detailed Description	35
4.22.2 Field Documentation	35
4.22.2.1 action	35
4.22.2.2 sub_id	35
4.22.2.3 nominal_wake	35
4.22.2.4 max_sta_support	35
4.22.2.5 twt_mantissa	36
4.22.2.6 twt_offset	36
4.22.2.7 twt_exponent	36
4.22.2.8 sp_gap	36
4.23 wifi_cal_data_t Struct Reference	36
4.23.1 Detailed Description	36
4.23.2 Field Documentation	36
4.23.2.1 data_len	37
4.23.2.2 data	37
4.24 wifi_chan_info_t Struct Reference	37
4.24.1 Detailed Description	37
4.24.2 Field Documentation	37
4.24.2.1 chan_num	37
4.24.2.2 chan_freq	37
4.24.2.3 passive_scan_or_radar_detect	38
4.25 wifi_chan_list_param_set_t Struct Reference	38
4.25.1 Detailed Description	38

4.25.2 Field Documentation	38
4.25.2.1 no_of_channels	38
4.25.2.2 chan_scan_param	38
4.26 wifi_chan_scan_param_set_t Struct Reference	39
4.26.1 Detailed Description	39
4.26.2 Field Documentation	39
4.26.2.1 chan_number	39
4.26.2.2 min_scan_time	39
4.26.2.3 max_scan_time	39
4.27 wifi_chanlist_t Struct Reference	39
4.27.1 Detailed Description	40
4.27.2 Field Documentation	40
4.27.2.1 num_chans	40
4.27.2.2 chan_info	40
4.28 wifi_channel_desc_t Struct Reference	40
4.28.1 Detailed Description	40
4.28.2 Field Documentation	41
4.28.2.1 start_freq	41
4.28.2.2 chan_width	41
4.28.2.3 chan_num	41
4.29 wifi_clock_sync_gpio_tsf_t Struct Reference	41
4.29.1 Detailed Description	41
4.29.2 Field Documentation	41
4.29.2.1 clock_sync_mode	42
4.29.2.2 clock_sync_Role	42
4.29.2.3 clock_sync_gpio_pin_number	42
4.29.2.4 clock_sync_gpio_level_toggle	42
4.29.2.5 clock_sync_gpio_pulse_width	42
4.30 wifi_cloud_keep_alive_t Struct Reference	42
4.30.1 Detailed Description	43
4.30.2 Field Documentation	43
4.30.2.1 mkeep_alive_id	43
4.30.2.2 enable	43
4.30.2.3 reset	43
4.30.2.4 cached	43
4.30.2.5 send_interval	43
4.30.2.6 retry_interval	43
4.30.2.7 retry_count	44
4.30.2.8 src_mac	44
4.30.2.9 dst_mac	44
4.30.2.10 src_ip	44
4.30.2.11 dst_ip	44

4.30.2.12 src_port	44
4.30.2.13 dst_port	44
4.30.2.14 pkt_len	44
4.30.2.15 packet	45
4.31 wifi_csi_config_params_t Struct Reference	45
4.31.1 Detailed Description	45
4.31.2 Field Documentation	45
4.31.2.1 bss_type	45
4.31.2.2 csi_enable	45
4.31.2.3 head_id	46
4.31.2.4 tail_id	46
4.31.2.5 csi_filter_cnt	46
4.31.2.6 chip_id	46
4.31.2.7 band_config	46
4.31.2.8 channel	46
4.31.2.9 csi_monitor_enable	46
4.31.2.10 ra4us	46
4.31.2.11 csi_filter	47
4.32 wifi_csi_filter_t Struct Reference	47
4.32.1 Detailed Description	47
4.32.2 Field Documentation	47
4.32.2.1 mac_addr	47
4.32.2.2 pkt_type	47
4.32.2.3 subtype	47
4.32.2.4 flags	48
4.33 wifi_cw_mode_ctrl_t Struct Reference	48
4.33.1 Detailed Description	48
4.33.2 Field Documentation	48
4.33.2.1 mode	48
4.33.2.2 channel	48
4.33.2.3 chanInfo	48
4.33.2.4 txPower	49
4.33.2.5 pktLength	49
4.33.2.6 rateInfo	49
4.34 wifi_data_rate_t Struct Reference	49
4.34.1 Detailed Description	49
4.34.2 Field Documentation	49
4.34.2.1 tx_data_rate	49
4.34.2.2 rx_data_rate	50
4.34.2.3 tx_bw	50
4.34.2.4 tx_gi	50
4.34.2.5 rx_bw	50

4.34.2.6 rx_gi	50
4.35 wifi_ds_rate Struct Reference	50
4.35.1 Detailed Description	51
4.35.2 Field Documentation	51
4.35.2.1 sub_command	51
4.35.2.2 rate_cfg	51
4.35.2.3 data_rate	51
4.35.2.4 param	51
4.36 wifi_ecsa_info Struct Reference	51
4.36.1 Field Documentation	52
4.36.1.1 bss_type	52
4.36.1.2 band_config	52
4.36.1.3 channel	52
4.37 wifi_ed_mac_ctrl_t Struct Reference	52
4.37.1 Detailed Description	52
4.37.2 Field Documentation	52
4.37.2.1 ed_ctrl_2g	53
4.37.2.2 ed_offset_2g	53
4.37.2.3 ed_ctrl_5g	53
4.37.2.4 ed_offset_5g	53
4.38 wifi_ext_coex_config_t Struct Reference	53
4.38.1 Detailed Description	53
4.38.2 Field Documentation	54
4.38.2.1 Enabled	54
4.38.2.2 IgnorePriority	54
4.38.2.3 DefaultPriority	54
4.38.2.4 EXT_RADIO_REQ_ip_gpio_num	54
4.38.2.5 EXT_RADIO_REQ_ip_gpio_polarity	54
4.38.2.6 EXT_RADIO_PRI_ip_gpio_num	54
4.38.2.7 EXT_RADIO_PRI_ip_gpio_polarity	54
4.38.2.8 WLAN_GRANT_op_gpio_num	55
4.38.2.9 WLAN_GRANT_op_gpio_polarity	55
4.38.2.10 reserved_1	55
4.38.2.11 reserved_2	55
4.39 wifi_ext_coex_stats_t Struct Reference	55
4.39.1 Detailed Description	55
4.39.2 Field Documentation	55
4.39.2.1 ext_radio_req_count	56
4.39.2.2 ext_radio_pri_count	56
4.39.2.3 wlan_grant_count	56
4.40 wififlt_cfg_t Struct Reference	56
4.40.1 Detailed Description	56

4.40.2 Field Documentation	56
4.40.2.1 criteria	56
4.40.2.2 nentries	57
4.40.2.3 mef_entry	57
4.41 wifi_frame_t Struct Reference	57
4.41.1 Field Documentation	57
4.41.1.1 frame_type	57
4.42 wifi_fw_version_ext_t Struct Reference	57
4.42.1 Detailed Description	57
4.42.2 Field Documentation	58
4.42.2.1 version_str_sel	58
4.42.2.2 version_str	58
4.43 wifi_fw_version_t Struct Reference	58
4.43.1 Detailed Description	58
4.43.2 Field Documentation	58
4.43.2.1 version_str	58
4.44 wifi_indrst_cfg_t Struct Reference	59
4.44.1 Detailed Description	59
4.44.2 Field Documentation	59
4.44.2.1 ir_mode	59
4.44.2.2 gpio_pin	59
4.45 wifi_mac_addr_t Struct Reference	59
4.45.1 Detailed Description	59
4.45.2 Field Documentation	60
4.45.2.1 mac	60
4.46 wifi_mef_entry_t Struct Reference	60
4.46.1 Detailed Description	60
4.46.2 Field Documentation	60
4.46.2.1 mode	60
4.46.2.2 action	60
4.46.2.3 filter_num	61
4.46.2.4 filter_item	61
4.46.2.5 rpn	61
4.47 wifi_mef_filter_t Struct Reference	61
4.47.1 Detailed Description	61
4.47.2 Field Documentation	61
4.47.2.1 fill_flag	62
4.47.2.2 type	62
4.47.2.3 pattern	62
4.47.2.4 offset	62
4.47.2.5 num_bytes	62
4.47.2.6 repeat	62

4.47.2.7 num_byte_seq	62
4.47.2.8 byte_seq	62
4.47.2.9 num_mask_seq	63
4.47.2.10 mask_seq	63
4.48 wifi_message Struct Reference	63
4.48.1 Field Documentation	63
4.48.1.1 event	63
4.48.1.2 reason	63
4.48.1.3 data	63
4.49 wifi_mfg_cmd_generic_cfg_t Struct Reference	64
4.49.1 Detailed Description	64
4.49.2 Field Documentation	64
4.49.2.1 mfg_cmd	64
4.49.2.2 action	64
4.49.2.3 device_id	64
4.49.2.4 error	64
4.49.2.5 data1	65
4.49.2.6 data2	65
4.49.2.7 data3	65
4.50 wifi_mfg_cmd_he_tb_tx_t Struct Reference	65
4.50.1 Field Documentation	65
4.50.1.1 mfg_cmd	65
4.50.1.2 action	66
4.50.1.3 device_id	66
4.50.1.4 error	66
4.50.1.5 enable	66
4.50.1.6 qnum	66
4.50.1.7 aid	66
4.50.1.8 axq_mu_timer	66
4.50.1.9 tx_power	67
4.51 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t Struct Reference	67
4.51.1 Field Documentation	67
4.51.1.1 mfg_cmd	67
4.51.1.2 action	67
4.51.1.3 device_id	68
4.51.1.4 error	68
4.51.1.5 enable_tx	68
4.51.1.6 standalone_hetb	68
4.51.1.7 frmCtl	68
4.51.1.8 duration	68
4.51.1.9 dest_addr	68
4.51.1.10 src_addr	68

4.51.1.11 trig_common_field	69
4.51.1.12 trig_user_info_field	69
4.51.1.13 basic_trig_user_info	69
4.52 wifi_mfg_cmd_otp_cal_data_rd_wr_t Struct Reference	69
4.52.1 Field Documentation	69
4.52.1.1 mfg_cmd	69
4.52.1.2 action	70
4.52.1.3 device_id	70
4.52.1.4 error	70
4.52.1.5 cal_data_status	70
4.52.1.6 cal_data_len	70
4.52.1.7 cal_data	70
4.53 wifi_mfg_cmd_otp_mac_addr_rd_wr_t Struct Reference	70
4.53.1 Field Documentation	71
4.53.1.1 mfg_cmd	71
4.53.1.2 action	71
4.53.1.3 device_id	71
4.53.1.4 error	71
4.53.1.5 mac_addr	71
4.54 wifi_mfg_cmd_tx_cont_t Struct Reference	71
4.54.1 Detailed Description	72
4.54.2 Field Documentation	72
4.54.2.1 mfg_cmd	72
4.54.2.2 action	72
4.54.2.3 device_id	72
4.54.2.4 error	72
4.54.2.5 enable_tx	72
4.54.2.6 cw_mode	72
4.54.2.7 payload_pattern	73
4.54.2.8 cs_mode	73
4.54.2.9 act_sub_ch	73
4.54.2.10 tx_rate	73
4.54.2.11 rsvd	73
4.55 wifi_mfg_cmd_tx_frame_t Struct Reference	74
4.55.1 Detailed Description	74
4.55.2 Field Documentation	74
4.55.2.1 mfg_cmd	74
4.55.2.2 action	75
4.55.2.3 device_id	75
4.55.2.4 error	75
4.55.2.5 enable	75
4.55.2.6 data_rate	75

4.55.2.7 frame_pattern	75
4.55.2.8 frame_length	75
4.55.2.9 bssid	75
4.55.2.10 adjust_burst_sifs	76
4.55.2.11 burst_sifs_in_us	76
4.55.2.12 short_preamble	76
4.55.2.13 act_sub_ch	76
4.55.2.14 short_gi	76
4.55.2.15 adv_coding	76
4.55.2.16 tx_bf	76
4.55.2.17 gf_mode	76
4.55.2.18 stbc	77
4.55.2.19 rsvd	77
4.55.2.20 signal_bw	77
4.55.2.21 NumPkt	77
4.55.2.22 MaxPE	77
4.55.2.23 BeamChange	77
4.55.2.24 Dcm	77
4.55.2.25 Doppler	77
4.55.2.26 MidP	78
4.55.2.27 QNum	78
4.56 wifi_mgmt_frame_t Struct Reference	78
4.56.1 Detailed Description	78
4.56.2 Field Documentation	78
4.56.2.1 frm_len	78
4.56.2.2 frame_type	79
4.56.2.3 frame_ctrl_flags	79
4.56.2.4 duration_id	79
4.56.2.5 addr1	79
4.56.2.6 addr2	79
4.56.2.7 addr3	79
4.56.2.8 seq_ctl	79
4.56.2.9 addr4	79
4.56.2.10 payload	80
4.57 wifi_nat_keep_alive_t Struct Reference	80
4.57.1 Detailed Description	80
4.57.2 Field Documentation	80
4.57.2.1 interval	80
4.57.2.2 dst_mac	80
4.57.2.3 dst_ip	80
4.57.2.4 dst_port	81
4.58 wifi_os_mem_info Struct Reference	81

4.58.1 Field Documentation	81
4.58.1.1 name	81
4.58.1.2 size	81
4.58.1.3 line_num	81
4.58.1.4 alloc_cnt	81
4.58.1.5 free_cnt	82
4.59 wifi_pmf_params_t Struct Reference	82
4.59.1 Field Documentation	82
4.59.1.1 mfpc	82
4.59.1.2 mfpr	82
4.60 wifi_rate_cfg_t Struct Reference	82
4.60.1 Detailed Description	83
4.60.2 Field Documentation	83
4.60.2.1 rate_format	83
4.60.2.2 rate_index	83
4.60.2.3 rate	83
4.60.2.4 nss	83
4.60.2.5 rate_setting	83
4.61 wifi_remain_on_channel_t Struct Reference	84
4.61.1 Detailed Description	84
4.61.2 Field Documentation	84
4.61.2.1 remove	84
4.61.2.2 status	84
4.61.2.3 bandcfg	84
4.61.2.4 channel	84
4.61.2.5 remain_period	85
4.62 wifi_rf_channel_t Struct Reference	85
4.62.1 Detailed Description	85
4.62.2 Field Documentation	85
4.62.2.1 current_channel	85
4.62.2.2 rf_type	85
4.63 wifi_rssi_info_t Struct Reference	86
4.63.1 Detailed Description	86
4.63.2 Field Documentation	86
4.63.2.1 data_rssi_last	86
4.63.2.2 data_nf_last	86
4.63.2.3 data_rssi_avg	86
4.63.2.4 data_nf_avg	86
4.63.2.5 bcn_snr_last	87
4.63.2.6 bcn_snr_avg	87
4.63.2.7 data_snr_last	87
4.63.2.8 data_snr_avg	87

4.63.2.9 bcn_rssi_last	87
4.63.2.10 bcn_nf_last	87
4.63.2.11 bcn_rssi_avg	87
4.63.2.12 bcn_nf_avg	88
4.64 wifi_rupwrlimit_config_t Struct Reference	88
4.64.1 Field Documentation	88
4.64.1.1 start_freq	88
4.64.1.2 width	88
4.64.1.3 chan_num	88
4.64.1.4 ruPower	89
4.65 wifi_rutxpwrlimit_t Struct Reference	89
4.65.1 Detailed Description	89
4.65.2 Field Documentation	89
4.65.2.1 num_chans	89
4.65.2.2 rupwrlimit_config	89
4.66 wifi_scan_chan_list_t Struct Reference	90
4.66.1 Detailed Description	90
4.66.2 Field Documentation	90
4.66.2.1 num_of_chan	90
4.66.2.2 chan_number	90
4.67 wifi_scan_channel_list_t Struct Reference	90
4.67.1 Detailed Description	90
4.67.2 Field Documentation	91
4.67.2.1 radio_type	91
4.67.2.2 chan_number	91
4.67.2.3 scan_type	91
4.67.2.4 scan_time	91
4.68 wifi_scan_params_t Struct Reference	91
4.68.1 Field Documentation	91
4.68.1.1 bssid	92
4.68.1.2 ssid	92
4.68.1.3 channel	92
4.68.1.4 bss_type	92
4.68.1.5 scan_duration	92
4.68.1.6 split_scan_delay	92
4.69 wifi_scan_params_v2_t Struct Reference	92
4.69.1 Detailed Description	93
4.69.2 Field Documentation	93
4.69.2.1 scan_only	93
4.69.2.2 is_bssid	93
4.69.2.3 is_ssid	93
4.69.2.4 bssid	93

4.69.2.5 ssid	93
4.69.2.6 num_channels	93
4.69.2.7 chan_list	94
4.69.2.8 num_probes	94
4.69.2.9 scan_chan_gap	94
4.69.2.10 cb	94
4.70 wifi_scan_result2 Struct Reference	94
4.70.1 Detailed Description	95
4.70.2 Field Documentation	95
4.70.2.1 bssid	95
4.70.2.2 is_ibss_bit_set	95
4.70.2.3 ssid	95
4.70.2.4 ssid_len	96
4.70.2.5 Channel	96
4.70.2.6 RSSI	96
4.70.2.7 beacon_period	96
4.70.2.8 dtim_period	96
4.70.2.9 WPA_WPA2_WEP	96
4.70.2.10 wpa_mcstCipher	96
4.70.2.11 wpa_ucstCipher	96
4.70.2.12 rsn_mcstCipher	97
4.70.2.13 rsn_ucstCipher	97
4.70.2.14 is_pmf_required	97
4.70.2.15 ap_mfpc	97
4.70.2.16 ap_mfpr	97
4.70.2.17 ap_pwe	97
4.70.2.18 phtcap_ie_present	97
4.70.2.19 phtinfo_ie_present	98
4.70.2.20 pvhtcap_ie_present	98
4.70.2.21 phecap_ie_present	98
4.70.2.22 wmm_ie_present	98
4.70.2.23 band	98
4.70.2.24 wps_IE_exist	98
4.70.2.25 wps_session	98
4.70.2.26 wpa2_entp_IE_exist	98
4.70.2.27 trans_mode	99
4.70.2.28 trans_bssid	99
4.70.2.29 trans_ssid	99
4.70.2.30 trans_ssid_len	99
4.70.2.31 mbo_assoc_disallowed	99
4.70.2.32 mdid	99
4.70.2.33 neighbor_report_supported	99

4.70.2.34 bss_transition_supported	100
4.71 wifi_sta_info_t Struct Reference	100
4.71.1 Detailed Description	100
4.71.2 Field Documentation	100
4.71.2.1 mac	100
4.71.2.2 power_mgmt_status	100
4.71.2.3 rssi	100
4.72 wifi_sta_list_t Struct Reference	101
4.72.1 Detailed Description	101
4.72.2 Field Documentation	101
4.72.2.1 count	101
4.73 wifi_sub_band_set_t Struct Reference	101
4.73.1 Detailed Description	101
4.73.2 Field Documentation	101
4.73.2.1 first_chan	102
4.73.2.2 no_of_chan	102
4.73.2.3 max_tx_pwr	102
4.74 wifi_tbtt_offset_t Struct Reference	102
4.74.1 Detailed Description	102
4.74.2 Field Documentation	102
4.74.2.1 min_tbtt_offset	102
4.74.2.2 max_tbtt_offset	103
4.74.2.3 avg_tbtt_offset	103
4.75 wifi_tcp_keep_alive_t Struct Reference	103
4.75.1 Detailed Description	103
4.75.2 Field Documentation	103
4.75.2.1 enable	103
4.75.2.2 reset	104
4.75.2.3 timeout	104
4.75.2.4 interval	104
4.75.2.5 max_keep_alives	104
4.75.2.6 dst_mac	104
4.75.2.7 dst_ip	104
4.75.2.8 dst_tcp_port	104
4.75.2.9 src_tcp_port	104
4.75.2.10 seq_no	105
4.76 wifi_tsf_info_t Struct Reference	105
4.76.1 Detailed Description	105
4.76.2 Field Documentation	105
4.76.2.1 tsf_format	105
4.76.2.2 tsf_info	105
4.76.2.3 tsf	105

4.76.2.4 tsf_offset	106
4.77 wifi_twt_information_t Struct Reference	106
4.77.1 Field Documentation	106
4.77.1.1 flow_identifier	106
4.77.1.2 suspend_duration	106
4.77.1.3 information_state	106
4.78 wifi_twt_report_t Struct Reference	107
4.78.1 Detailed Description	107
4.78.2 Field Documentation	107
4.78.2.1 type	107
4.78.2.2 length	107
4.78.2.3 reserve	107
4.78.2.4 data	107
4.79 wifi_twt_setup_config_t Struct Reference	108
4.79.1 Detailed Description	108
4.79.2 Field Documentation	108
4.79.2.1 implicit	108
4.79.2.2 announced	108
4.79.2.3 trigger_enabled	108
4.79.2.4 twt_info_disabled	108
4.79.2.5 negotiation_type	109
4.79.2.6 twt_wakeup_duration	109
4.79.2.7 flow_identifier	109
4.79.2.8 hard_constraint	109
4.79.2.9 twt_exponent	109
4.79.2.10 twt_mantissa	109
4.79.2.11 twt_request	109
4.80 wifi_twt_tearardown_config_t Struct Reference	110
4.80.1 Detailed Description	110
4.80.2 Field Documentation	110
4.80.2.1 flow_identifier	110
4.80.2.2 negotiation_type	110
4.80.2.3 teardown_all_twt	110
4.81 wifi_tx_power_t Struct Reference	110
4.81.1 Detailed Description	111
4.81.2 Field Documentation	111
4.81.2.1 current_level	111
4.81.2.2 max_power	111
4.81.2.3 min_power	111
4.82 wifi_txpwrlimit_config_t Struct Reference	111
4.82.1 Detailed Description	111
4.82.2 Field Documentation	112

4.82.2.1 num_mod_grps	112
4.82.2.2 chan_desc	112
4.82.2.3 txpwrlimit_entry	112
4.83 wifi_txpwrlimit_entry_t Struct Reference	112
4.83.1 Detailed Description	112
4.83.2 Field Documentation	113
4.83.2.1 mod_group	113
4.83.2.2 tx_power	113
4.84 wifi_txpwrlimit_t Struct Reference	113
4.84.1 Detailed Description	113
4.84.2 Field Documentation	113
4.84.2.1 subband	113
4.84.2.2 num_chans	114
4.84.2.3 txpwrlimit_config	114
4.85 wifi_uap_client_disassoc_t Struct Reference	114
4.85.1 Field Documentation	114
4.85.1.1 reason_code	114
4.85.1.2 sta_addr	114
4.86 wifi_wowlan_pattern_t Struct Reference	115
4.86.1 Field Documentation	115
4.86.1.1 pkt_offset	115
4.86.1.2 pattern_len	115
4.86.1.3 pattern	115
4.86.1.4 mask	115
4.87 wifi_wowlan_ptn_cfg_t Struct Reference	115
4.87.1 Detailed Description	116
4.87.2 Field Documentation	116
4.87.2.1 enable	116
4.87.2.2 n_patterns	116
4.87.2.3 patterns	116
4.88 wlan_cipher Struct Reference	116
4.88.1 Detailed Description	117
4.88.2 Field Documentation	117
4.88.2.1 none	117
4.88.2.2 wep40	117
4.88.2.3 wep104	117
4.88.2.4 tkip	117
4.88.2.5 ccmp	117
4.88.2.6 aes_128_cmac	117
4.88.2.7 gcmp	118
4.88.2.8 sms4	118
4.88.2.9 gcmp_256	118

4.88.2.10 ccmp_256	118
4.88.2.11 rsvd	118
4.88.2.12 bip_gmac_128	118
4.88.2.13 bip_gmac_256	118
4.88.2.14 bip_cmac_256	118
4.88.2.15 gtk_not_used	119
4.88.2.16 rsvd2	119
4.89 wlan_ieee80211_struct Struct Reference	119
4.89.1 Field Documentation	119
4.89.1.1 ps_null_interval	119
4.89.1.2 multiple_dtim_interval	119
4.89.1.3 listen_interval	120
4.89.1.4 adhoc_awake_period	120
4.89.1.5 bcn_miss_timeout	120
4.89.1.6 delay_to_ps	120
4.89.1.7 ps_mode	120
4.90 wlan_ip_config Struct Reference	120
4.90.1 Detailed Description	120
4.90.2 Field Documentation	121
4.90.2.1 ipv6	121
4.90.2.2 ipv6_count	121
4.90.2.3 ipv4	121
4.91 wlan_message Struct Reference	121
4.91.1 Field Documentation	121
4.91.1.1 id	121
4.91.1.2 data	122
4.92 wlan_network Struct Reference	122
4.92.1 Detailed Description	123
4.92.2 Field Documentation	123
4.92.2.1 id	123
4.92.2.2 wps_network	123
4.92.2.3 name	123
4.92.2.4 ssid	123
4.92.2.5 bssid	124
4.92.2.6 channel	124
4.92.2.7 sec_channel_offset	124
4.92.2.8 acs_band	124
4.92.2.9 rssi	124
4.92.2.10 ht_capab	124
4.92.2.11 vht_capab	124
4.92.2.12 vht_oper_chwidth	125
4.92.2.13 he_oper_chwidth	125

4.92.2.14 type	125
4.92.2.15 role	125
4.92.2.16 security	125
4.92.2.17 ip	125
4.92.2.18 ssid_specific	125
4.92.2.19 bssid_specific	126
4.92.2.20 channel_specific	126
4.92.2.21 security_specific	126
4.92.2.22 dot11n	126
4.92.2.23 dot11ac	126
4.92.2.24 dot11ax	126
4.92.2.25 mdid	127
4.92.2.26 ft_1x	127
4.92.2.27 ft_psk	127
4.92.2.28 ft_sae	127
4.92.2.29 beacon_period	127
4.92.2.30 dtim_period	127
4.92.2.31 wlan_capa	127
4.92.2.32 btm_mode	127
4.92.2.33 bss_transition_supported	128
4.92.2.34 neighbor_report_supported	128
4.93 wlan_network_security Struct Reference	128
4.93.1 Detailed Description	129
4.93.2 Field Documentation	129
4.93.2.1 type	129
4.93.2.2 key_mgmt	130
4.93.2.3 mcstCipher	130
4.93.2.4 ucstCipher	130
4.93.2.5 pkc	130
4.93.2.6 group_cipher	130
4.93.2.7 pairwise_cipher	130
4.93.2.8 group_mgmt_cipher	130
4.93.2.9 is_pmf_required	130
4.93.2.10 psk	131
4.93.2.11 psk_len	131
4.93.2.12 password	131
4.93.2.13 password_len	131
4.93.2.14 sae_groups	131
4.93.2.15 pwe_derivation	131
4.93.2.16 transition_disable	131
4.93.2.17 pmk	132
4.93.2.18 pmk_valid	132

4.93.2.19 mfpc	132
4.93.2.20 mfpr	132
4.93.2.21 wpa3_sb	132
4.93.2.22 wpa3_sb_192	132
4.93.2.23 eap_ver	132
4.93.2.24 peap_label	133
4.93.2.25 eap_crypto_binding	133
4.93.2.26 eap_result_ind	133
4.93.2.27 tls_cipher	133
4.93.2.28 identity	133
4.93.2.29 anonymous_identity	133
4.93.2.30 eap_password	133
4.93.2.31 ca_cert_data	134
4.93.2.32 ca_cert_len	134
4.93.2.33 client_cert_data	134
4.93.2.34 client_cert_len	134
4.93.2.35 client_key_data	134
4.93.2.36 client_key_len	134
4.93.2.37 client_key_passwd	134
4.93.2.38 ca_cert_hash	134
4.93.2.39 domain_match	135
4.93.2.40 domain_suffix_match	135
4.93.2.41 ca_cert2_data	135
4.93.2.42 ca_cert2_len	135
4.93.2.43 client_cert2_data	135
4.93.2.44 client_cert2_len	135
4.93.2.45 client_key2_data	135
4.93.2.46 client_key2_len	135
4.93.2.47 client_key2_passwd	136
4.93.2.48 dh_data	136
4.93.2.49 dh_len	136
4.93.2.50 server_cert_data	136
4.93.2.51 server_cert_len	136
4.93.2.52 server_key_data	136
4.93.2.53 server_key_len	136
4.93.2.54 server_key_passwd	136
4.93.2.55 nusers	137
4.93.2.56 identities	137
4.93.2.57 passwords	137
4.93.2.58 pac_opaque_encr_key	137
4.93.2.59 a_id	137
4.93.2.60 fast_prov	137

4.93.2.61 dpp_connector	137
4.93.2.62 dpp_c_sign_key	138
4.93.2.63 dpp_net_access_key	138
4.94 wlan_nlist_report_param Struct Reference	138
4.94.1 Field Documentation	138
4.94.1.1 nlist_mode	138
4.94.1.2 num_channels	138
4.94.1.3 channels	139
4.94.1.4 btm_mode	139
4.94.1.5 bssid	139
4.94.1.6 dialog_token	139
4.94.1.7 dst_addr	139
4.94.1.8 protect	139
4.95 wlan_rrm_beacon_report_data Struct Reference	139
4.95.1 Field Documentation	140
4.95.1.1 token	140
4.95.1.2 ssid	140
4.95.1.3 ssid_length	140
4.95.1.4 bssid	140
4.95.1.5 channel	140
4.95.1.6 channel_num	140
4.95.1.7 last_ind	140
4.95.1.8 duration	141
4.95.1.9 report_detail	141
4.95.1.10 bits_field	141
4.96 wlan_rrm_neighbor_ap_t Struct Reference	141
4.96.1 Field Documentation	141
4.96.1.1 ssid	141
4.96.1.2 bssid	141
4.96.1.3 bssidInfo	142
4.96.1.4 op_class	142
4.96.1.5 channel	142
4.96.1.6 phy_type	142
4.96.1.7 freq	142
4.97 wlan_rrm_neighbor_report_t Struct Reference	142
4.97.1 Field Documentation	142
4.97.1.1 neighbor_ap	143
4.97.1.2 neighbor_cnt	143
4.98 wlan_rrm_scan_cb_param Struct Reference	143
4.98.1 Field Documentation	143
4.98.1.1 rep_data	143
4.98.1.2 dialog_tok	143

4.98.1.3 dst_addr	143
4.98.1.4 protect	144
4.99 wlan_scan_result Struct Reference	144
4.99.1 Detailed Description	145
4.99.2 Field Documentation	145
4.99.2.1 ssid	145
4.99.2.2 ssid_len	145
4.99.2.3 bssid	145
4.99.2.4 channel	145
4.99.2.5 type	145
4.99.2.6 role	145
4.99.2.7 dot11n	146
4.99.2.8 dot11ac	146
4.99.2.9 dot11ax	146
4.99.2.10 wmm	146
4.99.2.11 wps	146
4.99.2.12 wps_session	146
4.99.2.13 wep	146
4.99.2.14 wpa	147
4.99.2.15 wpa2	147
4.99.2.16 wpa2_sha256	147
4.99.2.17 wpa3_sae	147
4.99.2.18 wpa2_entp	147
4.99.2.19 wpa2_entp_sha256	147
4.99.2.20 wpa3_1x_sha256	147
4.99.2.21 wpa3_1x_sha384	147
4.99.2.22 ft_1x	148
4.99.2.23 ft_1x_sha384	148
4.99.2.24 ft_psk	148
4.99.2.25 ft_sae	148
4.99.2.26 rssi	148
4.99.2.27 trans_ssid	148
4.99.2.28 trans_ssid_len	148
4.99.2.29 trans_bssid	149
4.99.2.30 beacon_period	149
4.99.2.31 dtim_period	149
4.99.2.32 ap_mfpc	149
4.99.2.33 ap_mfpr	149
4.99.2.34 ap_pwe	149
4.99.2.35 neighbor_report_supported	149
4.99.2.36 bss_transition_supported	150
4.100 wnm_sleep_result_t Struct Reference	150

4.100.1 Field Documentation	150
4.100.1.1 action	150
4.100.1.2 result	150
5 File Documentation	151
5.1 cli.h File Reference	151
5.1.1 Detailed Description	151
5.1.2 Usage	151
5.1.3 Function Documentation	151
5.1.3.1 lookup_command()	151
5.1.3.2 cli_register_command()	151
5.1.3.3 cli_unregister_command()	152
5.1.3.4 cli_init()	152
5.1.3.5 cli_deinit()	153
5.1.3.6 cli_stop()	153
5.1.3.7 cli_register_commands()	153
5.1.3.8 cli_unregister_commands()	154
5.1.3.9 cli_get_cmd_buffer()	154
5.1.3.10 cli_submit_cmd_buffer()	154
5.1.3.11 cli_add_history_hook()	155
5.1.3.12 help_command()	155
5.1.4 Macro Documentation	155
5.1.4.1 CONFIG_APP_FRM_CLI_HISTORY	155
5.1.5 Typedef Documentation	155
5.1.5.1 cli_name_val_get	155
5.1.5.2 cli_name_val_set	156
5.2 dhcp-server.h File Reference	156
5.2.1 Detailed Description	156
5.2.2 Function Documentation	156
5.2.2.1 dhcpcd_cli_init()	156
5.2.2.2 dhcpcd_cli_deinit()	156
5.2.2.3 dhcp_server_start()	157
5.2.2.4 dhcp_enable_dns_server()	157
5.2.2.5 dhcp_server_stop()	158
5.2.2.6 dhcp_server_lease_timeout()	158
5.2.2.7 dhcp_get_ip_from_mac()	158
5.2.2.8 dhcp_stat()	159
5.2.3 Macro Documentation	159
5.2.3.1 MAX_QNAME_SIZE	159
5.2.4 Enumeration Type Documentation	159
5.2.4.1 wm_dhcpcd_errno	159
5.3 iperf.h File Reference	160

5.3.1 Function Documentation	160
5.3.1.1 iperf_cli_init()	160
5.3.1.2 iperf_cli_deinit()	160
5.3.2 Macro Documentation	160
5.3.2.1 iperf_e	160
5.3.2.2 iperf_w	161
5.4 osa.h File Reference	161
5.4.1 Function Documentation	161
5.4.1.1 OSA_TimerCreate()	161
5.4.1.2 OSA_TimerActivate()	161
5.4.1.3 OSA_TimerChange()	162
5.4.1.4 OSA_TimerIsRunning()	162
5.4.1.5 OSA_TimerGetContext()	163
5.4.1.6 OSA_TimerReset()	163
5.4.1.7 OSA_TimerDeactivate()	164
5.4.1.8 OSA_TimerDestroy()	164
5.4.1.9 OSA_RWLockCreateWithCB()	164
5.4.1.10 OSA_RWLockCreate()	165
5.4.1.11 OSA_RWLockDestroy()	165
5.4.1.12 OSA_RWLockWriteLock()	165
5.4.1.13 OSA_RWLockWriteUnlock()	166
5.4.1.14 OSA_RWLockReadLock()	166
5.4.1.15 OSA_RWLockReadUnlock()	167
5.4.1.16 OSA_SetupIdleFunction()	167
5.4.1.17 OSA_SetupTickFunction()	167
5.4.1.18 OSA_RemoveIdleFunction()	168
5.4.1.19 OSA_RemoveTickFunction()	168
5.4.1.20 OSA_Srand()	169
5.4.1.21 OSA_Rand()	169
5.4.1.22 OSA_RandRange()	169
5.4.1.23 OSA_DumpThreadInfo()	170
5.4.1.24 OSA_ThreadSelfComplete()	170
5.4.1.25 OSA_MsgQWaiting()	170
5.4.2 Macro Documentation	170
5.4.2.1 MAX_CUSTOM_HOOKS	170
5.4.3 Typedef Documentation	171
5.4.3.1 cb_fn	171
5.4.4 Variable Documentation	171
5.4.4.1 g_osa_tick_hooks	171
5.4.4.2 g_osa_idle_hooks	171
5.4.4.3 wm_rand_seed	171
5.5 README.txt File Reference	171

5.6 wifi-decl.h File Reference	171
5.6.1 Macro Documentation	171
5.6.1.1 MLAN_MAC_ADDR_LENGTH	172
5.6.1.2 MLAN_MAX_VER_STR_LEN	172
5.6.1.3 WIFI_MAX_CHANNEL_NUM	172
5.6.1.4 PMK_BIN_LEN	172
5.6.1.5 PMK_HEX_LEN	172
5.6.1.6 MOD_GROUPS	172
5.6.1.7 WIFI_SUPPORT_11AX	172
5.6.1.8 WIFI_SUPPORT_11AC	172
5.6.1.9 WIFI_SUPPORT_11N	173
5.6.1.10 WIFI_SUPPORT_LEGACY	173
5.6.1.11 BSS_TYPE_STA	173
5.6.1.12 BSS_TYPE_UAP	173
5.6.1.13 UAP_DEFAULT_CHANNEL	173
5.6.1.14 UAP_DEFAULT_BANDWIDTH	173
5.6.1.15 UAP_DEFAULT_BEACON_PERIOD	173
5.6.1.16 UAP_DEFAULT_HIDDEN_SSID	173
5.6.1.17 MLAN_MAX_SSID_LENGTH	174
5.6.1.18 MLAN_MAX_PASS_LENGTH	174
5.6.1.19 BIT	174
5.6.1.20 WOWLAN_MAX_PATTERN_LEN	174
5.6.1.21 WOWLAN_MAX_OFFSET_LEN	174
5.6.1.22 MAX_NUM_FILTERS	174
5.6.1.23 MEF_MODE_HOST_SLEEP	174
5.6.1.24 MEF_MODE_NON_HOST_SLEEP	175
5.6.1.25 MEF_ACTION_WAKE	175
5.6.1.26 MEF_ACTION_ALLOW	175
5.6.1.27 MEF_ACTION_ALLOW_AND_WAKEUP_HOST	175
5.6.1.28 MEF_AUTO_ARP	175
5.6.1.29 MEF_AUTO_PING	175
5.6.1.30 MEF_NS_RESP	175
5.6.1.31 MEF_MAGIC_PKT	175
5.6.1.32 CRITERIA_BROADCAST	176
5.6.1.33 CRITERIA_UNICAST	176
5.6.1.34 CRITERIA_MULTICAST	176
5.6.1.35 MAX_NUM_ENTRIES	176
5.6.1.36 MAX_NUM_BYTE_SEQ	176
5.6.1.37 MAX_NUM_MASK_SEQ	176
5.6.1.38 OPERAND_DNUM	176
5.6.1.39 OPERAND_BYTE_SEQ	176
5.6.1.40 MAX_OPERAND	177

5.6.1.41 TYPE_BYTE_EQ	177
5.6.1.42 TYPE_DNUM_EQ	177
5.6.1.43 TYPE_BIT_EQ	177
5.6.1.44 RPN_TYPE_AND	177
5.6.1.45 RPN_TYPE_OR	177
5.6.1.46 ICMP_OF_IP_PROTOCOL	177
5.6.1.47 TCP_OF_IP_PROTOCOL	177
5.6.1.48 UDP_OF_IP_PROTOCOL	178
5.6.1.49 IPV4_PKT_OFFSET	178
5.6.1.50 IP_PROTOCOL_OFFSET	178
5.6.1.51 PORT_PROTOCOL_OFFSET	178
5.6.1.52 FILLING_TYPE	178
5.6.1.53 FILLING_PATTERN	178
5.6.1.54 FILLING_OFFSET	178
5.6.1.55 FILLING_NUM_BYTES	178
5.6.1.56 FILLING_REPEAT	179
5.6.1.57 FILLING_BYTE_SEQ	179
5.6.1.58 FILLING_MASK_SEQ	179
5.6.1.59 MKEEP_ALIVE_IP_PKT_MAX	179
5.6.1.60 WLAN_BTWT_REPORT_LEN	179
5.6.1.61 WLAN_BTWT_REPORT_MAX_NUM	179
5.6.1.62 BAND_SPECIFIED	179
5.6.1.63 MAX_CHANNEL_LIST	179
5.6.1.64 MAX_NUM_SSID	180
5.6.1.65 MAX_FUNC_SYMBOL_LEN	180
5.6.1.66 OS_MEM_STAT_TABLE_SIZE	180
5.6.1.67 CSI_FILTER_MAX	180
5.6.2 Enumeration Type Documentation	180
5.6.2.1 wifi_bss_security	180
5.6.2.2 wifi_bss_features	180
5.6.2.3 wlan_type	181
5.6.2.4 wifi_ds_command_type	181
5.6.2.5 wifi_SubBand_t	181
5.6.2.6 wifi_frame_type_t	182
5.7 wifi.h File Reference	182
5.7.1 Function Documentation	182
5.7.1.1 wifi_init()	182
5.7.1.2 wifi_init_fcc()	183
5.7.1.3 wifi_deinit()	183
5.7.1.4 wifi_set_tx_status()	183
5.7.1.5 wifi_set_rx_status()	184
5.7.1.6 reset_ie_index()	184

5.7.1.7	wifi_register_data_input_callback()	184
5.7.1.8	wifi_deregister_data_input_callback()	185
5.7.1.9	wifi_register_amsdu_data_input_callback()	185
5.7.1.10	wifi_deregister_amsdu_data_input_callback()	185
5.7.1.11	wifi_register_deliver_packet_above_callback()	185
5.7.1.12	wifi_deregister_deliver_packet_above_callback()	185
5.7.1.13	wifi_register_wrapper_net_is_ip_or_ipv6_callback()	186
5.7.1.14	wifi_deregister_wrapper_net_is_ip_or_ipv6_callback()	186
5.7.1.15	wifi_add_to_bypassq()	186
5.7.1.16	wifi_low_level_output()	186
5.7.1.17	wifi_set_packet_retry_count()	187
5.7.1.18	wifi_sta_ampdu_tx_enable()	187
5.7.1.19	wifi_sta_ampdu_tx_disable()	187
5.7.1.20	wifi_sta_ampdu_tx_enable_per_tid()	187
5.7.1.21	wifi_sta_ampdu_tx_enable_per_tid_is_allowed()	188
5.7.1.22	wifi_sta_ampdu_rx_enable()	188
5.7.1.23	wifi_sta_ampdu_rx_enable_per_tid()	188
5.7.1.24	wifi_sta_ampdu_rx_enable_per_tid_is_allowed()	188
5.7.1.25	wifi_uap_ampdu_rx_enable()	189
5.7.1.26	wifi_uap_ampdu_rx_enable_per_tid()	189
5.7.1.27	wifi_uap_ampdu_rx_enable_per_tid_is_allowed()	189
5.7.1.28	wifi_uap_ampdu_rx_disable()	189
5.7.1.29	wifi_uap_ampdu_tx_enable()	190
5.7.1.30	wifi_uap_ampdu_tx_enable_per_tid()	190
5.7.1.31	wifi_uap_ampdu_tx_enable_per_tid_is_allowed()	190
5.7.1.32	wifi_uap_ampdu_tx_disable()	190
5.7.1.33	wifi_sta_ampdu_rx_disable()	190
5.7.1.34	wifi_get_device_mac_addr()	191
5.7.1.35	wifi_get_device_uap_mac_addr()	191
5.7.1.36	wifi_get_device_firmware_version_ext()	191
5.7.1.37	wifi_get_last_cmd_sent_ms()	192
5.7.1.38	wifi_get_value1()	192
5.7.1.39	wifi_get_outbuf()	192
5.7.1.40	wifi_config_roaming()	192
5.7.1.41	wifi_config_bgscan_and_rssi()	192
5.7.1.42	wifi_stop_bgscan()	192
5.7.1.43	wifi_update_last_cmd_sent_ms()	193
5.7.1.44	wifi_register_event_queue()	193
5.7.1.45	wifi_unregister_event_queue()	193
5.7.1.46	wifi_get_scan_result()	194
5.7.1.47	wifi_get_scan_result_count()	194
5.7.1.48	wifi_uap_bss_sta_list()	194

5.7.1.49	wifi_sta_deauth()	195
5.7.1.50	wifi_enable_low_pwr_mode()	195
5.7.1.51	wifi_set_cal_data()	195
5.7.1.52	wifi_set_mac_addr()	196
5.7.1.53	_wifi_set_mac_addr()	196
5.7.1.54	wifi_get_wpa_ie_in_assoc()	196
5.7.1.55	wifi_add_mcast_filter()	197
5.7.1.56	wifi_remove_mcast_filter()	197
5.7.1.57	wifi_get_ipv4_multicast_mac()	197
5.7.1.58	wifi_get_ipv6_multicast_mac()	198
5.7.1.59	wifi_set_antenna()	198
5.7.1.60	wifi_get_antenna()	198
5.7.1.61	wifi_process_hs_cfg_resp()	199
5.7.1.62	wifi_process_ps_enh_response()	199
5.7.1.63	wifi_uap_rates_getset()	199
5.7.1.64	wifi_uap_sta_ageout_timer_getset()	199
5.7.1.65	wifi_uap_ps_sta_ageout_timer_getset()	199
5.7.1.66	wifi_mem_access()	199
5.7.1.67	wifi_scan_process_results()	200
5.7.1.68	wifi_get_region_code()	200
5.7.1.69	wifi_set_region_code()	200
5.7.1.70	wifi_set_country_code()	201
5.7.1.71	wifi_get_country_code()	201
5.7.1.72	wifi_set_country_ie_ignore()	201
5.7.1.73	wifi_get_uap_channel()	201
5.7.1.74	wifi_uap_pmf_getset()	202
5.7.1.75	wifi_uap_enable_11d_support()	202
5.7.1.76	wifi_11d_is_channel_allowed()	203
5.7.1.77	get_sub_band_from_region_code()	203
5.7.1.78	get_sub_band_from_region_code_5ghz()	203
5.7.1.79	wifi_enable_11d_support()	203
5.7.1.80	wifi_enable_uap_11d_support()	203
5.7.1.81	wifi_disable_11d_support()	203
5.7.1.82	wifi_disable_uap_11d_support()	203
5.7.1.83	wifi_set_region_power_cfg()	204
5.7.1.84	wifi_set_txbfcap()	204
5.7.1.85	wifi_set_htcapinfo()	204
5.7.1.86	wifi_set_httxcfg()	204
5.7.1.87	wifi_uap_set_httxcfg()	204
5.7.1.88	wifi_uap_set_httxcfg_int()	204
5.7.1.89	wifi_get_tx_power()	204
5.7.1.90	wifi_set_tx_power()	205

5.7.1.91 wrapper_wlan_cmd_get_hw_spec()	205
5.7.1.92 set_event_chanswann()	205
5.7.1.93 clear_event_chanswann()	205
5.7.1.94 wifi_set_ps_cfg()	205
5.7.1.95 wifi_send_hs_cfg_cmd()	205
5.7.1.96 wrapper_wlan_11d_support_is_enabled()	206
5.7.1.97 wrapper_wlan_11d_clear_parsedtable()	206
5.7.1.98 wrapper_clear_media_connected_event()	206
5.7.1.99 wifi_uap_ps_inactivity_sleep_exit()	206
5.7.1.100 wifi_uap_ps_inactivity_sleep_enter()	206
5.7.1.101 wifi_enter_ieee_power_save()	206
5.7.1.102 wifi_exit_ieee_power_save()	207
5.7.1.103 wifi_enter_wnm_power_save()	207
5.7.1.104 wifi_exit_wnm_power_save()	207
5.7.1.105 wifi_enter_deepsleep_power_save()	207
5.7.1.106 wifi_exit_deepsleep_power_save()	207
5.7.1.107 wifi_set_power_save_mode()	207
5.7.1.108 wifi_get_wakeup_reason()	207
5.7.1.109 send_sleep_confirm_command()	208
5.7.1.110 wifi_configure_listen_interval()	208
5.7.1.111 wifi_configure_delay_to_ps()	208
5.7.1.112 wifi_get_listen_interval()	208
5.7.1.113 wifi_get_delay_to_ps()	208
5.7.1.114 wifi_configure_null_pkt_interval()	208
5.7.1.115 wrapper_wifi_assoc()	208
5.7.1.116 wifi_uap_enable_sticky_bit()	209
5.7.1.117 wifi_get_xfer_pending()	209
5.7.1.118 wifi_set_xfer_pending()	209
5.7.1.119 wrapper_wlan_cmd_11n_ba_stream_timeout()	209
5.7.1.120 wifi_set_txratecfg()	209
5.7.1.121 wifi_get_txratecfg()	209
5.7.1.122 wifi_wake_up_card()	209
5.7.1.123 wifi_tx_card_awake_lock()	210
5.7.1.124 wifi_tx_card_awake_unlock()	210
5.7.1.125 wrapper_wlan_11d_enable()	210
5.7.1.126 wrapper_wlan_uap_11d_enable()	210
5.7.1.127 wifi_11h_enable()	210
5.7.1.128 wrapper_wlan_cmd_11n_addba_rspgen()	210
5.7.1.129 wrapper_wlan_cmd_11n_delba_rspgen()	210
5.7.1.130 wrapper_wlan_ecsa_enable()	211
5.7.1.131 wifi_uap_start()	211
5.7.1.132 wrapper_wlan_sta_ampdu_enable()	211

5.7.1.133 wrapper_wlan_uap_ampdu_enable()	211
5.7.1.134 wifi_set_packet_filters()	211
5.7.1.135 wifi_uap_stop()	212
5.7.1.136 wifi_uap_do_acs()	212
5.7.1.137 wifi_uap_config_wifi_capa()	212
5.7.1.138 wifi_get_fw_info()	212
5.7.1.139 wifi_get_data_rate()	212
5.7.1.140 wifi_uap_set_bandwidth()	212
5.7.1.141 wifi_uap_get_bandwidth()	213
5.7.1.142 wifi_uap_get_pmfcfg()	213
5.7.1.143 wifi_set_rts()	213
5.7.1.144 wifi_set_frag()	213
5.7.1.145 wifi_same_ess_ft()	213
5.7.1.146 wifi_host_11k_cfg()	213
5.7.1.147 wifi_host_11k_neighbor_req()	213
5.7.1.148 wifi_host_11v_bss_trans_query()	214
5.7.1.149 wifi_clear_mgmt_ie()	214
5.7.1.150 wifi_set_sta_mac_filter()	214
5.7.1.151 wifi_set_auto_arp()	214
5.7.1.152 wifi_tcp_keep_alive()	214
5.7.1.153 wifi_cloud_keep_alive()	214
5.7.1.154 wifi_raw_packet_send()	215
5.7.1.155 wifi_raw_packet_recv()	215
5.7.1.156 wifi_set_11ax_tx_omi()	215
5.7.1.157 wifi_set_11ax_tol_time()	215
5.7.1.158 wifi_set_11ax_rutxpowerlimit()	215
5.7.1.159 wifi_set_11ax_rutxpowerlimit_legacy()	215
5.7.1.160 wifi_get_11ax_rutxpowerlimit_legacy()	216
5.7.1.161 wifi_set_11ax_cfg()	216
5.7.1.162 wifi_set_btwt_cfg()	216
5.7.1.163 wifi_set_twt_setup_cfg()	216
5.7.1.164 wifi_set_twt_tearardown_cfg()	217
5.7.1.165 wifi_get_twt_report()	217
5.7.1.166 wifi_twt_information()	217
5.7.1.167 wifi_set_clocksync_cfg()	219
5.7.1.168 wifi_get_tsf_info()	219
5.7.1.169 wifi_set_rf_test_mode()	219
5.7.1.170 wifi_unset_rf_test_mode()	219
5.7.1.171 wifi_set_rf_channel()	219
5.7.1.172 wifi_set_rf_radio_mode()	220
5.7.1.173 wifi_get_rf_channel()	220
5.7.1.174 wifi_get_rf_radio_mode()	220

5.7.1.175 wifi_set_rf_band()	220
5.7.1.176 wifi_get_rf_band()	220
5.7.1.177 wifi_set_rf_bandwidth()	220
5.7.1.178 wifi_get_rf_bandwidth()	220
5.7.1.179 wifi_get_rf_per()	221
5.7.1.180 wifi_set_rf_tx_cont_mode()	221
5.7.1.181 wifi_set_rf_tx_antenna()	221
5.7.1.182 wifi_get_rf_tx_antenna()	221
5.7.1.183 wifi_set_rf_rx_antenna()	221
5.7.1.184 wifi_get_rf_rx_antenna()	221
5.7.1.185 wifi_set_rf_tx_power()	222
5.7.1.186 wifi_cfg_rf_he_tb_tx()	222
5.7.1.187 wifi_rf_trigger_frame_cfg()	222
5.7.1.188 wifi_set_rf_tx_frame()	223
5.7.1.189 wifi_set_rf_otp_mac_addr()	223
5.7.1.190 wifi_get_rf_otp_mac_addr()	223
5.7.1.191 wifi_set_rf_otp_cal_data()	223
5.7.1.192 wifi_get_rf_otp_cal_data()	223
5.7.1.193 wifi_register_fw_dump_cb()	223
5.7.1.194 wifi_wmm_init()	224
5.7.1.195 wifi_wmm_get_pkt_prio()	224
5.7.1.196 wifi_wmm_get_packet_cnt()	224
5.7.1.197 wifi_handle_event_data_pause()	224
5.7.1.198 wifi_wmm_tx_stats_dump()	224
5.7.1.199 wifi_set_rssi_low_threshold()	225
5.7.1.200 wifi_show_os_mem_stat()	225
5.7.1.201 wifi_inject_frame()	225
5.7.1.202 wifi_supp_inject_frame()	225
5.7.1.203 wifi_is_wpa_supPLICANT_input()	226
5.7.1.204 wifi_wpa_supPLICANT_eapol_input()	226
5.7.1.205 wifi_get_sec_channel_offset()	226
5.7.1.206 wifi_nxp_scan_res_get()	226
5.7.1.207 wifi_nxp_survey_res_get()	226
5.7.1.208 wifi_nxp_set_default_scan_ies()	226
5.7.1.209 wifi_nxp_reset_scan_flag()	227
5.7.1.210 wifi_host_mbo_cfg()	227
5.7.1.211 wifi_mbo_preferch_cfg()	227
5.7.1.212 wifi_mbo_send_preferch_wmm()	227
5.7.1.213 wifi_csi_cfg()	227
5.7.1.214 register_csi_user_callback()	228
5.7.1.215 unregister_csi_user_callback()	228
5.7.1.216 csi_local_buff_init()	228

5.7.1.217 csi_save_data_to_local_buff()	228
5.7.1.218 csi_deliver_data_to_user()	228
5.7.1.219 wifi_send_mgmt_auth_request()	228
5.7.1.220 wifi_send_scan_cmd()	229
5.7.1.221 wifi_deauthenticate()	229
5.7.1.222 wifi_get_turbo_mode()	229
5.7.1.223 wifi_get_uap_turbo_mode()	229
5.7.1.224 wifi_set_turbo_mode()	229
5.7.1.225 wifi_set_uap_turbo_mode()	229
5.7.1.226 wifi_get_default_ht_capab()	230
5.7.1.227 wifi_get_default_vht_capab()	230
5.7.1.228 wifi_uap_client_assoc()	230
5.7.1.229 wifi_uap_client_deauth()	230
5.7.1.230 region_string_2_region_code()	230
5.7.1.231 wifi_set_indrst_cfg()	230
5.7.1.232 wifi_get_indrst_cfg()	231
5.7.1.233 wifi_test_independent_reset()	231
5.7.1.234 wifi_trigger_oob_indrst()	231
5.7.1.235 hostapd_connected_sta_list()	231
5.7.1.236 wifi_is_remain_on_channel()	231
5.7.2 Macro Documentation	231
5.7.2.1 CONFIG_STA_AUTO_DHCPV4	231
5.7.2.2 CONFIG_WIFI_STA_RECONNECT	232
5.7.2.3 CONFIG_WIFI_AUTO_POWER_SAVE	232
5.7.2.4 CONFIG_GTK_REKEY_OFFLOAD	232
5.7.2.5 CONFIG_TCP_ACK_ENH	232
5.7.2.6 CONFIG_FW_VDLL	232
5.7.2.7 WIFI_REG8	232
5.7.2.8 WIFI_REG16	232
5.7.2.9 WIFI_REG32	233
5.7.2.10 WIFI_WRITE_REG8	233
5.7.2.11 WIFI_WRITE_REG16	233
5.7.2.12 WIFI_WRITE_REG32	233
5.7.2.13 WIFI_COMMAND_RESPONSE_WAIT_MS	233
5.7.2.14 BANDWIDTH_20MHZ	233
5.7.2.15 BANDWIDTH_40MHZ	233
5.7.2.16 BANDWIDTH_80MHZ	234
5.7.2.17 MAX_NUM_CHANS_IN_NBOR_RPT	234
5.7.2.18 MBIT	234
5.7.2.19 WIFI_MGMT_DIASOC	234
5.7.2.20 WIFI_MGMT_AUTH	234
5.7.2.21 WIFI_MGMT_DEAUTH	234

5.7.2.22 WIFI_MGMT_ACTION	234
5.7.2.23 BEACON_REPORT_BUF_SIZE	235
5.7.2.24 MAX_NEIGHBOR_AP_LIMIT	235
5.7.3 Enumeration Type Documentation	235
5.7.3.1 anonymous enum	235
5.7.3.2 anonymous enum	235
5.7.3.3 IEEEtypes_ElementId_t	235
5.7.3.4 wifi_reg_t	236
5.7.3.5 wlan_rrm_beacon_reporting_detail	236
5.7.3.6 wlan_nlist_mode	236
5.7.4 Variable Documentation	237
5.7.4.1 wifi_tx_status	237
5.7.4.2 wifi_tx_block_cnt	237
5.7.4.3 wifi_rx_status	237
5.7.4.4 wifi_rx_block_cnt	237
5.7.4.5 g_bcn_nf_last	237
5.7.4.6 g_rssi	237
5.7.4.7 g_data_nf_last	237
5.7.4.8 g_data_snr_last	238
5.7.4.9 wifi_shutdown_enable	238
5.7.4.10 csi_event_cnt	238
5.7.4.11 csi_event_data_len	238
5.8 wifi_events.h File Reference	238
5.8.1 Enumeration Type Documentation	238
5.8.1.1 wifi_event	238
5.8.1.2 wifi_event_reason	240
5.8.1.3 wlan_bss_type	240
5.8.1.4 wlan_bss_role	240
5.8.1.5 wifi_wakeup_event_t	241
5.9 wifi_ping.h File Reference	241
5.9.1 Function Documentation	241
5.9.1.1 ping_cli_init()	241
5.9.1.2 ping_stats()	242
5.9.1.3 ping_cli_deinit()	242
5.9.2 Macro Documentation	242
5.9.2.1 ping_e	242
5.9.2.2 ping_w	242
5.9.2.3 PING_ID	242
5.9.2.4 PING_INTERVAL	243
5.9.2.5 PING_DEFAULT_TIMEOUT_SEC	243
5.9.2.6 PING_DEFAULT_COUNT	243
5.9.2.7 PING_DEFAULT_SIZE	243

5.9.2.8 PING_MAX_SIZE	243
5.9.2.9 PING_MAX_COUNT	243
5.10 wlan.h File Reference	243
5.10.1 Detailed Description	243
5.10.2 Usage	244
5.10.3 Function Documentation	244
5.10.3.1 is_valid_security()	244
5.10.3.2 is_ep_valid_security()	244
5.10.3.3 verify_scan_duration_value()	244
5.10.3.4 verify_scan_channel_value()	244
5.10.3.5 verify_split_scan_delay()	245
5.10.3.6 set_scan_params()	245
5.10.3.7 get_scan_params()	245
5.10.3.8 wlan_get_current_rssi()	245
5.10.3.9 wlan_get_current_nf()	245
5.10.3.10 wlan_init()	245
5.10.3.11 wlan_start()	246
5.10.3.12 wlan_stop()	246
5.10.3.13 wlan_deinit()	247
5.10.3.14 wlan_remove_all_network_profiles()	247
5.10.3.15 wlan_initialize_uap_network()	247
5.10.3.16 wlan_initialize_sta_network()	247
5.10.3.17 wlan_add_network()	248
5.10.3.18 wlan_remove_network()	249
5.10.3.19 wlan_connect()	249
5.10.3.20 wlan_connect_opt()	251
5.10.3.21 wlan_reassociate()	252
5.10.3.22 wlan_disconnect()	252
5.10.3.23 wlan_start_network()	253
5.10.3.24 wlan_stop_network()	253
5.10.3.25 wlan_get_mac_address()	254
5.10.3.26 wlan_get_mac_address_uap()	254
5.10.3.27 wlan_get_address()	255
5.10.3.28 wlan_get_uap_address()	255
5.10.3.29 wlan_get_uap_channel()	256
5.10.3.30 wlan_get_current_network()	256
5.10.3.31 wlan_get_current_network_ssid()	257
5.10.3.32 wlan_get_current_network_bssid()	257
5.10.3.33 wlan_get_current_uap_network()	258
5.10.3.34 wlan_get_current_uap_network_ssid()	258
5.10.3.35 is_uap_started()	258
5.10.3.36 is_sta_connected()	259

5.10.3.37 is_sta_ipv4_connected()	259
5.10.3.38 is_sta_ipv6_connected()	259
5.10.3.39 wlan_get_network()	260
5.10.3.40 wlan_get_network_byname()	260
5.10.3.41 wlan_get_network_count()	261
5.10.3.42 wlan_get_connection_state()	261
5.10.3.43 wlan_get_uap_connection_state()	262
5.10.3.44 wlan_scan()	262
5.10.3.45 wlan_scan_with_opt()	263
5.10.3.46 wlan_get_scan_result()	263
5.10.3.47 wlan_enable_low_pwr_mode()	264
5.10.3.48 wlan_set_ed_mac_mode()	264
5.10.3.49 wlan_set_uap_ed_mac_mode()	265
5.10.3.50 wlan_get_ed_mac_mode()	266
5.10.3.51 wlan_get_uap_ed_mac_mode()	266
5.10.3.52 wlan_set_cal_data()	266
5.10.3.53 wlan_set_mac_addr()	268
5.10.3.54 wlan_set_sta_mac_addr()	268
5.10.3.55 wlan_set_uap_mac_addr()	269
5.10.3.56 wlan_set_roaming()	269
5.10.3.57 wlan_get_roaming_status()	270
5.10.3.58 wlan_set_ieeeeps_cfg()	270
5.10.3.59 wlan_configure_listen_interval()	270
5.10.3.60 wlan_configure_delay_to_ps()	271
5.10.3.61 wlan_get_listen_interval()	271
5.10.3.62 wlan_get_delay_to_ps()	272
5.10.3.63 wlan_is_power_save_enabled()	272
5.10.3.64 wlan_configure_null_pkt_interval()	272
5.10.3.65 wlan_set_antcfg()	272
5.10.3.66 wlan_get_antcfg()	273
5.10.3.67 wlan_get_firmware_version_ext()	273
5.10.3.68 wlan_version_extended()	274
5.10.3.69 wlan_get_tsf()	274
5.10.3.70 wlan_ieeeeps_on()	274
5.10.3.71 wlan_ieeeeps_off()	275
5.10.3.72 wlan_deepsleeps_on()	275
5.10.3.73 wlan_deepsleeps_off()	275
5.10.3.74 wlan_tcp_keep_alive()	276
5.10.3.75 wlan_get_beacon_period()	276
5.10.3.76 wlan_get_dtim_period()	277
5.10.3.77 wlan_get_data_rate()	277
5.10.3.78 wlan_get_pmfcfg()	277

5.10.3.79 wlan_uap_get_pmfcfg()	278
5.10.3.80 wlan_set_packet_filters()	278
5.10.3.81 wlan_set_auto_arp()	281
5.10.3.82 wlan_wowlan_cfg_ptn_match()	281
5.10.3.83 wlan_set_ipv6_ns_offload()	281
5.10.3.84 wlan_get_current_bssid()	282
5.10.3.85 wlan_get_current_channel()	282
5.10.3.86 wlan_get_ps_mode()	282
5.10.3.87 wlan_wlcmgr_send_msg()	283
5.10.3.88 wlan_wfa_basic_cli_init()	283
5.10.3.89 wlan_wfa_basic_cli_deinit()	284
5.10.3.90 wlan_basic_cli_init()	284
5.10.3.91 wlan_basic_cli_deinit()	285
5.10.3.92 wlan_cli_init()	285
5.10.3.93 wlan_cli_deinit()	286
5.10.3.94 wlan_enhanced_cli_init()	286
5.10.3.95 wlan_enhanced_cli_deinit()	287
5.10.3.96 wlan_test_mode_cli_init()	287
5.10.3.97 wlan_test_mode_cli_deinit()	288
5.10.3.98 wlan_get_uap_supported_max_clients()	288
5.10.3.99 wlan_get_uap_max_clients()	288
5.10.3.100 wlan_set_uap_max_clients()	289
5.10.3.101 wlan_set_htcapinfo()	289
5.10.3.102 wlan_set_httxcfg()	290
5.10.3.103 wlan_set_txratecfg()	291
5.10.3.104 wlan_get_txratecfg()	293
5.10.3.105 wlan_get_sta_tx_power()	293
5.10.3.106 wlan_set_sta_tx_power()	294
5.10.3.107 wlan_set_wwsm_txpwrlimit()	294
5.10.3.108 wlan_get_wlan_region_code()	294
5.10.3.109 wlan_get_mgmt_ie()	294
5.10.3.110 wlan_set_mgmt_ie()	295
5.10.3.111 wlan_get_ext_coex_stats()	295
5.10.3.112 wlan_set_ext_coex_config()	296
5.10.3.113 wlan_clear_mgmt_ie()	296
5.10.3.114 wlan_get_11d_enable_status()	297
5.10.3.115 wlan_get_current_signal_strength()	297
5.10.3.116 wlan_get_average_signal_strength()	297
5.10.3.117 wlan_remain_on_channel()	298
5.10.3.118 wlan_get_otp_user_data()	298
5.10.3.119 wlan_get_cal_data()	299
5.10.3.120 wlan_set_region_power_cfg()	299

5.10.3.121 wlan_set_chanlist_and_txpwrlimit()	300
5.10.3.122 wlan_set_chanlist()	300
5.10.3.123 wlan_get_chanlist()	300
5.10.3.124 wlan_set_txpwrlimit()	301
5.10.3.125 wlan_get_txpwrlimit()	301
5.10.3.126 wlan_auto_reconnect_enable()	302
5.10.3.127 wlan_auto_reconnect_disable()	303
5.10.3.128 wlan_get_auto_reconnect_config()	303
5.10.3.129 wlan_set_reassoc_control()	303
5.10.3.130 wlan_uap_set_beacon_period()	304
5.10.3.131 wlan_uap_set_bandwidth()	304
5.10.3.132 wlan_uap_get_bandwidth()	305
5.10.3.133 wlan_uap_set_hidden_ssid()	306
5.10.3.134 wlan_uap_ctrl_deauth()	306
5.10.3.135 wlan_uap_set_ecsa()	307
5.10.3.136 wlan_uap_set_htcapinfo()	307
5.10.3.137 wlan_uap_set_htxcfg()	308
5.10.3.138 wlan_sta_ampdu_tx_enable()	308
5.10.3.139 wlan_sta_ampdu_tx_disable()	309
5.10.3.140 wlan_sta_ampdu_rx_enable()	309
5.10.3.141 wlan_sta_ampdu_rx_disable()	309
5.10.3.142 wlan_uap_ampdu_tx_enable()	309
5.10.3.143 wlan_uap_ampdu_tx_disable()	309
5.10.3.144 wlan_uap_ampdu_rx_enable()	310
5.10.3.145 wlan_uap_ampdu_rx_disable()	310
5.10.3.146 wlan_uap_set_scan_chan_list()	310
5.10.3.147 wlan_set_rts()	310
5.10.3.148 wlan_set_uap_rts()	311
5.10.3.149 wlan_set_frag()	311
5.10.3.150 wlan_set_uap_frag()	311
5.10.3.151 wlan_set_sta_mac_filter()	312
5.10.3.152 print_mac()	312
5.10.3.153 wlan_set_rf_test_mode()	312
5.10.3.154 wlan_unset_rf_test_mode()	313
5.10.3.155 wlan_set_rf_channel()	313
5.10.3.156 wlan_set_rf_radio_mode()	313
5.10.3.157 wlan_get_rf_channel()	314
5.10.3.158 wlan_get_rf_radio_mode()	314
5.10.3.159 wlan_set_rf_band()	315
5.10.3.160 wlan_get_rf_band()	315
5.10.3.161 wlan_set_rf_bandwidth()	315
5.10.3.162 wlan_get_rf_bandwidth()	316

5.10.3.163 wlan_get_rf_per()	316
5.10.3.164 wlan_set_rf_tx_cont_mode()	317
5.10.3.165 wlan_cfg_rf_he_tb_tx()	317
5.10.3.166 wlan_rf_trigger_frame_cfg()	318
5.10.3.167 wlan_set_rf_tx_antenna()	320
5.10.3.168 wlan_get_rf_tx_antenna()	320
5.10.3.169 wlan_set_rf_rx_antenna()	320
5.10.3.170 wlan_get_rf_rx_antenna()	321
5.10.3.171 wlan_set_rf_tx_power()	321
5.10.3.172 wlan_set_rf_tx_frame()	322
5.10.3.173 wlan_set_rf_otp_mac_addr()	323
5.10.3.174 wlan_get_rf_otp_mac_addr()	323
5.10.3.175 wlan_set_rf_otp_cal_data()	323
5.10.3.176 wlan_get_rf_otp_cal_data()	324
5.10.3.177 wlan_register_fw_dump_cb()	324
5.10.3.178 wlan_set_crypto_RC4_encrypt()	325
5.10.3.179 wlan_set_crypto_RC4_decrypt()	325
5.10.3.180 wlan_set_crypto_AES_ECB_encrypt()	326
5.10.3.181 wlan_set_crypto_AES_ECB_decrypt()	327
5.10.3.182 wlan_set_crypto_AES_WRAP_encrypt()	328
5.10.3.183 wlan_set_crypto_AES_WRAP_decrypt()	328
5.10.3.184 wlan_set_crypto_AES_CCMP_encrypt()	329
5.10.3.185 wlan_set_crypto_AES_CCMP_decrypt()	330
5.10.3.186 wlan_set_crypto_AES_GCMP_encrypt()	330
5.10.3.187 wlan_set_crypto_AES_GCMP_decrypt()	331
5.10.3.188 wlan_send_hostcmd()	332
5.10.3.189 wlan_send_debug_htc()	333
5.10.3.190 wlan_enable_disable_htc()	333
5.10.3.191 wlan_set_11ax_tx_omi()	334
5.10.3.192 wlan_set_11ax_tol_time()	334
5.10.3.193 wlan_set_11ax_rutxpowerlimit()	335
5.10.3.194 wlan_set_11ax_rutxpowerlimit_legacy()	335
5.10.3.195 wlan_get_11ax_rutxpowerlimit_legacy()	335
5.10.3.196 wlan_set_11ax_cfg()	336
5.10.3.197 wlan_get_11ax_cfg()	336
5.10.3.198 wlan_set_btwt_cfg()	336
5.10.3.199 wlan_get_btwt_cfg()	337
5.10.3.200 wlan_set_twt_setup_cfg()	337
5.10.3.201 wlan_get_twt_setup_cfg()	337
5.10.3.202 wlan_set_twt_teardown_cfg()	338
5.10.3.203 wlan_get_twt_teardown_cfg()	338
5.10.3.204 wlan_get_twt_report()	338

5.10.3.205 wlan_twt_information()	339
5.10.3.206 wlan_set_clocksync_cfg()	339
5.10.3.207 wlan_get_tsf_info()	339
5.10.3.208 wlan_show_os_mem_stat()	340
5.10.3.209 wlan_ft_roam()	340
5.10.3.210 wlan_rx_mgmt_indication()	340
5.10.3.211 wlan_wmm_tx_stats_dump()	341
5.10.3.212 wlan_set_scan_channel_gap()	341
5.10.3.213 wlan_host_11k_cfg()	341
5.10.3.214 wlan_get_host_11k_status()	342
5.10.3.215 wlan_host_11k_neighbor_req()	342
5.10.3.216 wlan_host_11v_bss_trans_query()	342
5.10.3.217 wlan_set_okc()	342
5.10.3.218 wlan_pmksa_list()	343
5.10.3.219 wlan_pmksa_flush()	343
5.10.3.220 wlan_set_scan_interval()	343
5.10.3.221 wlan_tx_ampdu_prot_mode()	344
5.10.3.222 wlan_mef_set_auto_arp()	344
5.10.3.223 wlan_mef_set_auto_ping()	345
5.10.3.224 wlan_config_mef()	345
5.10.3.225 wlan_set_ipv6_ns_mef()	345
5.10.3.226 wlan_csi_cfg()	346
5.10.3.227 wlan_register_csi_user_callback()	346
5.10.3.228 wlan_unregister_csi_user_callback()	347
5.10.3.229 wlan_set_rssi_low_threshold()	347
5.10.3.230 wlan_wps_generate_pin()	347
5.10.3.231 wlan_start_wps_pin()	347
5.10.3.232 wlan_start_wps_pbc()	348
5.10.3.233 wlan_wps_cancel()	348
5.10.3.234 wlan_start_ap_wps_pin()	348
5.10.3.235 wlan_start_ap_wps_pbc()	349
5.10.3.236 wlan_wps_ap_cancel()	349
5.10.3.237 wlan_set_entp_cert_files()	349
5.10.3.238 wlan_get_entp_cert_files()	350
5.10.3.239 wlan_free_entp_cert_files()	350
5.10.3.240 wlan_check_11n_capa()	350
5.10.3.241 wlan_check_11ac_capa()	351
5.10.3.242 wlan_check_11ax_capa()	351
5.10.3.243 wlan_get_signal_info()	351
5.10.3.244 wlan_set_rg_power_cfg()	353
5.10.3.245 wlan_get_turbo_mode()	353
5.10.3.246 wlan_get_uap_turbo_mode()	353

5.10.3.247 wlan_set_turbo_mode()	355
5.10.3.248 wlan_set_uap_turbo_mode()	355
5.10.3.249 wlan_set_ps_cfg()	355
5.10.3.250 wlan_save_cloud_keep_alive_params()	356
5.10.3.251 wlan_cloud_keep_alive_enabled()	356
5.10.3.252 wlan_start_cloud_keep_alive()	357
5.10.3.253 wlan_stop_cloud_keep_alive()	357
5.10.3.254 wlan_set_country_code()	357
5.10.3.255 wlan_set_country_ie_ignore()	358
5.10.3.256 wlan_set_region_code()	358
5.10.3.257 wlan_get_region_code()	359
5.10.3.258 wlan_set_11d_state()	359
5.10.3.259 wlan_dpp_configurator_add()	359
5.10.3.260 wlan_dpp_configurator_params()	360
5.10.3.261 wlan_dpp_mud_url()	360
5.10.3.262 wlan_dpp_bootstrap_gen()	361
5.10.3.263 wlan_dpp_bootstrap_get_uri()	361
5.10.3.264 wlan_dpp_qr_code()	362
5.10.3.265 wlan_dpp_auth_init()	362
5.10.3.266 wlan_dpp_listen()	362
5.10.3.267 wlan_dpp_stop_listen()	363
5.10.3.268 wlan_dpp_pkex_add()	363
5.10.3.269 wlan_dpp_chirp()	364
5.10.3.270 wlan_dpp_reconfig()	364
5.10.3.271 wlan_dpp_configurator_sign()	364
5.10.3.272 wlan_host_set_sta_mac_filter()	365
5.10.3.273 wlan_set_indrst_cfg()	365
5.10.3.274 wlan_get_indrst_cfg()	365
5.10.3.275 wlan_independent_reset()	366
5.10.3.276 wlan_set_network_ip_byname()	366
5.10.3.277 wlan_string_dup()	366
5.10.3.278 wlan_get_board_type()	366
5.10.3.279 wlan_uap_disconnect_sta()	366
5.10.4 Macro Documentation	367
5.10.4.1 WLAN_DRV_VERSION	367
5.10.4.2 ARG_UNUSED	367
5.10.4.3 CONFIG_WLAN_KNOWN_NETWORKS	367
5.10.4.4 wlcm_e	367
5.10.4.5 wlcm_w	367
5.10.4.6 wlcm_d	368
5.10.4.7 ACTION_GET	368
5.10.4.8 ACTION_SET	368

5.10.4.9 IEEEtypes_SSID_SIZE	368
5.10.4.10 IEEEtypes_ADDRESS_SIZE	368
5.10.4.11 WLAN_REASON_CODE_PREV_AUTH_NOT_VALID	368
5.10.4.12 WLAN_RESCAN_LIMIT	368
5.10.4.13 WLAN_11D_SCAN_LIMIT	369
5.10.4.14 WLAN_RECONNECT_LIMIT	369
5.10.4.15 WLAN_NETWORK_NAME_MIN_LENGTH	369
5.10.4.16 WLAN_NETWORK_NAME_MAX_LENGTH	369
5.10.4.17 WLAN_PSK_MIN_LENGTH	369
5.10.4.18 WLAN_PSK_MAX_LENGTH	369
5.10.4.19 WLAN_PASSWORD_MIN_LENGTH	369
5.10.4.20 WLAN_PASSWORD_MAX_LENGTH	370
5.10.4.21 IDENTITY_MAX_LENGTH	370
5.10.4.22 PASSWORD_MAX_LENGTH	370
5.10.4.23 MAX_USERS	370
5.10.4.24 PAC_OPAQUE_ENCR_KEY_MAX_LENGTH	370
5.10.4.25 A_ID_MAX_LENGTH	370
5.10.4.26 HASH_MAX_LENGTH	370
5.10.4.27 DOMAIN_MATCH_MAX_LENGTH	371
5.10.4.28 WLAN_MAX_KNOWN_NETWORKS	371
5.10.4.29 WLAN_PMK_LENGTH	371
5.10.4.30 WLAN_MAX_STA_FILTER_NUM	371
5.10.4.31 WLAN_MAC_ADDR_LENGTH	371
5.10.4.32 WLAN_ERROR_NONE	371
5.10.4.33 WLAN_ERROR_PARAM	371
5.10.4.34 WLAN_ERROR_NOMEM	371
5.10.4.35 WLAN_ERROR_STATE	372
5.10.4.36 WLAN_ERROR_ACTION	372
5.10.4.37 WLAN_ERROR_PS_ACTION	372
5.10.4.38 WLAN_ERROR_NOT_SUPPORTED	372
5.10.4.39 HOST_WAKEUP_GPIO_PIN	372
5.10.4.40 CARD_WAKEUP_GPIO_PIN	372
5.10.4.41 WLAN_MGMT_DIASOC	372
5.10.4.42 WLAN_MGMT_AUTH	372
5.10.4.43 WLAN_MGMT_DEAUTH	373
5.10.4.44 WLAN_MGMT_ACTION	373
5.10.4.45 WLAN_KEY_MGMT_IEEE8021X	373
5.10.4.46 WLAN_KEY_MGMT_PSK	373
5.10.4.47 WLAN_KEY_MGMT_NONE	373
5.10.4.48 WLAN_KEY_MGMT_IEEE8021X_NO_WPA	373
5.10.4.49 WLAN_KEY_MGMT_WPA_NONE	373
5.10.4.50 WLAN_KEY_MGMT_FT_IEEE8021X	373

5.10.4.51 WLAN_KEY_MGMT_FT_PSK	374
5.10.4.52 WLAN_KEY_MGMT_IEEE8021X_SHA256	374
5.10.4.53 WLAN_KEY_MGMT_PSK_SHA256	374
5.10.4.54 WLAN_KEY_MGMT_WPS	374
5.10.4.55 WLAN_KEY_MGMT_SAE	374
5.10.4.56 WLAN_KEY_MGMT_FT_SAE	374
5.10.4.57 WLAN_KEY_MGMT_WAPI_PSK	374
5.10.4.58 WLAN_KEY_MGMT_WAPI_CERT	374
5.10.4.59 WLAN_KEY_MGMT_CCKM	375
5.10.4.60 WLAN_KEY_MGMT_OSEN	375
5.10.4.61 WLAN_KEY_MGMT_IEEE8021X_SUITE_B	375
5.10.4.62 WLAN_KEY_MGMT_IEEE8021X_SUITE_B_192	375
5.10.4.63 WLAN_KEY_MGMT_FILS_SHA256	375
5.10.4.64 WLAN_KEY_MGMT_FILS_SHA384	375
5.10.4.65 WLAN_KEY_MGMT_FT_FILS_SHA256	375
5.10.4.66 WLAN_KEY_MGMT_FT_FILS_SHA384	375
5.10.4.67 WLAN_KEY_MGMT_OWE	376
5.10.4.68 WLAN_KEY_MGMT_DPP	376
5.10.4.69 WLAN_KEY_MGMT_FT_IEEE8021X_SHA384	376
5.10.4.70 WLAN_KEY_MGMT_PASN	376
5.10.4.71 WLAN_KEY_MGMT_SAE_EXT_KEY	376
5.10.4.72 WLAN_KEY_MGMT_FT	376
5.10.4.73 WLAN_CIPHER_NONE	376
5.10.4.74 WLAN_CIPHER_WEP40	377
5.10.4.75 WLAN_CIPHER_WEP104	377
5.10.4.76 WLAN_CIPHER_TKIP	377
5.10.4.77 WLAN_CIPHER_CCMP	377
5.10.4.78 WLAN_CIPHER_AES_128_CMAC	377
5.10.4.79 WLAN_CIPHER_GCMP	377
5.10.4.80 WLAN_CIPHER_SMS4	377
5.10.4.81 WLAN_CIPHER_GCMP_256	377
5.10.4.82 WLAN_CIPHER_CCMP_256	378
5.10.4.83 WLAN_CIPHER_BIP_GMAC_128	378
5.10.4.84 WLAN_CIPHER_BIP_GMAC_256	378
5.10.4.85 WLAN_CIPHER_BIP_CMAC_256	378
5.10.4.86 WLAN_CIPHER_GTK_NOT_USED	378
5.10.4.87 NUM_CHAN_BAND_ENUMS	378
5.10.4.88 DFS_REC_HDR_LEN	378
5.10.4.89 DFS_REC_HDR_NUM	378
5.10.4.90 BIN_COUNTER_LEN	379
5.10.4.91 MAX_CHANNEL_LIST	379
5.10.4.92 TX_AMPDU_RTS_CTS	379

5.10.4.93 TX_AMPDU_CTS_2_SELF	379
5.10.4.94 TX_AMPDU_DISABLE_PROTECTION	379
5.10.4.95 TX_AMPDU_DYNAMIC_RTS_CTS	379
5.10.4.96 EU_CRYPTOP_DATA_MAX_LENGTH	379
5.10.4.97 EU_CRYPTOP_KEY_MAX_LENGTH	379
5.10.4.98 EU_CRYPTOP_KEYIV_MAX_LENGTH	380
5.10.4.99 EU_CRYPTOP_NONCE_MAX_LENGTH	380
5.10.4.100 EU_CRYPTOP_AAD_MAX_LENGTH	380
5.10.4.101 FILE_TYPE_NONE	380
5.10.4.102 FILE_TYPE_ENTP_CA_CERT	380
5.10.4.103 FILE_TYPE_ENTP_CLIENT_CERT	380
5.10.4.104 FILE_TYPE_ENTP_CLIENT_KEY	380
5.10.4.105 FILE_TYPE_ENTP_CA_CERT2	380
5.10.4.106 FILE_TYPE_ENTP_CLIENT_CERT2	381
5.10.4.107 FILE_TYPE_ENTP_CLIENT_KEY2	381
5.10.4.108 FILE_TYPE_ENTP_SERVER_CERT	381
5.10.4.109 FILE_TYPE_ENTP_SERVER_KEY	381
5.10.4.110 FILE_TYPE_ENTP_DH_PARAMS	381
5.10.5 Typedef Documentation	381
5.10.5.1 wlan_scan_channel_list_t	381
5.10.5.2 wlan_scan_params_v2_t	381
5.10.5.3 wlan_cal_data_t	382
5.10.5.4 wlan_auto_reconnect_config_t	382
5.10.5.5 wlanflt_cfg_t	382
5.10.5.6 wlan_wowlan_ptn_cfg_t	382
5.10.5.7 wlan_tcp_keep_alive_t	382
5.10.5.8 wlan_cloud_keep_alive_t	382
5.10.5.9 wlan_ds_rate	382
5.10.5.10 wlan_ed_mac_ctrl_t	382
5.10.5.11 wlan_bandcfg_t	383
5.10.5.12 wlan_cw_mode_ctrl_t	383
5.10.5.13 wlan_chanlist_t	383
5.10.5.14 wlan_txpwrlimit_t	383
5.10.5.15 wlan_ext_coex_stats_t	383
5.10.5.16 wlan_ext_coex_config_t	383
5.10.5.17 wlan_rutxpwrlimit_t	383
5.10.5.18 wlan_11ax_config_t	383
5.10.5.19 wlan_twt_setup_config_t	384
5.10.5.20 wlan_twt_tearardown_config_t	384
5.10.5.21 wlan_btwt_config_t	384
5.10.5.22 wlan_twt_report_t	384
5.10.5.23 wlan_twt_information_t	384

5.10.5.24 wlan_clock_sync_gpio_tsf_t	384
5.10.5.25 wlan_tsf_info_t	384
5.10.5.26 wlan_mgmt_frame_t	384
5.10.5.27 wlan_csi_config_params_t	385
5.10.5.28 wlan_indrst_cfg_t	385
5.10.5.29 wlan_txrate_setting	385
5.10.5.30 wlan_rssi_info_t	385
5.10.5.31 wlan_uap_client_disassoc_t	385
5.10.6 Enumeration Type Documentation	385
5.10.6.1 IEEEtypes_Bss_t	385
5.10.6.2 wm_wlan_errno	386
5.10.6.3 wlan_event_reason	386
5.10.6.4 wlan_wakeup_event_t	387
5.10.6.5 wlan_connection_state	388
5.10.6.6 wlan_ps_mode	388
5.10.6.7 wlan_ps_state	389
5.10.6.8 ENH_PS_MODES	389
5.10.6.9 Host_Sleep_Action	389
5.10.6.10 wlan_csi_opt	390
5.10.6.11 wlan_monitor_opt	390
5.10.6.12 ChanBand_e	390
5.10.6.13 ChanWidth_e	390
5.10.6.14 Chan2Offset_e	391
5.10.6.15 ScanMode_e	391
5.10.6.16 wlan_security_type	391
5.10.6.17 eap_tls_cipher_type	393
5.10.6.18 address_types	393
5.10.6.19 wlan_mef_type	393
5.11 wlan_11d.h File Reference	393
5.11.1 Function Documentation	394
5.11.1.1 wlan_enable_11d()	394
5.11.1.2 wlan_enable_uap_11d()	394
5.12 wlan_tests.h File Reference	395
5.12.1 Function Documentation	395
5.12.1.1 test_wlan_cfg_process()	395
5.12.1.2 print_txpwrlimit()	395
5.12.2 Enumeration Type Documentation	395
5.12.2.1 anonymous enum	395
5.13 wm_net.h File Reference	396
5.13.1 Detailed Description	396
5.13.2 Function Documentation	396
5.13.2.1 net_dhcp_hostname_set()	396

5.13.2.2 net_stop_dhcp_timer()	396
5.13.2.3 net_socket_blocking()	396
5.13.2.4 net_get_sock_error()	397
5.13.2.5 net_inet_aton()	397
5.13.2.6 net_wlan_set_mac_address()	398
5.13.2.7 net_stack_buffer_skip()	398
5.13.2.8 net_inet_ntoa()	398
5.13.2.9 net_sock_to_interface()	399
5.13.2.10 net_wlan_init()	399
5.13.2.11 net_wlan_deinit()	399
5.13.2.12 net_get_sta_interface()	400
5.13.2.13 net_get_uap_interface()	400
5.13.2.14 net_alloc_client_data_id()	400
5.13.2.15 net_get_sta_handle()	400
5.13.2.16 net_get_uap_handle()	401
5.13.2.17 net_interface_up()	401
5.13.2.18 net_interface_down()	401
5.13.2.19 net_interface_dhcp_stop()	402
5.13.2.20 net_interface_dhcp_cleanup()	402
5.13.2.21 net_configure_address()	402
5.13.2.22 net_configure_dns()	403
5.13.2.23 net_get_if_addr()	403
5.13.2.24 net_get_if_ipv6_addr()	404
5.13.2.25 net_get_if_ipv6_pref_addr()	404
5.13.2.26 ipv6_addr_state_to_desc()	404
5.13.2.27 ipv6_addr_addr_to_desc()	405
5.13.2.28 ipv6_addr_type_to_desc()	405
5.13.2.29 net_get_if_name()	406
5.13.2.30 net_get_if_ip_addr()	406
5.13.2.31 net_get_if_ip_mask()	406
5.13.2.32 net_ipv4stack_init()	408
5.13.2.33 net_stat()	408
5.13.3 Macro Documentation	408
5.13.3.1 NET_SUCCESS	408
5.13.3.2 NET_ERROR	409
5.13.3.3 NET_ENOBUFS	409
5.13.3.4 NET_BLOCKING_OFF	409
5.13.3.5 NET_BLOCKING_ON	409
5.13.3.6 net_socket	409
5.13.3.7 net_select	409
5.13.3.8 net_bind	410
5.13.3.9 net_listen	410

5.13.3.10 net_close	410
5.13.3.11 net_accept	410
5.13.3.12 net_shutdown	410
5.13.3.13 net_connect	410
5.13.3.14 net_read	411
5.13.3.15 net_write	411
5.13.3.16 net_get_mlan_handle	411
5.13.4 Enumeration Type Documentation	411
5.13.4.1 net_address_types	411
5.14 wm_utils.h File Reference	411
5.14.1 Detailed Description	411
5.14.2 Function Documentation	412
5.14.2.1 wmpanic()	412
5.14.2.2 wm_hex2bin()	412
5.14.2.3 wm_bin2hex()	412
5.14.2.4 random_register_handler()	413
5.14.2.5 random_unregister_handler()	413
5.14.2.6 random_register_seed_handler()	414
5.14.2.7 random_unregister_seed_handler()	414
5.14.2.8 random_initialize_seed()	414
5.14.2.9 sample_initialise_random_seed()	415
5.14.2.10 get_random_sequence()	415
5.14.2.11 wm_frac_part_of()	415
5.14.2.12 strdup()	416
5.14.2.13 soft_crc32()	416
5.14.2.14 wm_strtof()	417
5.14.2.15 fill_sequential_pattern()	417
5.14.2.16 verify_sequential_pattern()	417
5.14.3 Macro Documentation	418
5.14.3.1 ffs	418
5.14.3.2 WARN_UNUSED_RET	418
5.14.3.3 PACK_START	418
5.14.3.4 PACK_END	418
5.14.3.5 NORETURN	418
5.14.3.6 __WM_ALIGN__	418
5.14.3.7 WM_MASK	419
5.14.3.8 dump_hex	419
5.14.3.9 dump_hex_ascii	419
5.14.3.10 dump_ascii	419
5.14.3.11 print_ascii	420
5.14.3.12 dump_json	420
5.14.3.13 wm_int_part_of	420

5.14.4 Typedef Documentation	420
5.14.4.1 random_hdlr_t	420

Confidential

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:

_Cipher_t	9
_SecurityMode_t	12
BandConfig_t	15
ChanBandInfo_t	15
cli_command	16
csi_local_buff_statu	17
Event_Radar_Detected_Info	18
ipv4_config	20
ipv6_config	21
net_ip_config	22
net_ipv4_config	23
net_ipv6_config	24
osa_rw_lock_t	25
test_cfg_param_t	26
test_cfg_table_t	27
tx_ampdu_prot_mode_para	28
txrate_setting	29
wifi_11ax_config_t	31
wifi_antcfg_t	32
wifi_auto_reconnect_config_t	33
wifi_bandcfg_t	34
wifi_btwt_config_t	35
wifi_cal_data_t	36
wifi_chan_info_t	37
wifi_chan_list_param_set_t	38
wifi_chan_scan_param_set_t	39
wifi_chanlist_t	39
wifi_channel_desc_t	40
wifi_clock_sync_gpio_ts_f_t	41
wifi_cloud_keep_alive_t	42
wifi_csi_config_params_t	45
wifi_csi_filter_t	47
wifi_cw_mode_ctrl_t	48
wifi_data_rate_t	49
wifi_ds_rate	50

wifi_ecsa_info	51
wifi_ed_mac_ctrl_t	52
wifi_ext_coex_config_t	53
wifi_ext_coex_stats_t	55
wififlt_cfg_t	56
wifi_frame_t	57
wifi_fw_version_ext_t	57
wifi_fw_version_t	58
wifi_indrst_cfg_t	59
wifi_mac_addr_t	59
wifi_mef_entry_t	60
wifi_mef_filter_t	61
wifi_message	63
wifi_mfg_cmd_generic_cfg_t	64
wifi_mfg_cmd_he_tb_tx_t	65
wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t	67
wifi_mfg_cmd_otp_cal_data_rd_wr_t	69
wifi_mfg_cmd_otp_mac_addr_rd_wr_t	70
wifi_mfg_cmd_tx_cont_t	71
wifi_mfg_cmd_tx_frame_t	74
wifi_mgmt_frame_t	78
wifi_nat_keep_alive_t	80
wifi_os_mem_info	81
wifi_pmf_params_t	82
wifi_rate_cfg_t	82
wifi_remain_on_channel_t	84
wifi_rf_channel_t	85
wifi_rssi_info_t	86
wifi_rupwrlimit_config_t	88
wifi_rutxpwrlimit_t	89
wifi_scan_chan_list_t	90
wifi_scan_channel_list_t	90
wifi_scan_params_t	91
wifi_scan_params_v2_t	92
wifi_scan_result2	94
wifi_sta_info_t	100
wifi_sta_list_t	101
wifi_sub_band_set_t	101
wifi_tbt_offset_t	102
wifi_tcp_keep_alive_t	103
wifi_tsf_info_t	105
wifi_twt_information_t	106
wifi_twt_report_t	107
wifi_twt_setup_config_t	108
wifi_twt_tearardown_config_t	110
wifi_tx_power_t	110
wifi_txpwrlimit_config_t	111
wifi_txpwrlimit_entry_t	112
wifi_txpwrlimit_t	113
wifi_uap_client_disassoc_t	114
wifi_wowlan_pattern_t	115
wifi_wowlan_ptn_cfg_t	115
wlan_cipher	116
wlan_ieeeeps_config	119
wlan_ip_config	120
wlan_message	121
wlan_network	122
wlan_network_security	128

wlan_nlist_report_param	138
wlan_rrm_beacon_report_data	139
wlan_rrm_neighbor_ap_t	141
wlan_rrm_neighbor_report_t	142
wlan_rrm_scan_cb_param	143
wlan_scan_result	144
wnm_sleep_result_t	150

Confidential

Confidential

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

cli.h	CLI module	151
dhcp-server.h	DHCP server	156
iperf.h	This file provides the support for network utility iperf	160
osa.h	161
wifi-decl.h	Wifi structure declarations	171
wifi.h	This file contains interface to wifi driver	182
wifi_events.h	Wi-Fi events	238
wifi_ping.h	This file provides the support for network utility ping	241
wlan.h	WLAN Connection Manager	243
wlan_11d.h	WLAN module 11d API	393
wlan_tests.h	WLAN Connection Manager Tests	395
wm_net.h	Network Abstraction Layer	396
wm_utils.h	Utility functions	411

Confidential

Chapter 4

Data Structure Documentation

4.1 `_Cipher_t` Struct Reference

Data Fields

- `uint16_t none`: 1
- `uint16_t wep40`: 1
- `uint16_t wep104`: 1
- `uint16_t tkip`: 1
- `uint16_t ccmp`: 1
- `uint16_t aes_128_cmac`: 1
- `uint16_t gcmp`: 1
- `uint16_t sms4`: 1
- `uint16_t gcmp_256`: 1
- `uint16_t ccmp_256`: 1
- `uint16_t rsvd`: 1
- `uint16_t bip_gmac_128`: 1
- `uint16_t bip_gmac_256`: 1
- `uint16_t bip_cmac_256`: 1
- `uint16_t gtk_not_used`: 1
- `uint16_t rsvd2`: 2

4.1.1 Field Documentation

4.1.1.1 `none`

`uint16_t _Cipher_t::none`

1 bit value can be set for none

4.1.1.2 wep40

`uint16_t _Cipher_t::wep40`

1 bit value can be set for wep40

4.1.1.3 wep104

`uint16_t _Cipher_t::wep104`

1 bit value can be set for wep104

4.1.1.4 tkip

`uint16_t _Cipher_t::tkip`

1 bit value can be set for tkip

4.1.1.5 ccmp

`uint16_t _Cipher_t::ccmp`

1 bit value can be set for ccmp

4.1.1.6 aes_128_cmac

`uint16_t _Cipher_t::aes_128_cmac`

1 bit value can be set for aes 128 cmac

4.1.1.7 gcmp

`uint16_t _Cipher_t::gcmp`

1 bit value can be set for gcmp

4.1.1.8 sms4

`uint16_t _Cipher_t::sms4`

1 bit value can be set for sms4

4.1.1.9 gcmp_256

`uint16_t _Cipher_t::gcmp_256`

1 bit value can be set for gcmp 256

4.1.1.10 ccmp_256

uint16_t _Cipher_t::ccmp_256

1 bit value can be set for ccmp 256

4.1.1.11 rsvd

uint16_t _Cipher_t::rsvd

1 bit is reserved

4.1.1.12 bip_gmac_128

uint16_t _Cipher_t::bip_gmac_128

1 bit value can be set for bip gmac 128

4.1.1.13 bip_gmac_256

uint16_t _Cipher_t::bip_gmac_256

1 bit value can be set for bip gmac 256

4.1.1.14 bip_cmac_256

uint16_t _Cipher_t::bip_cmac_256

1 bit value can be set for bip cmac 256

4.1.1.15 gtk_not_used

uint16_t _Cipher_t::gtk_not_used

1 bit value can be set for gtk not used

4.1.1.16 rsvd2

uint16_t _Cipher_t::rsvd2

4 bits are reserved

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

- [wifi-decl.h](#)

4.2 `_SecurityMode_t` Struct Reference

Data Fields

- `uint32_t noRsn`: 1
- `uint32_t wepStatic`: 1
- `uint32_t wepDynamic`: 1
- `uint32_t wpa`: 1
- `uint32_t wpaNone`: 1
- `uint32_t wpa2`: 1
- `uint32_t wpa2_sha256`: 1
- `uint32_t owe`: 1
- `uint32_t wpa3_sae`: 1
- `uint32_t wpa2_entp`: 1
- `uint32_t wpa2_entp_sha256`: 1
- `uint32_t ft_1x`: 1
- `uint32_t ft_1x_sha384`: 1
- `uint32_t ft_psk`: 1
- `uint32_t ft_sae`: 1
- `uint32_t wpa3_1x_sha256`: 1
- `uint32_t wpa3_1x_sha384`: 1
- `uint32_t rsvd`: 16

4.2.1 Field Documentation

4.2.1.1 `noRsn`

`uint32_t _SecurityMode_t::noRsn`

No security

4.2.1.2 `wepStatic`

`uint32_t _SecurityMode_t::wepStatic`

WEP static

4.2.1.3 `wepDynamic`

`uint32_t _SecurityMode_t::wepDynamic`

WEP dynamic

4.2.1.4 wpa

uint32_t _SecurityMode_t::wpa

WPA

4.2.1.5 wpaNone

uint32_t _SecurityMode_t::wpaNone

WPA none

4.2.1.6 wpa2

uint32_t _SecurityMode_t::wpa2

WPA 2

4.2.1.7 wpa2_sha256

uint32_t _SecurityMode_t::wpa2_sha256

WPA 2 sha256

4.2.1.8 owe

uint32_t _SecurityMode_t::owe

OWE

4.2.1.9 wpa3_sae

uint32_t _SecurityMode_t::wpa3_sae

WPA3 SAE

4.2.1.10 wpa2_entp

uint32_t _SecurityMode_t::wpa2_entp

802.1x

4.2.1.11 wpa2_entp_sha256

uint32_t _SecurityMode_t::wpa2_entp_sha256

802.1x sha256

4.2.1.12 ft_1x

uint32_t _SecurityMode_t::ft_1x

FT 802.1x

4.2.1.13 ft_1x_sha384

uint32_t _SecurityMode_t::ft_1x_sha384

FT 802.1x sha384

4.2.1.14 ft_psk

uint32_t _SecurityMode_t::ft_psk

FT PSK

4.2.1.15 ft_sae

uint32_t _SecurityMode_t::ft_sae

FT SAE

4.2.1.16 wpa3_1x_sha256

uint32_t _SecurityMode_t::wpa3_1x_sha256

WPA3 802.1x sha256

4.2.1.17 wpa3_1x_sha384

uint32_t _SecurityMode_t::wpa3_1x_sha384

WPA3 802.1x sha384

4.2.1.18 rsvd

uint32_t _SecurityMode_t::rsvd

Reserved 16 bits

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

- [wifi-decl.h](#)

4.3 BandConfig_t Struct Reference

Data Fields

- [ChanBand_e](#) chanBand: 2
- [ChanWidth_e](#) chanWidth: 2
- [Chan2Offset_e](#) chan2Offset: 2
- [ScanMode_e](#) scanMode: 2

4.3.1 Field Documentation

4.3.1.1 chanBand

[ChanBand_e](#) BandConfig_t::chanBand

4.3.1.2 chanWidth

[ChanWidth_e](#) BandConfig_t::chanWidth

4.3.1.3 chan2Offset

[Chan2Offset_e](#) BandConfig_t::chan2Offset

4.3.1.4 scanMode

[ScanMode_e](#) BandConfig_t::scanMode

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

- [wlan.h](#)

4.4 ChanBandInfo_t Struct Reference

Data Fields

- [BandConfig_t](#) bandConfig
- [uint8_t](#) chanNum

4.4.1 Field Documentation

4.4.1.1 bandConfig

`BandConfig_t ChanBandInfo_t::bandConfig`

4.4.1.2 chanNum

`uint8_t ChanBandInfo_t::chanNum`

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

- [wlan.h](#)

4.5 cli_command Struct Reference

Data Fields

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

4.5.1 Detailed Description

Structure for registering CLI commands

4.5.2 Field Documentation

4.5.2.1 name

`const char* cli_command::name`

The name of the CLI command

4.5.2.2 help

`const char* cli_command::help`

The help text associated with the command

4.5.2.3 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](#)

4.6 csi_local_buff_statu Struct Reference

Public Member Functions

- [OSA_SEMAPHORE_HANDLE_DEFINE](#) (csi_data_sem)

Data Fields

- [t_u8 write_index](#)
- [t_u8 read_index](#)
- [t_u8 valid_data_cnt](#)

4.6.1 Member Function Documentation

4.6.1.1 OSA_SEMAPHORE_HANDLE_DEFINE()

```
csi_local_buff_statu::OSA_SEMAPHORE_HANDLE_DEFINE (
    csi_data_sem )
```

Semaphore to protect data parameters

4.6.2 Field Documentation

4.6.2.1 write_index

```
t_u8 csi_local_buff_statu::write_index
```

4.6.2.2 read_index

```
t_u8 csi_local_buff_statu::read_index
```

4.6.2.3 valid_data_cnt

```
t_u8 csi_local_buff_statu::valid_data_cnt
```

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

- [wifi.h](#)

4.7 Event_Radar_Detected_Info Struct Reference

Data Fields

- t_u32 [detect_count](#)
- t_u8 [reg_domain](#)
- t_u8 [main_det_type](#)
- t_u16 [pw_chirp_type](#)
- t_u8 [pw_chirp_idx](#)
- t_u8 [pw_value](#)
- t_u8 [pri_radar_type](#)
- t_u8 [pri_binCnt](#)
- t_u8 [binCounter](#) [BIN_COUNTER_LEN]
- t_u8 [numDfsRecords](#)
- t_u8 [dfsRecordHdrs](#) [DFS_REC_HDR_NUM][DFS_REC_HDR_LEN]
- t_u32 [reallyPassed](#)

4.7.1 Field Documentation

4.7.1.1 detect_count

```
t_u32 Event_Radar_Detected_Info::detect_count
```

4.7.1.2 reg_domain

```
t_u8 Event_Radar_Detected_Info::reg_domain
```

4.7.1.3 main_det_type

t_u8 Event_Radar_Detected_Info::main_det_type

4.7.1.4 pw_chirp_type

t_u16 Event_Radar_Detected_Info::pw_chirp_type

4.7.1.5 pw_chirp_idx

t_u8 Event_Radar_Detected_Info::pw_chirp_idx

4.7.1.6 pw_value

t_u8 Event_Radar_Detected_Info::pw_value

4.7.1.7 pri_radar_type

t_u8 Event_Radar_Detected_Info::pri_radar_type

4.7.1.8 pri_binCnt

t_u8 Event_Radar_Detected_Info::pri_binCnt

4.7.1.9 binCounter

t_u8 Event_Radar_Detected_Info::binCounter[BIN_COUNTER_LEN]

4.7.1.10 numDfsRecords

t_u8 Event_Radar_Detected_Info::numDfsRecords

4.7.1.11 dfsRecordHdrs

```
t_u8 Event_Radar_Detected_Info::dfsRecordHdrs [DFS_REC_HDR_NUM] [DFS_REC_HDR_LEN]
```

4.7.1.12 reallyPassed

```
t_u32 Event_Radar_Detected_Info::reallyPassed
```

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

- [wlan.h](#)

4.8 ipv4_config Struct Reference

Data Fields

- enum [address_types](#) [addr_type](#)
- unsigned [address](#)
- unsigned [gw](#)
- unsigned [netmask](#)
- unsigned [dns1](#)
- unsigned [dns2](#)

4.8.1 Detailed Description

This data structure represents an IPv4 address

4.8.2 Field Documentation

4.8.2.1 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.

4.8.2.2 address

```
unsigned ipv4_config::address
```

The system's IP address in network order.

4.8.2.3 gw

```
unsigned ipv4_config::gw
```

The system's default gateway in network order.

4.8.2.4 netmask

```
unsigned ipv4_config::netmask
```

The system's subnet mask in network order.

4.8.2.5 dns1

```
unsigned ipv4_config::dns1
```

The system's primary dns server in network order.

4.8.2.6 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](#)

4.9 ipv6_config Struct Reference

Data Fields

- unsigned [address](#) [4]
- unsigned char [addr_type](#)
- unsigned char [addr_state](#)

4.9.1 Detailed Description

This data structure represents an IPv6 address

4.9.2 Field Documentation

4.9.2.1 address

```
unsigned ipv6_config::address[4]
```

The system's IPv6 address in network order.

4.9.2.2 addr_type

```
unsigned char ipv6_config::addr_type
```

The address type: linklocal, site-local or global.

4.9.2.3 addr_state

```
unsigned char ipv6_config::addr_state
```

The state of IPv6 address (Tentative, Preferred, etc).

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

- [wlan.h](#)

4.10 net_ip_config Struct Reference

Data Fields

- struct [net_ipv6_config](#) [ipv6](#) [CONFIG_MAX_IPV6_ADDRESSES]
- size_t [ipv6_count](#)
- struct [net_ipv4_config](#) [ipv4](#)

4.10.1 Detailed Description

Network IP configuration.

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

4.10.2 Field Documentation

4.10.2.1 ipv6

```
struct net\_ipv6\_config net_ip_config::ipv6[CONFIG_MAX_IPV6_ADDRESSES]
```

The network IPv6 address configuration that should be associated with this interface.

4.10.2.2 ipv6_count

```
size_t net_ip_config::ipv6_count
```

The network IPv6 valid addresses count

4.10.2.3 ipv4

```
struct net_ipv4_config net_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:

- [wm_net.h](#)

4.11 net_ipv4_config Struct Reference

Data Fields

- enum [net_address_types](#) [addr_type](#)
- unsigned [address](#)
- unsigned [gw](#)
- unsigned [netmask](#)
- unsigned [dns1](#)
- unsigned [dns2](#)

4.11.1 Detailed Description

This data structure represents an IPv4 address

4.11.2 Field Documentation

4.11.2.1 addr_type

```
enum net_address_types net_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.

4.11.2.2 address

```
unsigned net_ipv4_config::address
```

The system's IP address in network order.

4.11.2.3 gw

```
unsigned net_ipv4_config::gw
```

The system's default gateway in network order.

4.11.2.4 netmask

```
unsigned net_ipv4_config::netmask
```

The system's subnet mask in network order.

4.11.2.5 dns1

```
unsigned net_ipv4_config::dns1
```

The system's primary dns server in network order.

4.11.2.6 dns2

```
unsigned net_ipv4_config::dns2
```

The system's secondary dns server in network order.

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

- [wm_net.h](#)

4.12 net_ipv6_config Struct Reference

Data Fields

- unsigned [address](#) [4]
- unsigned char [addr_type](#)
- unsigned char [addr_state](#)

4.12.1 Detailed Description

This data structure represents an IPv6 address

4.12.2 Field Documentation

4.12.2.1 address

```
unsigned net_ipv6_config::address[4]
```

The system's IPv6 address in network order.

4.12.2.2 addr_type

```
unsigned char net_ipv6_config::addr_type
```

The address type: linklocal, site-local or global.

4.12.2.3 addr_state

```
unsigned char net_ipv6_config::addr_state
```

The state of IPv6 address (Tentative, Preferred, etc).

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

- [wm_net.h](#)

4.13 osa_rw_lock_t Struct Reference

Public Member Functions

- [OSA_MUTEX_HANDLE_DEFINE](#) (reader_mutex)
- [OSA_MUTEX_HANDLE_DEFINE](#) (write_mutex)
- [OSA_SEMAPHORE_HANDLE_DEFINE](#) (rw_lock)

Data Fields

- [cb_fn](#) reader_cb
- unsigned int [reader_count](#)

4.13.1 Member Function Documentation

4.13.1.1 OSA_MUTEX_HANDLE_DEFINE() [1/2]

```
osa_rw_lock_t::OSA_MUTEX_HANDLE_DEFINE (
    reader_mutex )
```

Mutex for reader mutual exclusion

4.13.1.2 OSA_MUTEX_HANDLE_DEFINE() [2/2]

```
osa_rw_lock_t::OSA_MUTEX_HANDLE_DEFINE (
    write_mutex )
```

Mutex for write mutual exclusion

4.13.1.3 OSA_SEMAPHORE_HANDLE_DEFINE()

```
osa_rw_lock_t::OSA_SEMAPHORE_HANDLE_DEFINE (
    rw_lock )
```

Lock which when held by reader, writer cannot enter critical section

4.13.2 Field Documentation

4.13.2.1 reader_cb

```
cb_fn osa_rw_lock_t::reader_cb
```

Function being called when first reader gets the lock

4.13.2.2 reader_count

```
unsigned int osa_rw_lock_t::reader_count
```

Counter to maintain number of readers in critical section

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

- [osa.h](#)

4.14 test_cfg_param_t Struct Reference

Data Fields

- const char * [name](#)
- int [offset](#)
- int [len](#)
- const char * [notes](#)

4.14.1 Field Documentation

4.14.1.1 name

```
const char* test_cfg_param_t::name
```

4.14.1.2 offset

```
int test_cfg_param_t::offset
```

4.14.1.3 len

```
int test_cfg_param_t::len
```

4.14.1.4 notes

```
const char* test_cfg_param_t::notes
```

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

- [wlan_tests.h](#)

4.15 test_cfg_table_t Struct Reference

Data Fields

- const char * [name](#)
- uint8_t * [data](#)
- int [len](#)
- const [test_cfg_param_t](#) * [param_list](#)
- int [param_num](#)

4.15.1 Field Documentation

4.15.1.1 name

```
const char* test_cfg_table_t::name
```

4.15.1.2 data

```
uint8_t* test_cfg_table_t::data
```

4.15.1.3 len

```
int test_cfg_table_t::len
```

4.15.1.4 param_list

```
const test_cfg_param_t* test_cfg_table_t::param_list
```

4.15.1.5 param_num

```
int test_cfg_table_t::param_num
```

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

- [wlan_tests.h](#)

4.16 tx_ampdu_prot_mode_para Struct Reference

Data Fields

- int [mode](#)

4.16.1 Detailed Description

tx_ampdu_prot_mode parameters

4.16.2 Field Documentation

4.16.2.1 mode

```
int tx_ampdu_prot_mode_para::mode
```

set prot mode

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

- [wlan.h](#)

4.17 txrate_setting Struct Reference

Data Fields

- t_u16 [preamble](#): 2
- t_u16 [bandwidth](#): 3
- t_u16 [shortGI](#): 2
- t_u16 [stbc](#): 1
- t_u16 [dcm](#): 1
- t_u16 [adv_coding](#): 1
- t_u16 [doppler](#): 2
- t_u16 [max_pkttext](#): 2
- t_u16 [reserverd](#): 2

4.17.1 Detailed Description

TX Rate Setting

4.17.2 Field Documentation

4.17.2.1 preamble

```
t_u16 txrate_setting::preamble
```

Preamble

4.17.2.2 bandwidth

```
t_u16 txrate_setting::bandwidth
```

Bandwidth

4.17.2.3 shortGI

```
t_u16 txrate_setting::shortGI
```

Short GI

4.17.2.4 stbc

```
t_u16 txrate_setting::stbc
```

STBC

4.17.2.5 dcm

```
t_u16 txrate_setting::dcm
```

DCM

4.17.2.6 adv_coding

```
t_u16 txrate_setting::adv_coding
```

Adv coding

4.17.2.7 doppler

```
t_u16 txrate_setting::doppler
```

Doppler

4.17.2.8 max_pkttext

```
t_u16 txrate_setting::max_pkttext
```

Max PK text

4.17.2.9 reserverd

```
t_u16 txrate_setting::reserverd
```

Reserved

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

- [wifi-decl.h](#)

4.18 wifi_11ax_config_t Struct Reference

Data Fields

- t_u8 [band](#)
- t_u16 [id](#)
- t_u16 [len](#)
- t_u8 [ext_id](#)
- t_u8 [he_mac_cap](#) [6]
- t_u8 [he_phy_cap](#) [11]
- t_u8 [he_txrx_mcs_support](#) [4]
- t_u8 [val](#) [4]

4.18.1 Detailed Description

Wi-Fi 11AX Configuration

4.18.2 Field Documentation

4.18.2.1 band

t_u8 wifi_11ax_config_t::band

Band

4.18.2.2 id

t_u16 wifi_11ax_config_t::id

tlv id of he capability

4.18.2.3 len

t_u16 wifi_11ax_config_t::len

length of the payload

4.18.2.4 ext_id

t_u8 wifi_11ax_config_t::ext_id

extension id

4.18.2.5 he_mac_cap

```
t_u8 wifi_11ax_config_t::he_mac_cap[6]
```

he mac capability info

4.18.2.6 he_phy_cap

```
t_u8 wifi_11ax_config_t::he_phy_cap[11]
```

he phy capability info

4.18.2.7 he_txrx_mcs_support

```
t_u8 wifi_11ax_config_t::he_txrx_mcs_support[4]
```

he txrx mcs support for 80MHz

4.18.2.8 val

```
t_u8 wifi_11ax_config_t::val[4]
```

val for PE thresholds

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

- [wifi-decl.h](#)

4.19 wifi_antcfg_t Struct Reference

Data Fields

- t_u32 * [ant_mode](#)
- t_u16 * [evaluate_time](#)
- t_u16 * [current_antenna](#)

4.19.1 Detailed Description

Type definition of [wifi_antcfg_t](#)

4.19.2 Field Documentation

4.19.2.1 ant_mode

t_u32* wifi_antcfg_t::ant_mode

Antenna Mode

4.19.2.2 evaluate_time

t_u16* wifi_antcfg_t::evaluate_time

Evaluate Time

4.19.2.3 current_antenna

t_u16* wifi_antcfg_t::current_antenna

Current antenna

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

- [wifi-decl.h](#)

4.20 wifi_auto_reconnect_config_t Struct Reference

Data Fields

- t_u8 [reconnect_counter](#)
- t_u8 [reconnect_interval](#)
- t_u16 [flags](#)

4.20.1 Detailed Description

Auto reconnect structure

4.20.2 Field Documentation

4.20.2.1 reconnect_counter

t_u8 wifi_auto_reconnect_config_t::reconnect_counter

Reconnect counter

4.20.2.2 reconnect_interval

t_u8 wifi_auto_reconnect_config_t::reconnect_interval

Reconnect interval

4.20.2.3 flags

t_u16 wifi_auto_reconnect_config_t::flags

Flags

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

- [wifi-decl.h](#)

4.21 wifi_bandcfg_t Struct Reference

Data Fields

- t_u16 [config_bands](#)
- t_u16 [fw_bands](#)

4.21.1 Detailed Description

Type definition of [wifi_bandcfg_t](#)

4.21.2 Field Documentation

4.21.2.1 config_bands

t_u16 wifi_bandcfg_t::config_bands

Infra band

4.21.2.2 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](#)

4.22 wifi_btwt_config_t Struct Reference

Data Fields

- t_u16 [action](#)
- t_u16 [sub_id](#)
- t_u8 [nominal_wake](#)
- t_u8 [max_sta_support](#)
- t_u16 [tw_t_mantissa](#)
- t_u16 [tw_t_offset](#)
- t_u8 [tw_t_exponent](#)
- t_u8 [sp_gap](#)

4.22.1 Detailed Description

Wi-Fi BTWT Configuration

4.22.2 Field Documentation

4.22.2.1 action

t_u16 wifi_btwt_config_t::action

Only support 1: Set

4.22.2.2 sub_id

t_u16 wifi_btwt_config_t::sub_id

Broadcast TWT AP config

4.22.2.3 nominal_wake

t_u8 wifi_btwt_config_t::nominal_wake

Range 64-255

4.22.2.4 max_sta_support

t_u8 wifi_btwt_config_t::max_sta_support

Max STA Support

4.22.2.5 twt_mantissa

```
t_u16 wifi_btwt_config_t::twt_mantissa
```

TWT Mantissa

4.22.2.6 twt_offset

```
t_u16 wifi_btwt_config_t::twt_offset
```

TWT Offset

4.22.2.7 twt_exponent

```
t_u8 wifi_btwt_config_t::twt_exponent
```

TWT Exponent

4.22.2.8 sp_gap

```
t_u8 wifi_btwt_config_t::sp_gap
```

SP Gap

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

- [wifi-decl.h](#)

4.23 wifi_cal_data_t Struct Reference

Data Fields

- t_u16 [data_len](#)
- t_u8 * [data](#)

4.23.1 Detailed Description

Calibration Data

4.23.2 Field Documentation

4.23.2.1 data_len

```
t_u16 wifi_cal_data_t::data_len
```

Calibration data length

4.23.2.2 data

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

Calibration data

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

- [wifi-decl.h](#)

4.24 wifi_chan_info_t Struct Reference

Data Fields

- t_u8 [chan_num](#)
- t_u16 [chan_freq](#)
- bool [passive_scan_or_radar_detect](#)

4.24.1 Detailed Description

Data structure for Channel attributes

4.24.2 Field Documentation

4.24.2.1 chan_num

```
t_u8 wifi_chan_info_t::chan_num
```

Channel Number

4.24.2.2 chan_freq

```
t_u16 wifi_chan_info_t::chan_freq
```

Channel frequency for this channel

4.24.2.3 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](#)

4.25 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]

4.25.1 Detailed Description

Channel list parameter set

4.25.2 Field Documentation

4.25.2.1 no_of_channels

```
t_u8 wifi_chan_list_param_set_t::no_of_channels
```

number of channels

4.25.2.2 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](#)

4.26 wifi_chan_scan_param_set_t Struct Reference

Data Fields

- [t_u8 chan_number](#)
- [t_u16 min_scan_time](#)
- [t_u16 max_scan_time](#)

4.26.1 Detailed Description

Channel scan parameters

4.26.2 Field Documentation

4.26.2.1 chan_number

`t_u8 wifi_chan_scan_param_set_t::chan_number`

channel number

4.26.2.2 min_scan_time

`t_u16 wifi_chan_scan_param_set_t::min_scan_time`

minimum scan time

4.26.2.3 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](#)

4.27 wifi_chanlist_t Struct Reference

Data Fields

- [t_u8 num_chans](#)
- [wifi_chan_info_t chan_info](#) [54]

4.27.1 Detailed Description

Data structure for Channel List Config

4.27.2 Field Documentation

4.27.2.1 num_chans

```
t_u8 wifi_chanlist_t::num_chans
```

Number of Channels

4.27.2.2 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](#)

4.28 wifi_channel_desc_t Struct Reference

Data Fields

- t_u16 [start_freq](#)
- t_u8 [chan_width](#)
- t_u8 [chan_num](#)

4.28.1 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

4.28.2 Field Documentation

4.28.2.1 start_freq

t_u16 wifi_channel_desc_t::start_freq

Starting frequency of the band for this channel

4.28.2.2 chan_width

t_u8 wifi_channel_desc_t::chan_width

Channel width

4.28.2.3 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](#)

4.29 wifi_clock_sync_gpio_tsf_t Struct Reference

Data Fields

- t_u8 [clock_sync_mode](#)
- t_u8 [clock_sync_Role](#)
- t_u8 [clock_sync_gpio_pin_number](#)
- t_u8 [clock_sync_gpio_level_toggle](#)
- t_u16 [clock_sync_gpio_pulse_width](#)

4.29.1 Detailed Description

Wi-Fi Clock sync configuration

4.29.2 Field Documentation

4.29.2.1 clock_sync_mode

t_u8 wifi_clock_sync_gpio_tsf_t::clock_sync_mode

clock sync Mode

4.29.2.2 clock_sync_Role

t_u8 wifi_clock_sync_gpio_tsf_t::clock_sync_Role

clock sync Role

4.29.2.3 clock_sync_gpio_pin_number

t_u8 wifi_clock_sync_gpio_tsf_t::clock_sync_gpio_pin_number

clock sync GPIO Pin Number

4.29.2.4 clock_sync_gpio_level_toggle

t_u8 wifi_clock_sync_gpio_tsf_t::clock_sync_gpio_level_toggle

clock sync GPIO Level or Toggle

4.29.2.5 clock_sync_gpio_pulse_width

t_u16 wifi_clock_sync_gpio_tsf_t::clock_sync_gpio_pulse_width

clock sync GPIO Pulse Width

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

- [wifi-decl.h](#)

4.30 wifi_cloud_keep_alive_t Struct Reference

Data Fields

- t_u8 [mkeep_alive_id](#)
- t_u8 [enable](#)
- t_u8 [reset](#)
- t_u8 [cached](#)
- t_u32 [send_interval](#)
- t_u16 [retry_interval](#)
- t_u16 [retry_count](#)
- t_u8 [src_mac](#) [MLAN_MAC_ADDR_LENGTH]
- t_u8 [dst_mac](#) [MLAN_MAC_ADDR_LENGTH]
- t_u32 [src_ip](#)
- t_u32 [dst_ip](#)
- t_u16 [src_port](#)
- t_u16 [dst_port](#)
- t_u16 [pkt_len](#)
- t_u8 [packet](#) [MKEEP_ALIVE_IP_PKT_MAX]

4.30.1 Detailed Description

Cloud keep alive information

4.30.2 Field Documentation

4.30.2.1 mkeep_alive_id

`t_u8 wifi_cloud_keep_alive_t::mkeep_alive_id`

Keep alive id

4.30.2.2 enable

`t_u8 wifi_cloud_keep_alive_t::enable`

Enable keep alive

4.30.2.3 reset

`t_u8 wifi_cloud_keep_alive_t::reset`

Enable/Disable tcp reset

4.30.2.4 cached

`t_u8 wifi_cloud_keep_alive_t::cached`

Saved in driver

4.30.2.5 send_interval

`t_u32 wifi_cloud_keep_alive_t::send_interval`

Period to send keep alive packet(The unit is milliseconds)

4.30.2.6 retry_interval

`t_u16 wifi_cloud_keep_alive_t::retry_interval`

Period to send retry packet(The unit is milliseconds)

4.30.2.7 retry_count

t_u16 wifi_cloud_keep_alive_t::retry_count

Count to send retry packet

4.30.2.8 src_mac

t_u8 wifi_cloud_keep_alive_t::src_mac[MLAN_MAC_ADDR_LENGTH]

Source MAC address

4.30.2.9 dst_mac

t_u8 wifi_cloud_keep_alive_t::dst_mac[MLAN_MAC_ADDR_LENGTH]

Destination MAC address

4.30.2.10 src_ip

t_u32 wifi_cloud_keep_alive_t::src_ip

Source IP

4.30.2.11 dst_ip

t_u32 wifi_cloud_keep_alive_t::dst_ip

Destination IP

4.30.2.12 src_port

t_u16 wifi_cloud_keep_alive_t::src_port

Source Port

4.30.2.13 dst_port

t_u16 wifi_cloud_keep_alive_t::dst_port

Destination Port

4.30.2.14 pkt_len

t_u16 wifi_cloud_keep_alive_t::pkt_len

Packet length

4.30.2.15 packet

```
t_u8 wifi_cloud_keep_alive_t::packet[MKEEP_ALIVE_IP_PKT_MAX]
```

Packet buffer

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

- [wifi-decl.h](#)

4.31 wifi_csi_config_params_t Struct Reference

Data Fields

- t_u8 [bss_type](#)
- t_u16 [csi_enable](#)
- t_u32 [head_id](#)
- t_u32 [tail_id](#)
- t_u8 [csi_filter_cnt](#)
- t_u8 [chip_id](#)
- t_u8 [band_config](#)
- t_u8 [channel](#)
- t_u8 [csi_monitor_enable](#)
- t_u8 [ra4us](#)
- [wifi_csi_filter_t](#) [csi_filter](#) [CSI_FILTER_MAX]

4.31.1 Detailed Description

Structure of CSI parameters

4.31.2 Field Documentation

4.31.2.1 bss_type

```
t_u8 wifi_csi_config_params_t::bss_type
```

4.31.2.2 csi_enable

```
t_u16 wifi_csi_config_params_t::csi_enable
```

CSI enable flag. 1: enable, 2: disable

4.31.2.3 head_id

t_u32 wifi_csi_config_params_t::head_id

Header ID

4.31.2.4 tail_id

t_u32 wifi_csi_config_params_t::tail_id

Tail ID

4.31.2.5 csi_filter_cnt

t_u8 wifi_csi_config_params_t::csi_filter_cnt

Number of CSI filters

4.31.2.6 chip_id

t_u8 wifi_csi_config_params_t::chip_id

Chip ID

4.31.2.7 band_config

t_u8 wifi_csi_config_params_t::band_config

band config

4.31.2.8 channel

t_u8 wifi_csi_config_params_t::channel

Channel num

4.31.2.9 csi_monitor_enable

t_u8 wifi_csi_config_params_t::csi_monitor_enable

Enable getting CSI data on special channel

4.31.2.10 ra4us

t_u8 wifi_csi_config_params_t::ra4us

CSI data received in cfg channel with mac addr filter, not only RA is us or other

4.31.2.11 csi_filter

```
wifi_csi_filter_t wifi_csi_config_params_t::csi_filter[CSI_FILTER_MAX]
```

CSI filters

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

- [wifi-decl.h](#)

4.32 wifi_csi_filter_t Struct Reference

Data Fields

- `t_u8 mac_addr` [MLAN_MAC_ADDR_LENGTH]
- `t_u8 pkt_type`
- `t_u8 subtype`
- `t_u8 flags`

4.32.1 Detailed Description

Structure of CSI filters

4.32.2 Field Documentation

4.32.2.1 mac_addr

```
t_u8 wifi_csi_filter_t::mac_addr[MLAN_MAC_ADDR_LENGTH]
```

Source address of the packet to receive

4.32.2.2 pkt_type

```
t_u8 wifi_csi_filter_t::pkt_type
```

Pakcet type of the interested CSI

4.32.2.3 subtype

```
t_u8 wifi_csi_filter_t::subtype
```

Packet subtype of the interested CSI

4.32.2.4 flags

```
t_u8 wifi_csi_filter_t::flags
```

Other filter flags

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

- [wifi-decl.h](#)

4.33 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](#)

4.33.1 Detailed Description

CW_MODE_CTRL structure

4.33.2 Field Documentation

4.33.2.1 mode

```
t_u8 wifi_cw_mode_ctrl_t::mode
```

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

4.33.2.2 channel

```
t_u8 wifi_cw_mode_ctrl_t::channel
```

channel

4.33.2.3 chanInfo

```
t_u8 wifi_cw_mode_ctrl_t::chanInfo
```

channel info

4.33.2.4 txPower

t_u16 wifi_cw_mode_ctrl_t::txPower

Tx Power level in dBm

4.33.2.5 pktLength

t_u16 wifi_cw_mode_ctrl_t::pktLength

Packet Length

4.33.2.6 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](#)

4.34 wifi_data_rate_t Struct Reference

Data Fields

- t_u32 [tx_data_rate](#)
- t_u32 [rx_data_rate](#)
- t_u32 [tx_bw](#)
- t_u32 [tx_gi](#)
- t_u32 [rx_bw](#)
- t_u32 [rx_gi](#)

4.34.1 Detailed Description

Data structure for cmd get data rate

4.34.2 Field Documentation

4.34.2.1 tx_data_rate

t_u32 wifi_data_rate_t::tx_data_rate

Tx data rate

4.34.2.2 rx_data_rate

```
t_u32 wifi_data_rate_t::rx_data_rate
```

Rx data rate

4.34.2.3 tx_bw

```
t_u32 wifi_data_rate_t::tx_bw
```

Tx channel bandwidth

4.34.2.4 tx_gi

```
t_u32 wifi_data_rate_t::tx_gi
```

Tx guard interval

4.34.2.5 rx_bw

```
t_u32 wifi_data_rate_t::rx_bw
```

Rx channel bandwidth

4.34.2.6 rx_gi

```
t_u32 wifi_data_rate_t::rx_gi
```

Rx guard interval

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

- [wifi-decl.h](#)

4.35 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

4.35.1 Detailed Description

Type definition of [wifi_ds_rate](#)

4.35.2 Field Documentation

4.35.2.1 sub_command

```
enum wifi\_ds\_command\_type wifi_ds_rate::sub_command
```

Sub-command

4.35.2.2 rate_cfg

```
wifi\_rate\_cfg\_t wifi_ds_rate::rate_cfg
```

Rate configuration for MLAN_OID_RATE_CFG

4.35.2.3 data_rate

```
wifi\_data\_rate\_t wifi_ds_rate::data_rate
```

Data rate for MLAN_OID_GET_DATA_RATE

4.35.2.4 param

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

Rate configuration parameter

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

- [wifi-decl.h](#)

4.36 wifi_ecsa_info Struct Reference

Data Fields

- [t_u8 bss_type](#)
- [t_u8 band_config](#)
- [t_u8 channel](#)

4.36.1 Field Documentation

4.36.1.1 bss_type

`t_u8 wifi_ecsa_info::bss_type`

4.36.1.2 band_config

`t_u8 wifi_ecsa_info::band_config`

4.36.1.3 channel

`t_u8 wifi_ecsa_info::channel`

channel

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

- [wifi.h](#)

4.37 wifi_ed_mac_ctrl_t Struct Reference

Data Fields

- `t_u16` [ed_ctrl_2g](#)
- `t_s16` [ed_offset_2g](#)
- `t_u16` [ed_ctrl_5g](#)
- `t_s16` [ed_offset_5g](#)

4.37.1 Detailed Description

Type definition of [wifi_ed_mac_ctrl_t](#)

4.37.2 Field Documentation

4.37.2.1 ed_ctrl_2g

t_u16 wifi_ed_mac_ctrl_t::ed_ctrl_2g

ED CTRL 2G

4.37.2.2 ed_offset_2g

t_s16 wifi_ed_mac_ctrl_t::ed_offset_2g

ED Offset 2G

4.37.2.3 ed_ctrl_5g

t_u16 wifi_ed_mac_ctrl_t::ed_ctrl_5g

ED CTRL 5G

4.37.2.4 ed_offset_5g

t_s16 wifi_ed_mac_ctrl_t::ed_offset_5g

ED Offset 5G

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

- [wifi-decl.h](#)

4.38 wifi_ext_coex_config_t Struct Reference

Data Fields

- t_u8 [Enabled](#)
- t_u8 [IgnorePriority](#)
- t_u8 [DefaultPriority](#)
- t_u8 [EXT_RADIO_REQ_ip_gpio_num](#)
- t_u8 [EXT_RADIO_REQ_ip_gpio_polarity](#)
- t_u8 [EXT_RADIO_PRI_ip_gpio_num](#)
- t_u8 [EXT_RADIO_PRI_ip_gpio_polarity](#)
- t_u8 [WLAN_GRANT_op_gpio_num](#)
- t_u8 [WLAN_GRANT_op_gpio_polarity](#)
- t_u16 [reserved_1](#)
- t_u16 [reserved_2](#)

4.38.1 Detailed Description

Type definition of [wifi_ext_coex_config_t](#)

4.38.2 Field Documentation

4.38.2.1 Enabled

`t_u8 wifi_ext_coex_config_t::Enabled`

Enable or disable external coexistence

4.38.2.2 IgnorePriority

`t_u8 wifi_ext_coex_config_t::IgnorePriority`

Ignore the priority of the external radio request

4.38.2.3 DefaultPriority

`t_u8 wifi_ext_coex_config_t::DefaultPriority`

Default priority when the priority of the external radio request is ignored

4.38.2.4 EXT_RADIO_REQ_ip_gpio_num

`t_u8 wifi_ext_coex_config_t::EXT_RADIO_REQ_ip_gpio_num`

Input request GPIO pin for EXT_RADIO_REQ signal

4.38.2.5 EXT_RADIO_REQ_ip_gpio_polarity

`t_u8 wifi_ext_coex_config_t::EXT_RADIO_REQ_ip_gpio_polarity`

Input request GPIO polarity for EXT_RADIO_REQ signal

4.38.2.6 EXT_RADIO_PRI_ip_gpio_num

`t_u8 wifi_ext_coex_config_t::EXT_RADIO_PRI_ip_gpio_num`

Input priority GPIO pin for EXT_RADIO_PRI signal

4.38.2.7 EXT_RADIO_PRI_ip_gpio_polarity

`t_u8 wifi_ext_coex_config_t::EXT_RADIO_PRI_ip_gpio_polarity`

Input priority GPIO polarity for EXT_RADIO_PRI signal

4.38.2.8 WLAN_GRANT_op_gpio_num

t_u8 wifi_ext_coex_config_t::WLAN_GRANT_op_gpio_num

Output grant GPIO pin for WLAN_GRANT signal

4.38.2.9 WLAN_GRANT_op_gpio_polarity

t_u8 wifi_ext_coex_config_t::WLAN_GRANT_op_gpio_polarity

Output grant GPIO polarity of WLAN_GRANT

4.38.2.10 reserved_1

t_u16 wifi_ext_coex_config_t::reserved_1

Reserved Bytes

4.38.2.11 reserved_2

t_u16 wifi_ext_coex_config_t::reserved_2

Reserved Bytes

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

- [wifi-decl.h](#)

4.39 wifi_ext_coex_stats_t Struct Reference

Data Fields

- t_u16 [ext_radio_req_count](#)
- t_u16 [ext_radio_pri_count](#)
- t_u16 [wlan_grant_count](#)

4.39.1 Detailed Description

Type definition of [wifi_ext_coex_stats_t](#)

4.39.2 Field Documentation

4.39.2.1 ext_radio_req_count

t_u16 wifi_ext_coex_stats_t::ext_radio_req_count

External Radio Request count

4.39.2.2 ext_radio_pri_count

t_u16 wifi_ext_coex_stats_t::ext_radio_pri_count

External Radio Priority count

4.39.2.3 wlan_grant_count

t_u16 wifi_ext_coex_stats_t::wlan_grant_count

WLAN GRANT count

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

- [wifi-decl.h](#)

4.40 wififlt_cfg_t Struct Reference

Data Fields

- t_u32 [criteria](#)
- t_u16 [nentries](#)
- [wifi_mef_entry_t](#) [mef_entry](#) [MAX_NUM_ENTRIES]

4.40.1 Detailed Description

Wifi filter config struct

4.40.2 Field Documentation

4.40.2.1 criteria

t_u32 wififlt_cfg_t::criteria

Filter Criteria

4.40.2.2 nentries

```
t_u16 wifi_flt_cfg_t::nentries
```

Number of entries

4.40.2.3 mef_entry

```
wifi_mef_entry_t wifi_flt_cfg_t::mef_entry[MAX_NUM_ENTRIES]
```

MEF entry

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

- [wifi-decl.h](#)

4.41 wifi_frame_t Struct Reference

Data Fields

- [wifi_frame_type_t frame_type](#)

4.41.1 Field Documentation

4.41.1.1 frame_type

```
wifi_frame_type_t wifi_frame_t::frame_type
```

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

- [wifi-decl.h](#)

4.42 wifi_fw_version_ext_t Struct Reference

Data Fields

- uint8_t [version_str_sel](#)
- char [version_str](#) [MLAN_MAX_VER_STR_LEN]

4.42.1 Detailed Description

Extended Firmware version

4.42.2 Field Documentation

4.42.2.1 version_str_sel

```
uint8_t wifi_fw_version_ext_t::version_str_sel
```

ID for extended version select

4.42.2.2 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](#)

4.43 wifi_fw_version_t Struct Reference

Data Fields

- char [version_str](#) [MLAN_MAX_VER_STR_LEN]

4.43.1 Detailed Description

Firmware version

4.43.2 Field Documentation

4.43.2.1 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](#)

4.44 wifi_indrst_cfg_t Struct Reference

Data Fields

- t_u8 [ir_mode](#)
- t_u8 [gpio_pin](#)

4.44.1 Detailed Description

Wi-Fi independent reset config

4.44.2 Field Documentation

4.44.2.1 ir_mode

t_u8 wifi_indrst_cfg_t::ir_mode

reset mode enable/ disable

4.44.2.2 gpio_pin

t_u8 wifi_indrst_cfg_t::gpio_pin

gpio pin

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

- [wifi-decl.h](#)

4.45 wifi_mac_addr_t Struct Reference

Data Fields

- char [mac](#) [MLAN_MAC_ADDR_LENGTH]

4.45.1 Detailed Description

MAC address

4.45.2 Field Documentation

4.45.2.1 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](#)

4.46 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]

4.46.1 Detailed Description

MEF entry struct

4.46.2 Field Documentation

4.46.2.1 mode

```
t_u8 wifi_mef_entry_t::mode
```

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

4.46.2.2 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;

4.46.2.3 filter_num

`t_u8 wifi_mef_entry_t::filter_num`

filter number

4.46.2.4 filter_item

`wifi_mef_filter_t wifi_mef_entry_t::filter_item[MAX_NUM_FILTERS]`

filter array

4.46.2.5 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](#)

4.47 wifi_mef_filter_t Struct Reference

Data Fields

- `t_u32 fill_flag`
- `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]`

4.47.1 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

4.47.2 Field Documentation

4.47.2.1 fill_flag

t_u32 wifi_mef_filter_t::fill_flag

flag

4.47.2.2 type

t_u16 wifi_mef_filter_t::type

BYTE 0X41; Decimal 0X42; Bit 0x43

4.47.2.3 pattern

t_u32 wifi_mef_filter_t::pattern

value

4.47.2.4 offset

t_u16 wifi_mef_filter_t::offset

offset

4.47.2.5 num_bytes

t_u16 wifi_mef_filter_t::num_bytes

number of bytes

4.47.2.6 repeat

t_u16 wifi_mef_filter_t::repeat

repeat

4.47.2.7 num_byte_seq

t_u8 wifi_mef_filter_t::num_byte_seq

byte number

4.47.2.8 byte_seq

t_u8 wifi_mef_filter_t::byte_seq[[MAX_NUM_BYTE_SEQ](#)]

array

4.47.2.9 num_mask_seq

```
t_u8 wifi_mef_filter_t::num_mask_seq
```

mask numbers

4.47.2.10 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](#)

4.48 wifi_message Struct Reference

Data Fields

- uint16_t [event](#)
- enum [wifi_event_reason](#) [reason](#)
- void * [data](#)

4.48.1 Field Documentation

4.48.1.1 event

```
uint16_t wifi_message::event
```

4.48.1.2 reason

```
enum wifi\_event\_reason wifi_message::reason
```

4.48.1.3 data

```
void* wifi_message::data
```

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

- [wifi-decl.h](#)

4.49 wifi_mfg_cmd_generic_cfg_t Struct Reference

Data Fields

- t_u32 [mfg_cmd](#)
- t_u16 [action](#)
- t_u16 [device_id](#)
- t_u32 [error](#)
- t_u32 [data1](#)
- t_u32 [data2](#)
- t_u32 [data3](#)

4.49.1 Detailed Description

Configuration for Manufacturing generic command

4.49.2 Field Documentation

4.49.2.1 mfg_cmd

t_u32 wifi_mfg_cmd_generic_cfg_t::mfg_cmd

MFG command code

4.49.2.2 action

t_u16 wifi_mfg_cmd_generic_cfg_t::action

Action

4.49.2.3 device_id

t_u16 wifi_mfg_cmd_generic_cfg_t::device_id

Device ID

4.49.2.4 error

t_u32 wifi_mfg_cmd_generic_cfg_t::error

MFG Error code

4.49.2.5 data1

t_u32 wifi_mfg_cmd_generic_cfg_t::data1

value 1

4.49.2.6 data2

t_u32 wifi_mfg_cmd_generic_cfg_t::data2

value 2

4.49.2.7 data3

t_u32 wifi_mfg_cmd_generic_cfg_t::data3

value 3

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

- [wifi-decl.h](#)

4.50 wifi_mfg_cmd_he_tb_tx_t Struct Reference

Data Fields

- t_u32 [mfg_cmd](#)
- t_u16 [action](#)
- t_u16 [device_id](#)
- t_u32 [error](#)
- t_u16 [enable](#)
- t_u16 [qnum](#)
- t_u16 [aid](#)
- t_u16 [axq_mu_timer](#)
- t_s16 [tx_power](#)

4.50.1 Field Documentation

4.50.1.1 mfg_cmd

t_u32 wifi_mfg_cmd_he_tb_tx_t::mfg_cmd

MFG command code

4.50.1.2 action

t_u16 wifi_mfg_cmd_he_tb_tx_t::action

Action

4.50.1.3 device_id

t_u16 wifi_mfg_cmd_he_tb_tx_t::device_id

Device ID

4.50.1.4 error

t_u32 wifi_mfg_cmd_he_tb_tx_t::error

MFG Error code

4.50.1.5 enable

t_u16 wifi_mfg_cmd_he_tb_tx_t::enable

Enable Tx

4.50.1.6 qnum

t_u16 wifi_mfg_cmd_he_tb_tx_t::qnum

Q num

4.50.1.7 aid

t_u16 wifi_mfg_cmd_he_tb_tx_t::aid

AID

4.50.1.8 axq_mu_timer

t_u16 wifi_mfg_cmd_he_tb_tx_t::axq_mu_timer

AXQ Mu Timer

4.50.1.9 tx_power

```
t_s16 wifi_mfg_cmd_he_tb_tx_t::tx_power
```

Tx Power

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

- [wifi-decl.h](#)

4.51 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t Struct Reference

Data Fields

- t_u32 [mfg_cmd](#)
- t_u16 [action](#)
- t_u16 [device_id](#)
- t_u32 [error](#)
- t_u32 [enable_tx](#)
- t_u32 [standalone_hetb](#)
- mfg_cmd_IEEEtypes_FrameCtrl_t [frmCtl](#)
- t_u16 [duration](#)
- t_u8 [dest_addr](#) [MLAN_MAC_ADDR_LENGTH]
- t_u8 [src_addr](#) [MLAN_MAC_ADDR_LENGTH]
- mfg_cmd_IEEEtypes_HETrigComInfo_t [trig_common_field](#)
- mfg_cmd_IEEEtypes_HETrigUserInfo_t [trig_user_info_field](#)
- mfg_cmd_IEEEtypes_BasicHETrigUserInfo_t [basic_trig_user_info](#)

4.51.1 Field Documentation

4.51.1.1 mfg_cmd

```
t_u32 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::mfg_cmd
```

MFG command code

4.51.1.2 action

```
t_u16 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::action
```

Action

4.51.1.3 device_id

```
t_u16 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::device_id
```

Device ID

4.51.1.4 error

```
t_u32 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::error
```

MFG Error code

4.51.1.5 enable_tx

```
t_u32 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::enable_tx
```

enable Tx

4.51.1.6 standalone_hetb

```
t_u32 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::standalone_hetb
```

enable Stand Alone HE TB

4.51.1.7 frmCtl

```
mfg_cmd_IEEEtypes_FrameCtrl_t wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::frmCtl
```

Frame Control

4.51.1.8 duration

```
t_u16 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::duration
```

Duration

4.51.1.9 dest_addr

```
t_u8 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::dest_addr[MLAN_MAC_ADDR_LENGTH]
```

Destination MAC Address

4.51.1.10 src_addr

```
t_u8 wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::src_addr[MLAN_MAC_ADDR_LENGTH]
```

Source MAC Address

4.51.1.11 trig_common_field

```
mfg_cmd_IEEEtypes_HETrigComInfo_t wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::trig_common_field
```

Common Info Field

4.51.1.12 trig_user_info_field

```
mfg_cmd_IEEEtypes_HETrigUserInfo_t wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::trig_user_info_↵  
field
```

User Info Field

4.51.1.13 basic_trig_user_info

```
mfg_cmd_IEEEtypes_BasicHETrigUserInfo_t wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t::basic_trig_↵  
_user_info
```

Trigger Dependent User Info Field

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

- [wifi-decl.h](#)

4.52 wifi_mfg_cmd_otp_cal_data_rd_wr_t Struct Reference

Data Fields

- t_u32 [mfg_cmd](#)
- t_u16 [action](#)
- t_u16 [device_id](#)
- t_u32 [error](#)
- t_u32 [cal_data_status](#)
- t_u32 [cal_data_len](#)
- t_u8 [cal_data](#) [CAL_DATA_LEN]

4.52.1 Field Documentation

4.52.1.1 mfg_cmd

```
t_u32 wifi_mfg_cmd_otp_cal_data_rd_wr_t::mfg_cmd
```

MFG command code

4.52.1.2 action

```
t_u16 wifi_mfg_cmd_otp_cal_data_rd_wr_t::action
```

Action

4.52.1.3 device_id

```
t_u16 wifi_mfg_cmd_otp_cal_data_rd_wr_t::device_id
```

Device ID

4.52.1.4 error

```
t_u32 wifi_mfg_cmd_otp_cal_data_rd_wr_t::error
```

MFG Error code

4.52.1.5 cal_data_status

```
t_u32 wifi_mfg_cmd_otp_cal_data_rd_wr_t::cal_data_status
```

CAL Data write status

4.52.1.6 cal_data_len

```
t_u32 wifi_mfg_cmd_otp_cal_data_rd_wr_t::cal_data_len
```

CAL Data Length

4.52.1.7 cal_data

```
t_u8 wifi_mfg_cmd_otp_cal_data_rd_wr_t::cal_data[CAL_DATA_LEN]
```

Destination MAC Address

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

- [wifi-decl.h](#)

4.53 wifi_mfg_cmd_otp_mac_addr_rd_wr_t Struct Reference

Data Fields

- t_u32 [mfg_cmd](#)
- t_u16 [action](#)
- t_u16 [device_id](#)
- t_u32 [error](#)
- t_u8 [mac_addr](#) [MLAN_MAC_ADDR_LENGTH]

4.53.1 Field Documentation

4.53.1.1 mfg_cmd

`t_u32 wifi_mfg_cmd_otp_mac_addr_rd_wr_t::mfg_cmd`

MFG command code

4.53.1.2 action

`t_u16 wifi_mfg_cmd_otp_mac_addr_rd_wr_t::action`

Action

4.53.1.3 device_id

`t_u16 wifi_mfg_cmd_otp_mac_addr_rd_wr_t::device_id`

Device ID

4.53.1.4 error

`t_u32 wifi_mfg_cmd_otp_mac_addr_rd_wr_t::error`

MFG Error code

4.53.1.5 mac_addr

`t_u8 wifi_mfg_cmd_otp_mac_addr_rd_wr_t::mac_addr` [\[MLAN_MAC_ADDR_LENGTH\]](#)

Destination MAC Address

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

- [wifi-decl.h](#)

4.54 wifi_mfg_cmd_tx_cont_t Struct Reference

Data Fields

- `t_u32` [mfg_cmd](#)
- `t_u16` [action](#)
- `t_u16` [device_id](#)
- `t_u32` [error](#)
- `t_u32` [enable_tx](#)
- `t_u32` [cw_mode](#)
- `t_u32` [payload_pattern](#)
- `t_u32` [cs_mode](#)
- `t_u32` [act_sub_ch](#)
- `t_u32` [tx_rate](#)
- `t_u32` [rsvd](#)

4.54.1 Detailed Description

Configuration for Manufacturing command Tx Continuous

4.54.2 Field Documentation

4.54.2.1 mfg_cmd

`t_u32 wifi_mfg_cmd_tx_cont_t::mfg_cmd`

MFG command code

4.54.2.2 action

`t_u16 wifi_mfg_cmd_tx_cont_t::action`

Action

4.54.2.3 device_id

`t_u16 wifi_mfg_cmd_tx_cont_t::device_id`

Device ID

4.54.2.4 error

`t_u32 wifi_mfg_cmd_tx_cont_t::error`

MFG Error code

4.54.2.5 enable_tx

`t_u32 wifi_mfg_cmd_tx_cont_t::enable_tx`

enable Tx

4.54.2.6 cw_mode

`t_u32 wifi_mfg_cmd_tx_cont_t::cw_mode`

Continuous Wave mode

4.54.2.7 payload_pattern

```
t_u32 wifi_mfg_cmd_tx_cont_t::payload_pattern
```

payload pattern

4.54.2.8 cs_mode

```
t_u32 wifi_mfg_cmd_tx_cont_t::cs_mode
```

CS Mode

4.54.2.9 act_sub_ch

```
t_u32 wifi_mfg_cmd_tx_cont_t::act_sub_ch
```

active sub channel

4.54.2.10 tx_rate

```
t_u32 wifi_mfg_cmd_tx_cont_t::tx_rate
```

Tx rate

4.54.2.11 rsvd

```
t_u32 wifi_mfg_cmd_tx_cont_t::rsvd
```

power id

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

- [wifi-decl.h](#)

4.55 wifi_mfg_cmd_tx_frame_t Struct Reference

Data Fields

- t_u32 mfg_cmd
- t_u16 action
- t_u16 device_id
- t_u32 error
- t_u32 enable
- t_u32 data_rate
- t_u32 frame_pattern
- t_u32 frame_length
- t_u8 bssid [MLAN_MAC_ADDR_LENGTH]
- t_u16 adjust_burst_sifs
- t_u32 burst_sifs_in_us
- t_u32 short_preamble
- t_u32 act_sub_ch
- t_u32 short_gi
- t_u32 adv_coding
- t_u32 tx_bf
- t_u32 gf_mode
- t_u32 stbc
- t_u32 rsvd [1]
- t_u32 signal_bw
- t_u32 NumPkt
- t_u32 MaxPE
- t_u32 BeamChange
- t_u32 Dcm
- t_u32 Doppler
- t_u32 MidP
- t_u32 QNum

4.55.1 Detailed Description

Configuration for Manufacturing command Tx Frame

4.55.2 Field Documentation

4.55.2.1 mfg_cmd

t_u32 wifi_mfg_cmd_tx_frame_t::mfg_cmd

MFG command code

4.55.2.2 action

t_u16 wifi_mfg_cmd_tx_frame_t::action

Action

4.55.2.3 device_id

t_u16 wifi_mfg_cmd_tx_frame_t::device_id

Device ID

4.55.2.4 error

t_u32 wifi_mfg_cmd_tx_frame_t::error

MFG Error code

4.55.2.5 enable

t_u32 wifi_mfg_cmd_tx_frame_t::enable

enable

4.55.2.6 data_rate

t_u32 wifi_mfg_cmd_tx_frame_t::data_rate

data_rate

4.55.2.7 frame_pattern

t_u32 wifi_mfg_cmd_tx_frame_t::frame_pattern

frame pattern

4.55.2.8 frame_length

t_u32 wifi_mfg_cmd_tx_frame_t::frame_length

frame length

4.55.2.9 bssid

t_u8 wifi_mfg_cmd_tx_frame_t::bssid[MLAN_MAC_ADDR_LENGTH]

BSSID

4.55.2.10 adjust_burst_sifs

t_u16 wifi_mfg_cmd_tx_frame_t::adjust_burst_sifs

Adjust burst sifs

4.55.2.11 burst_sifs_in_us

t_u32 wifi_mfg_cmd_tx_frame_t::burst_sifs_in_us

Burst sifs in us

4.55.2.12 short_preamble

t_u32 wifi_mfg_cmd_tx_frame_t::short_preamble

short preamble

4.55.2.13 act_sub_ch

t_u32 wifi_mfg_cmd_tx_frame_t::act_sub_ch

active sub channel

4.55.2.14 short_gi

t_u32 wifi_mfg_cmd_tx_frame_t::short_gi

short GI

4.55.2.15 adv_coding

t_u32 wifi_mfg_cmd_tx_frame_t::adv_coding

Adv coding

4.55.2.16 tx_bf

t_u32 wifi_mfg_cmd_tx_frame_t::tx_bf

Tx beamforming

4.55.2.17 gf_mode

t_u32 wifi_mfg_cmd_tx_frame_t::gf_mode

HT Greenfield Mode

4.55.2.18 stbc

```
t_u32 wifi_mfg_cmd_tx_frame_t::stbc
```

STBC

4.55.2.19 rsvd

```
t_u32 wifi_mfg_cmd_tx_frame_t::rsvd[1]
```

power id

4.55.2.20 signal_bw

```
t_u32 wifi_mfg_cmd_tx_frame_t::signal_bw
```

signal bw

4.55.2.21 NumPkt

```
t_u32 wifi_mfg_cmd_tx_frame_t::NumPkt
```

NumPkt

4.55.2.22 MaxPE

```
t_u32 wifi_mfg_cmd_tx_frame_t::MaxPE
```

MaxPE

4.55.2.23 BeamChange

```
t_u32 wifi_mfg_cmd_tx_frame_t::BeamChange
```

BeamChange

4.55.2.24 Dcm

```
t_u32 wifi_mfg_cmd_tx_frame_t::Dcm
```

Dcm

4.55.2.25 Doppler

```
t_u32 wifi_mfg_cmd_tx_frame_t::Doppler
```

Doppler

4.55.2.26 MidP

`t_u32 wifi_mfg_cmd_tx_frame_t::MidP`

MidP

4.55.2.27 QNum

`t_u32 wifi_mfg_cmd_tx_frame_t::QNum`

QNum

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

- [wifi-decl.h](#)

4.56 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 [1]`

4.56.1 Detailed Description

802_11_header packet

4.56.2 Field Documentation

4.56.2.1 frm_len

`t_u16 wifi_mgmt_frame_t::frm_len`

Packet Length

4.56.2.2 frame_type

wifi_frame_type_t wifi_mgmt_frame_t::frame_type

Frame Type

4.56.2.3 frame_ctrl_flags

t_u8 wifi_mgmt_frame_t::frame_ctrl_flags

Frame Control flags

4.56.2.4 duration_id

t_u16 wifi_mgmt_frame_t::duration_id

Duration ID

4.56.2.5 addr1

t_u8 wifi_mgmt_frame_t::addr1[MLAN_MAC_ADDR_LENGTH]

Address 1

4.56.2.6 addr2

t_u8 wifi_mgmt_frame_t::addr2[MLAN_MAC_ADDR_LENGTH]

Address 2

4.56.2.7 addr3

t_u8 wifi_mgmt_frame_t::addr3[MLAN_MAC_ADDR_LENGTH]

Address 3

4.56.2.8 seq_ctl

t_u16 wifi_mgmt_frame_t::seq_ctl

Sequence Control

4.56.2.9 addr4

t_u8 wifi_mgmt_frame_t::addr4[MLAN_MAC_ADDR_LENGTH]

Address 4

4.56.2.10 payload

```
t_u8 wifi_mgmt_frame_t::payload[1]
```

Frame payload

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

- [wifi-decl.h](#)

4.57 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](#)

4.57.1 Detailed Description

TCP nat keep alive information

4.57.2 Field Documentation

4.57.2.1 interval

```
t_u16 wifi_nat_keep_alive_t::interval
```

Keep alive interval

4.57.2.2 dst_mac

```
t_u8 wifi_nat_keep_alive_t::dst_mac[MLAN\_MAC\_ADDR\_LENGTH]
```

Destination MAC address

4.57.2.3 dst_ip

```
t_u32 wifi_nat_keep_alive_t::dst_ip
```

Destination IP

4.57.2.4 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](#)

4.58 wifi_os_mem_info Struct Reference

Data Fields

- char [name](#) [[MAX_FUNC_SYMBOL_LEN](#)]
- t_u32 [size](#)
- t_u32 [line_num](#)
- t_u32 [alloc_cnt](#)
- t_u32 [free_cnt](#)

4.58.1 Field Documentation

4.58.1.1 name

```
char wifi_os_mem_info::name[MAX\_FUNC\_SYMBOL\_LEN]
```

4.58.1.2 size

```
t_u32 wifi_os_mem_info::size
```

4.58.1.3 line_num

```
t_u32 wifi_os_mem_info::line_num
```

4.58.1.4 alloc_cnt

```
t_u32 wifi_os_mem_info::alloc_cnt
```

4.58.1.5 free_cnt

```
t_u32 wifi_os_mem_info::free_cnt
```

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

- [wifi-decl.h](#)

4.59 wifi_pmf_params_t Struct Reference

Data Fields

- uint8_t [mfpc](#)
- uint8_t [mfpr](#)

4.59.1 Field Documentation

4.59.1.1 mfpc

```
uint8_t wifi_pmf_params_t::mfpc
```

4.59.1.2 mfpr

```
uint8_t wifi_pmf_params_t::mfpr
```

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

- [wifi-decl.h](#)

4.60 wifi_rate_cfg_t Struct Reference

Data Fields

- mlan_rate_format [rate_format](#)
- t_u32 [rate_index](#)
- t_u32 [rate](#)
- t_u32 [nss](#)
- t_u16 [rate_setting](#)

4.60.1 Detailed Description

Data structure for cmd txratecfg

4.60.2 Field Documentation

4.60.2.1 rate_format

mlan_rate_format wifi_rate_cfg_t::rate_format

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

4.60.2.2 rate_index

t_u32 wifi_rate_cfg_t::rate_index

Rate/MCS index (0xFF: auto)

4.60.2.3 rate

t_u32 wifi_rate_cfg_t::rate

Rate rate

4.60.2.4 nss

t_u32 wifi_rate_cfg_t::nss

NSS

4.60.2.5 rate_setting

t_u16 wifi_rate_cfg_t::rate_setting

Rate Setting

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

- [wifi-decl.h](#)

4.61 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](#)

4.61.1 Detailed Description

Remain on channel info structure

4.61.2 Field Documentation

4.61.2.1 remove

```
uint16_t wifi_remain_on_channel_t::remove
```

Remove

4.61.2.2 status

```
uint8_t wifi_remain_on_channel_t::status
```

Current status

4.61.2.3 bandcfg

```
uint8_t wifi_remain_on_channel_t::bandcfg
```

band configuration

4.61.2.4 channel

```
uint8_t wifi_remain_on_channel_t::channel
```

Channel

4.61.2.5 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](#)

4.62 wifi_rf_channel_t Struct Reference

Data Fields

- uint16_t [current_channel](#)
- uint16_t [rf_type](#)

4.62.1 Detailed Description

Rf channel

4.62.2 Field Documentation

4.62.2.1 current_channel

```
uint16_t wifi_rf_channel_t::current_channel
```

Current channel

4.62.2.2 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](#)

4.63 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](#)

4.63.1 Detailed Description

RSSI information

4.63.2 Field Documentation

4.63.2.1 data_rssi_last

```
int16_t wifi_rssi_info_t::data_rssi_last
```

Data RSSI last

4.63.2.2 data_nf_last

```
int16_t wifi_rssi_info_t::data_nf_last
```

Data nf last

4.63.2.3 data_rssi_avg

```
int16_t wifi_rssi_info_t::data_rssi_avg
```

Data RSSI average

4.63.2.4 data_nf_avg

```
int16_t wifi_rssi_info_t::data_nf_avg
```

Data nf average

4.63.2.5 bcn_snr_last

```
int16_t wifi_rssi_info_t::bcn_snr_last
```

BCN SNR

4.63.2.6 bcn_snr_avg

```
int16_t wifi_rssi_info_t::bcn_snr_avg
```

BCN SNR average

4.63.2.7 data_snr_last

```
int16_t wifi_rssi_info_t::data_snr_last
```

Data SNR last

4.63.2.8 data_snr_avg

```
int16_t wifi_rssi_info_t::data_snr_avg
```

Data SNR average

4.63.2.9 bcn_rssi_last

```
int16_t wifi_rssi_info_t::bcn_rssi_last
```

BCN RSSI

4.63.2.10 bcn_nf_last

```
int16_t wifi_rssi_info_t::bcn_nf_last
```

BCN nf

4.63.2.11 bcn_rssi_avg

```
int16_t wifi_rssi_info_t::bcn_rssi_avg
```

BCN RSSI average

4.63.2.12 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](#)

4.64 wifi_rupwrlimit_config_t Struct Reference

Data Fields

- t_u16 [start_freq](#)
- t_u8 [width](#)
- t_u8 [chan_num](#)
- t_s16 [ruPower](#) [MAX_RU_COUNT]

4.64.1 Field Documentation

4.64.1.1 start_freq

```
t_u16 wifi_rupwrlimit_config_t::start_freq
```

start freq

4.64.1.2 width

```
t_u8 wifi_rupwrlimit_config_t::width
```

4.64.1.3 chan_num

```
t_u8 wifi_rupwrlimit_config_t::chan_num
```

channel number

4.64.1.4 ruPower

```
t_s16 wifi_rupwrlimit_config_t::ruPower[MAX_RU_COUNT]
```

chan ru Power

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

- [wifi-decl.h](#)

4.65 wifi_rutxpwrlimit_t Struct Reference

Data Fields

- [t_u8 num_chans](#)
- [wifi_rupwrlimit_config_t rupwrlimit_config](#) [MAX_RUTXPWR_NUM]

4.65.1 Detailed Description

Data structure for Channel RU PWR config

For RU PWR support

4.65.2 Field Documentation

4.65.2.1 num_chans

```
t_u8 wifi_rutxpwrlimit_t::num_chans
```

Number of Channels

4.65.2.2 rupwrlimit_config

```
wifi_rupwrlimit_config_t wifi_rutxpwrlimit_t::rupwrlimit_config[MAX_RUTXPWR_NUM]
```

RU PWR config

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

- [wifi-decl.h](#)

4.66 wifi_scan_chan_list_t Struct Reference

Data Fields

- [uint8_t num_of_chan](#)
- [uint8_t chan_number](#) [MLAN_MAX_CHANNEL]

4.66.1 Detailed Description

Channel list structure

4.66.2 Field Documentation

4.66.2.1 num_of_chan

```
uint8_t wifi_scan_chan_list_t::num_of_chan
```

Number of channels

4.66.2.2 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](#)

4.67 wifi_scan_channel_list_t Struct Reference

Data Fields

- [t_u8 radio_type](#)
- [t_u8 chan_number](#)
- [mlan_scan_type scan_type](#)
- [t_u16 scan_time](#)

4.67.1 Detailed Description

Scan channel list

4.67.2 Field Documentation

4.67.2.1 radio_type

t_u8 wifi_scan_channel_list_t::radio_type

Channel scan parameter : Radio type

4.67.2.2 chan_number

t_u8 wifi_scan_channel_list_t::chan_number

Channel number

4.67.2.3 scan_type

mlan_scan_type wifi_scan_channel_list_t::scan_type

Scan type Active = 1, Passive = 2

4.67.2.4 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](#)

4.68 wifi_scan_params_t Struct Reference

Data Fields

- uint8_t * [bssid](#)
- char * [ssid](#)
- int [channel](#) [MAX_CHANNEL_LIST]
- [IEEEtypes_Bss_t](#) [bss_type](#)
- int [scan_duration](#)
- int [split_scan_delay](#)

4.68.1 Field Documentation

4.68.1.1 bssid

```
uint8_t* wifi_scan_params_t::bssid
```

4.68.1.2 ssid

```
char* wifi_scan_params_t::ssid
```

4.68.1.3 channel

```
int wifi_scan_params_t::channel[MAX\_CHANNEL\_LIST]
```

4.68.1.4 bss_type

```
IEEEtypes\_Bss\_t wifi_scan_params_t::bss_type
```

4.68.1.5 scan_duration

```
int wifi_scan_params_t::scan_duration
```

4.68.1.6 split_scan_delay

```
int wifi_scan_params_t::split_scan_delay
```

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

- [wlan.h](#)

4.69 wifi_scan_params_v2_t Struct Reference

Data Fields

- [t_u8 scan_only](#)
- [t_u8 is_bssid](#)
- [t_u8 is_ssid](#)
- [t_u8 bssid](#) [[MLAN_MAC_ADDR_LENGTH](#)]
- [char ssid](#) [[MAX_NUM_SSID](#)][[MLAN_MAX_SSID_LENGTH](#)+1]
- [t_u8 num_channels](#)
- [wifi_scan_channel_list_t chan_list](#) [[MAX_CHANNEL_LIST](#)]
- [t_u8 num_probes](#)
- [t_u16 scan_chan_gap](#)
- [int\(* cb\)](#) (unsigned int count)

4.69.1 Detailed Description

V2 scan parameters

4.69.2 Field Documentation

4.69.2.1 scan_only

t_u8 wifi_scan_params_v2_t::scan_only

Scan Only

4.69.2.2 is_bssid

t_u8 wifi_scan_params_v2_t::is_bssid

BSSID present

4.69.2.3 is_ssid

t_u8 wifi_scan_params_v2_t::is_ssid

SSID present

4.69.2.4 bssid

t_u8 wifi_scan_params_v2_t::bssid[MLAN_MAC_ADDR_LENGTH]

BSSID to scan

4.69.2.5 ssid

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

SSID to scan

4.69.2.6 num_channels

t_u8 wifi_scan_params_v2_t::num_channels

Number of channels

4.69.2.7 chan_list

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

Channel list with channel information

4.69.2.8 num_probes

```
t_u8 wifi_scan_params_v2_t::num_probes
```

Number of probes

4.69.2.9 scan_chan_gap

```
t_u16 wifi_scan_params_v2_t::scan_chan_gap
```

scan channel gap

4.69.2.10 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](#)

4.70 wifi_scan_result2 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](#)
- uint16_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](#)
- t_u8 [ap_mfpc](#)
- t_u8 [ap_mfpr](#)

- t_u8 ap_pwe
- bool phtcap_ie_present
- bool phtinfo_ie_present
- bool pvhtcap_ie_present
- bool phecap_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
- bool mbo_assoc_disallowed
- uint16_t mdid
- bool neighbor_report_supported
- bool bss_transition_supported

4.70.1 Detailed Description

Scan result information

4.70.2 Field Documentation

4.70.2.1 bssid

```
uint8_t wifi_scan_result2::bssid[MLAN_MAC_ADDR_LENGTH]
```

BSSID array

4.70.2.2 is_ibss_bit_set

```
bool wifi_scan_result2::is_ibss_bit_set
```

Is bssid set?

4.70.2.3 ssid

```
uint8_t wifi_scan_result2::ssid[MLAN_MAX_SSID_LENGTH]
```

ssid array

4.70.2.4 ssid_len

```
int wifi_scan_result2::ssid_len
```

SSID length

4.70.2.5 Channel

```
uint8_t wifi_scan_result2::Channel
```

Channel associated to the BSSID

4.70.2.6 RSSI

```
uint8_t wifi_scan_result2::RSSI
```

Received signal strength

4.70.2.7 beacon_period

```
uint16_t wifi_scan_result2::beacon_period
```

Beacon period

4.70.2.8 dtim_period

```
uint16_t wifi_scan_result2::dtim_period
```

DTIM period

4.70.2.9 WPA_WPA2_WEP

```
_SecurityMode_t wifi_scan_result2::WPA_WPA2_WEP
```

Security mode info

4.70.2.10 wpa_mcstCipher

```
_Cipher_t wifi_scan_result2::wpa_mcstCipher
```

WPA multicast cipher

4.70.2.11 wpa_ucstCipher

```
_Cipher_t wifi_scan_result2::wpa_ucstCipher
```

WPA unicast cipher

4.70.2.12 rsn_mcstCipher

`_Cipher_t` wifi_scan_result2::rsn_mcstCipher

No security multicast cipher

4.70.2.13 rsn_ucstCipher

`_Cipher_t` wifi_scan_result2::rsn_ucstCipher

No security unicast cipher

4.70.2.14 is_pmf_required

`bool` wifi_scan_result2::is_pmf_required

Is pmf required flag

4.70.2.15 ap_mfpc

`t_u8` wifi_scan_result2::ap_mfpc

MFPC bit of AP

4.70.2.16 ap_mfpr

`t_u8` wifi_scan_result2::ap_mfpr

MFPR bit of AP

4.70.2.17 ap_pwe

`t_u8` wifi_scan_result2::ap_pwe

PWE bit of AP WPA_WPA2 = 0 => Security not enabled = 1 => WPA mode = 2 => WPA2 mode = 3 => WEP mode

4.70.2.18 phtcap_ie_present

`bool` wifi_scan_result2::phtcap_ie_present

PHT CAP IE present info

4.70.2.19 phtinfo_ie_present

```
bool wifi_scan_result2::phtinfo_ie_present
```

PHT INFO IE present info

4.70.2.20 pvhtcap_ie_present

```
bool wifi_scan_result2::pvhtcap_ie_present
```

11AC VHT capab support

4.70.2.21 phecap_ie_present

```
bool wifi_scan_result2::phecap_ie_present
```

11AX HE capab support

4.70.2.22 wmm_ie_present

```
bool wifi_scan_result2::wmm_ie_present
```

WMM IE present info

4.70.2.23 band

```
uint16_t wifi_scan_result2::band
```

Band info

4.70.2.24 wps_IE_exist

```
bool wifi_scan_result2::wps_IE_exist
```

WPS IE exist info

4.70.2.25 wps_session

```
uint16_t wifi_scan_result2::wps_session
```

WPS session

4.70.2.26 wpa2_entp_IE_exist

```
bool wifi_scan_result2::wpa2_entp_IE_exist
```

WPA2 enterprise IE exist info

4.70.2.27 trans_mode

```
uint8_t wifi_scan_result2::trans_mode
```

Trans mode

4.70.2.28 trans_bssid

```
uint8_t wifi_scan_result2::trans_bssid[MLAN_MAC_ADDR_LENGTH]
```

Trans bssid array

4.70.2.29 trans_ssid

```
uint8_t wifi_scan_result2::trans_ssid[MLAN_MAX_SSID_LENGTH]
```

Trans ssid array

4.70.2.30 trans_ssid_len

```
int wifi_scan_result2::trans_ssid_len
```

Trans bssid length

4.70.2.31 mbo_assoc_disallowed

```
bool wifi_scan_result2::mbo_assoc_disallowed
```

MBO disallowed

4.70.2.32 mdid

```
uint16_t wifi_scan_result2::mdid
```

Mobility domain identifier

4.70.2.33 neighbor_report_supported

```
bool wifi_scan_result2::neighbor_report_supported
```

Neighbor report support

4.70.2.34 bss_transition_supported

```
bool wifi_scan_result2::bss_transition_supported
```

bss transition support

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

- [wifi-decl.h](#)

4.71 wifi_sta_info_t Struct Reference

Data Fields

- `t_u8 mac` [MLAN_MAC_ADDR_LENGTH]
- `t_u8 power_mgmt_status`
- `t_s8 rssi`

4.71.1 Detailed Description

Station information structure

4.71.2 Field Documentation

4.71.2.1 mac

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

MAC address buffer

4.71.2.2 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

4.71.2.3 rssi

```
t_s8 wifi_sta_info_t::rssi
```

RSSI: dBm

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

- [wifi-decl.h](#)

4.72 wifi_sta_list_t Struct Reference

Data Fields

- int [count](#)

4.72.1 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.

4.72.2 Field Documentation

4.72.2.1 count

```
int wifi_sta_list_t::count
```

Count

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

- [wifi-decl.h](#)

4.73 wifi_sub_band_set_t Struct Reference

Data Fields

- t_u8 [first_chan](#)
- t_u8 [no_of_chan](#)
- t_u8 [max_tx_pwr](#)

4.73.1 Detailed Description

Data structure for subband set

For uAP 11d support

4.73.2 Field Documentation

4.73.2.1 first_chan

`t_u8 wifi_sub_band_set_t::first_chan`

First channel

4.73.2.2 no_of_chan

`t_u8 wifi_sub_band_set_t::no_of_chan`

Number of channels

4.73.2.3 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](#)

4.74 wifi_tbtt_offset_t Struct Reference

Data Fields

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

4.74.1 Detailed Description

TBTT offset structure

4.74.2 Field Documentation

4.74.2.1 min_tbtt_offset

`t_u32 wifi_tbtt_offset_t::min_tbtt_offset`

Min TBTT offset

4.74.2.2 max_tbtt_offset

t_u32 wifi_tbtt_offset_t::max_tbtt_offset

Max TBTT offset

4.74.2.3 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](#)

4.75 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](#)

4.75.1 Detailed Description

TCP keep alive information

4.75.2 Field Documentation

4.75.2.1 enable

t_u8 wifi_tcp_keep_alive_t::enable

Enable keep alive

4.75.2.2 reset

```
t_u8 wifi_tcp_keep_alive_t::reset
```

Reset

4.75.2.3 timeout

```
t_u32 wifi_tcp_keep_alive_t::timeout
```

Keep alive timeout

4.75.2.4 interval

```
t_u16 wifi_tcp_keep_alive_t::interval
```

Keep alive interval

4.75.2.5 max_keep_alives

```
t_u16 wifi_tcp_keep_alive_t::max_keep_alives
```

Maximum keep alives

4.75.2.6 dst_mac

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

Destination MAC address

4.75.2.7 dst_ip

```
t_u32 wifi_tcp_keep_alive_t::dst_ip
```

Destination IP

4.75.2.8 dst_tcp_port

```
t_u16 wifi_tcp_keep_alive_t::dst_tcp_port
```

Destination TCP port

4.75.2.9 src_tcp_port

```
t_u16 wifi_tcp_keep_alive_t::src_tcp_port
```

Source TCP port

4.75.2.10 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](#)

4.76 wifi_tsf_info_t Struct Reference

Data Fields

- t_u16 [tsf_format](#)
- t_u16 [tsf_info](#)
- t_u64 [tsf](#)
- t_s32 [tsf_offset](#)

4.76.1 Detailed Description

Wi-Fi TSF information

4.76.2 Field Documentation

4.76.2.1 tsf_format

t_u16 wifi_tsf_info_t::tsf_format

get tsf info format

4.76.2.2 tsf_info

t_u16 wifi_tsf_info_t::tsf_info

tsf info

4.76.2.3 tsf

t_u64 wifi_tsf_info_t::tsf

tsf

4.76.2.4 tsf_offset

```
t_s32 wifi_tsf_info_t::tsf_offset
```

Positive or negative offset in microsecond from Beacon TSF to GPIO toggle TSF

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

- [wifi-decl.h](#)

4.77 wifi_twt_information_t Struct Reference

Data Fields

- [t_u8 flow_identifier](#)
- [t_u32 suspend_duration](#)
- [t_u8 information_state](#)

4.77.1 Field Documentation

4.77.1.1 flow_identifier

```
t_u8 wifi_twt_information_t::flow_identifier
```

TWT Flow Identifier. Range: [0-7]

4.77.1.2 suspend_duration

```
t_u32 wifi_twt_information_t::suspend_duration
```

TWT operation suspend duration in milli seconds.

4.77.1.3 information_state

```
t_u8 wifi_twt_information_t::information_state
```

TWT information state from FW.

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

- [wifi-decl.h](#)

4.78 wifi_twt_report_t Struct Reference

Data Fields

- [t_u8 type](#)
- [t_u8 length](#)
- [t_u8 reserve](#) [2]
- [t_u8 data](#) [[WLAN_BTWT_REPORT_LEN](#) * [WLAN_BTWT_REPORT_MAX_NUM](#)]

4.78.1 Detailed Description

Wi-Fi TWT Report Configuration

4.78.2 Field Documentation

4.78.2.1 type

```
t_u8 wifi_twt_report_t::type
```

TWT report type, 0: BTWT id

4.78.2.2 length

```
t_u8 wifi_twt_report_t::length
```

TWT report length of value in data

4.78.2.3 reserve

```
t_u8 wifi_twt_report_t::reserve[2]
```

Reserved 2

4.78.2.4 data

```
t_u8 wifi_twt_report_t::data [WLAN\_BTWT\_REPORT\_LEN * WLAN\_BTWT\_REPORT\_MAX\_NUM]
```

TWT report buffer

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

- [wifi-decl.h](#)

4.79 wifi_twt_setup_config_t Struct Reference

Data Fields

- t_u8 [implicit](#)
- t_u8 [announced](#)
- t_u8 [trigger_enabled](#)
- t_u8 [twt_info_disabled](#)
- t_u8 [negotiation_type](#)
- t_u8 [twt_wakeup_duration](#)
- t_u8 [flow_identifier](#)
- t_u8 [hard_constraint](#)
- t_u8 [twt_exponent](#)
- t_u16 [twt_mantissa](#)
- t_u8 [twt_request](#)

4.79.1 Detailed Description

Wi-Fi TWT setup configuration

4.79.2 Field Documentation

4.79.2.1 implicit

t_u8 wifi_twt_setup_config_t::implicit

Implicit, 0: TWT session is explicit, 1: Session is implicit

4.79.2.2 announced

t_u8 wifi_twt_setup_config_t::announced

Announced, 0: Unannounced, 1: Announced TWT

4.79.2.3 trigger_enabled

t_u8 wifi_twt_setup_config_t::trigger_enabled

Trigger Enabled, 0: Non-Trigger enabled, 1: Trigger enabled TWT

4.79.2.4 twt_info_disabled

t_u8 wifi_twt_setup_config_t::twt_info_disabled

TWT Information Disabled, 0: TWT info enabled, 1: TWT info disabled

4.79.2.5 negotiation_type

t_u8 wifi_twt_setup_config_t::negotiation_type

Negotiation Type, 0: Future Individual TWT SP start time, 1: Next Wake TBTT time

4.79.2.6 twt_wakeup_duration

t_u8 wifi_twt_setup_config_t::twt_wakeup_duration

TWT Wakeup Duration, time after which the TWT requesting STA can transition to doze state

4.79.2.7 flow_identifier

t_u8 wifi_twt_setup_config_t::flow_identifier

Flow Identifier. Range: [0-7]

4.79.2.8 hard_constraint

t_u8 wifi_twt_setup_config_t::hard_constraint

Hard Constraint, 0: FW can tweak the TWT setup parameters if it is rejected by AP. 1: Firmware should not tweak any parameters.

4.79.2.9 twt_exponent

t_u8 wifi_twt_setup_config_t::twt_exponent

TWT Exponent, Range: [0-63]

4.79.2.10 twt_mantissa

t_u16 wifi_twt_setup_config_t::twt_mantissa

TWT Mantissa Range: [0-sizeof(UINT16)]

4.79.2.11 twt_request

t_u8 wifi_twt_setup_config_t::twt_request

TWT Request Type, 0: REQUEST_TWT, 1: SUGGEST_TWT

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

- [wifi-decl.h](#)

4.80 wifi_twt_teardown_config_t Struct Reference

Data Fields

- t_u8 [flow_identifier](#)
- t_u8 [negotiation_type](#)
- t_u8 [teardown_all_twt](#)

4.80.1 Detailed Description

Wi-Fi Teardown Configuration

4.80.2 Field Documentation

4.80.2.1 flow_identifier

t_u8 wifi_twt_teardown_config_t::flow_identifier

TWT Flow Identifier. Range: [0-7]

4.80.2.2 negotiation_type

t_u8 wifi_twt_teardown_config_t::negotiation_type

Negotiation Type. 0: Future Individual TWT SP start time, 1: Next Wake TBTT time

4.80.2.3 teardown_all_twt

t_u8 wifi_twt_teardown_config_t::teardown_all_twt

Tear down all TWT. 1: To teardown all TWT, 0 otherwise

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

- [wifi-decl.h](#)

4.81 wifi_tx_power_t Struct Reference

Data Fields

- uint16_t [current_level](#)
- uint8_t [max_power](#)
- uint8_t [min_power](#)

4.81.1 Detailed Description

Tx power levels

4.81.2 Field Documentation

4.81.2.1 current_level

```
uint16_t wifi_tx_power_t::current_level
```

Current power level

4.81.2.2 max_power

```
uint8_t wifi_tx_power_t::max_power
```

Maximum power level

4.81.2.3 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](#)

4.82 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](#) [20]

4.82.1 Detailed Description

Data structure for TRPC config

For TRPC support

4.82.2 Field Documentation

4.82.2.1 num_mod_grps

`t_u8 wifi_txpwrlimit_config_t::num_mod_grps`

Number of modulation groups

4.82.2.2 chan_desc

`wifi_channel_desc_t wifi_txpwrlimit_config_t::chan_desc`

Channel descriptor

4.82.2.3 txpwrlimit_entry

`wifi_txpwrlimit_entry_t wifi_txpwrlimit_config_t::txpwrlimit_entry[20]`

Channel Modulation groups

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

- [wifi-decl.h](#)

4.83 wifi_txpwrlimit_entry_t Struct Reference

Data Fields

- `t_u8 mod_group`
- `t_u8 tx_power`

4.83.1 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

4.83.2 Field Documentation

4.83.2.1 mod_group

t_u8 wifi_txpwrlimit_entry_t::mod_group

Modulation group

4.83.2.2 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](#)

4.84 wifi_txpwrlimit_t Struct Reference

Data Fields

- [wifi_SubBand_t](#) subband
- t_u8 num_chans
- [wifi_txpwrlimit_config_t](#) txpwrlimit_config [43]

4.84.1 Detailed Description

Data structure for Channel TRPC config

For TRPC support

4.84.2 Field Documentation

4.84.2.1 subband

wifi_SubBand_t wifi_txpwrlimit_t::subband

SubBand

4.84.2.2 num_chans

```
t_u8 wifi_txpwrlimit_t::num_chans
```

Number of Channels

4.84.2.3 txpwrlimit_config

```
wifi_txpwrlimit_config_t wifi_txpwrlimit_t::txpwrlimit_config[43]
```

TRPC config

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

- [wifi-decl.h](#)

4.85 wifi_uap_client_disassoc_t Struct Reference

Data Fields

- int [reason_code](#)
- t_u8 [sta_addr](#) [[MLAN_MAC_ADDR_LENGTH](#)]

4.85.1 Field Documentation

4.85.1.1 reason_code

```
int wifi_uap_client_disassoc_t::reason_code
```

4.85.1.2 sta_addr

```
t_u8 wifi_uap_client_disassoc_t::sta_addr [MLAN\_MAC\_ADDR\_LENGTH]
```

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

- [wifi.h](#)

4.86 wifi_wowlan_pattern_t Struct Reference

Data Fields

- t_u8 [pkt_offset](#)
- t_u8 [pattern_len](#)
- t_u8 [pattern](#) [[WOWLAN_MAX_PATTERN_LEN](#)]
- t_u8 [mask](#) [6]

4.86.1 Field Documentation

4.86.1.1 pkt_offset

t_u8 wifi_wowlan_pattern_t::pkt_offset

pattern offset of received pattern

4.86.1.2 pattern_len

t_u8 wifi_wowlan_pattern_t::pattern_len

pattern length

4.86.1.3 pattern

t_u8 wifi_wowlan_pattern_t::pattern [[WOWLAN_MAX_PATTERN_LEN](#)]

wowlan pattern

4.86.1.4 mask

t_u8 wifi_wowlan_pattern_t::mask [6]

mask

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

- [wifi-decl.h](#)

4.87 wifi_wowlan_ptn_cfg_t Struct Reference

Data Fields

- t_u8 [enable](#)
- t_u8 [n_patterns](#)
- [wifi_wowlan_pattern_t](#) [patterns](#) [[MAX_NUM_FILTERS](#)]

4.87.1 Detailed Description

Wowlan Pattern config struct

4.87.2 Field Documentation

4.87.2.1 enable

```
t_u8 wifi_wowlan_ptn_cfg_t::enable
```

Enable user defined pattern

4.87.2.2 n_patterns

```
t_u8 wifi_wowlan_ptn_cfg_t::n_patterns
```

number of patterns

4.87.2.3 patterns

```
wifi_wowlan_pattern_t wifi_wowlan_ptn_cfg_t::patterns[MAX_NUM_FILTERS]
```

user define pattern

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

- [wifi-decl.h](#)

4.88 wlan_cipher Struct Reference

Data Fields

- uint16_t [none](#): 1
- uint16_t [wep40](#): 1
- uint16_t [wep104](#): 1
- uint16_t [tkip](#): 1
- uint16_t [ccmp](#): 1
- uint16_t [aes_128_cmac](#): 1
- uint16_t [gcmp](#): 1
- uint16_t [sms4](#): 1
- uint16_t [gcmp_256](#): 1
- uint16_t [ccmp_256](#): 1
- uint16_t [rsvd](#): 1
- uint16_t [bip_gmac_128](#): 1
- uint16_t [bip_gmac_256](#): 1
- uint16_t [bip_cmac_256](#): 1
- uint16_t [gtk_not_used](#): 1
- uint16_t [rsvd2](#): 2

4.88.1 Detailed Description

Wlan Cipher structure

4.88.2 Field Documentation

4.88.2.1 none

```
uint16_t wlan_cipher::none
```

1 bit value can be set for none

4.88.2.2 wep40

```
uint16_t wlan_cipher::wep40
```

1 bit value can be set for wep40

4.88.2.3 wep104

```
uint16_t wlan_cipher::wep104
```

1 bit value can be set for wep104

4.88.2.4 tkip

```
uint16_t wlan_cipher::tkip
```

1 bit value can be set for tkip

4.88.2.5 ccmp

```
uint16_t wlan_cipher::ccmp
```

1 bit value can be set for ccmp

4.88.2.6 aes_128_cmac

```
uint16_t wlan_cipher::aes_128_cmac
```

1 bit value can be set for aes 128 cmac

4.88.2.7 gcmp

```
uint16_t wlan_cipher::gcmp
```

1 bit value can be set for gcmp

4.88.2.8 sms4

```
uint16_t wlan_cipher::sms4
```

1 bit value can be set for sms4

4.88.2.9 gcmp_256

```
uint16_t wlan_cipher::gcmp_256
```

1 bit value can be set for gcmp 256

4.88.2.10 ccmp_256

```
uint16_t wlan_cipher::ccmp_256
```

1 bit value can be set for ccmp 256

4.88.2.11 rsvd

```
uint16_t wlan_cipher::rsvd
```

1 bit is reserved

4.88.2.12 bip_gmac_128

```
uint16_t wlan_cipher::bip_gmac_128
```

1 bit value can be set for bip gmac 128

4.88.2.13 bip_gmac_256

```
uint16_t wlan_cipher::bip_gmac_256
```

1 bit value can be set for bip gmac 256

4.88.2.14 bip_cmac_256

```
uint16_t wlan_cipher::bip_cmac_256
```

1 bit value can be set for bip cmac 256

4.88.2.15 gtk_not_used

```
uint16_t wlan_cipher::gtk_not_used
```

1 bit value can be set for gtk not used

4.88.2.16 rsvd2

```
uint16_t wlan_cipher::rsvd2
```

4 bits are reserved

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

- [wlan.h](#)

4.89 wlan_ieee80211_config Struct Reference

Data Fields

- [t_u32 ps_null_interval](#)
- [t_u32 multiple_dtim_interval](#)
- [t_u32 listen_interval](#)
- [t_u32 adhoc_awake_period](#)
- [t_u32 bcn_miss_timeout](#)
- [t_s32 delay_to_ps](#)
- [t_u32 ps_mode](#)

4.89.1 Field Documentation

4.89.1.1 ps_null_interval

```
t_u32 wlan_ieee80211_config::ps_null_interval
```

PS null interval in seconds

4.89.1.2 multiple_dtim_interval

```
t_u32 wlan_ieee80211_config::multiple_dtim_interval
```

Multiple DTIM interval

4.89.1.3 listen_interval

```
t_u32 wlan_ieeeps_config::listen_interval
```

Listen interval

4.89.1.4 adhoc_aware_period

```
t_u32 wlan_ieeeps_config::adhoc_aware_period
```

Adhoc aware period

4.89.1.5 bcn_miss_timeout

```
t_u32 wlan_ieeeps_config::bcn_miss_timeout
```

Beacon miss timeout in milliseconds

4.89.1.6 delay_to_ps

```
t_s32 wlan_ieeeps_config::delay_to_ps
```

Delay to PS in milliseconds

4.89.1.7 ps_mode

```
t_u32 wlan_ieeeps_config::ps_mode
```

PS mode

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

- [wlan.h](#)

4.90 wlan_ip_config Struct Reference

Data Fields

- struct [ipv6_config](#) [ipv6](#) [CONFIG_MAX_IPV6_ADDRESSES]
- size_t [ipv6_count](#)
- struct [ipv4_config](#) [ipv4](#)

4.90.1 Detailed Description

Network IP configuration.

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

4.90.2 Field Documentation

4.90.2.1 ipv6

```
struct ipv6\_config wlan_ip_config::ipv6[CONFIG_MAX_IPV6_ADDRESSES]
```

The network IPv6 address configuration that should be associated with this interface.

4.90.2.2 ipv6_count

```
size_t wlan_ip_config::ipv6_count
```

The network IPv6 valid addresses count

4.90.2.3 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](#)

4.91 wlan_message Struct Reference

Data Fields

- [t_u16](#) [id](#)
- void * [data](#)

4.91.1 Field Documentation

4.91.1.1 id

```
t_u16 wlan_message::id
```

4.91.1.2 data

```
void* wlan_message::data
```

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

- [wlan.h](#)

4.92 wlan_network Struct Reference

Data Fields

- int [id](#)
- int [wps_network](#)
- char [name](#) [WLAN_NETWORK_NAME_MAX_LENGTH+1]
- char [ssid](#) [IEEEtypes_SSID_SIZE+1]
- char [bssid](#) [IEEEtypes_ADDRESS_SIZE]
- unsigned int [channel](#)
- uint8_t [sec_channel_offset](#)
- uint16_t [acs_band](#)
- int [rssi](#)
- unsigned short [ht_capab](#)
- unsigned int [vht_capab](#)
- unsigned char [vht_oper_chwidth](#)
- unsigned char [he_oper_chwidth](#)
- 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
- unsigned [dot11n](#): 1
- unsigned [dot11ac](#): 1
- unsigned [dot11ax](#): 1
- uint16_t [mdid](#)
- unsigned [ft_1x](#): 1
- unsigned [ft_psk](#): 1
- unsigned [ft_sae](#): 1
- uint16_t [beacon_period](#)
- uint8_t [dtim_period](#)
- uint8_t [wlan_capa](#)
- uint8_t [btm_mode](#)
- bool [bss_transition_supported](#)
- bool [neighbor_report_supported](#)

4.92.1 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.

4.92.2 Field Documentation

4.92.2.1 id

```
int wlan_network::id
```

Identifier for network profile

4.92.2.2 wps_network

```
int wlan_network::wps_network
```

4.92.2.3 name

```
char wlan_network::name[WLAN_NETWORK_NAME_MAX_LENGTH+1]
```

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

4.92.2.4 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.

4.92.2.5 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.

4.92.2.6 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.

4.92.2.7 sec_channel_offset

```
uint8_t wlan_network::sec_channel_offset
```

The secondary channel offset

4.92.2.8 acs_band

```
uint16_t wlan_network::acs_band
```

The ACS band if set channel to 0.

4.92.2.9 rssi

```
int wlan_network::rssi
```

RSSI

4.92.2.10 ht_capab

```
unsigned short wlan_network::ht_capab
```

HT capabilities

4.92.2.11 vht_capab

```
unsigned int wlan_network::vht_capab
```

VHT capabilities

4.92.2.12 vht_oper_chwidth

```
unsigned char wlan_network::vht_oper_chwidth
```

VHT bandwidth

4.92.2.13 he_oper_chwidth

```
unsigned char wlan_network::he_oper_chwidth
```

HE bandwidth

4.92.2.14 type

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

BSS type

4.92.2.15 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.

4.92.2.16 security

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

The network security configuration specified by struct [wlan_network_security](#) for the network.

4.92.2.17 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.

4.92.2.18 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.

4.92.2.19 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.

4.92.2.20 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.

4.92.2.21 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.

4.92.2.22 dot11n

```
unsigned wlan_network::dot11n
```

The network supports 802.11N. (For internal use only)

4.92.2.23 dot11ac

```
unsigned wlan_network::dot11ac
```

The network supports 802.11AC. (For internal use only)

4.92.2.24 dot11ax

```
unsigned wlan_network::dot11ax
```

The network supports 802.11AX. (For internal use only)

4.92.2.25 mdid

```
uint16_t wlan_network::mdid
```

Mobility Domain ID

4.92.2.26 ft_1x

```
unsigned wlan_network::ft_1x
```

The network uses FT 802.1x security (For internal use only)

4.92.2.27 ft_psk

```
unsigned wlan_network::ft_psk
```

The network uses FT PSK security (For internal use only)

4.92.2.28 ft_sae

```
unsigned wlan_network::ft_sae
```

The network uses FT SAE security (For internal use only)

4.92.2.29 beacon_period

```
uint16_t wlan_network::beacon_period
```

Beacon period of associated BSS

4.92.2.30 dtim_period

```
uint8_t wlan_network::dtim_period
```

DTIM period of associated BSS

4.92.2.31 wlan_capa

```
uint8_t wlan_network::wlan_capa
```

Wireless capabilities of uAP network 802.11n, 802.11ac or/and 802.11ax

4.92.2.32 btm_mode

```
uint8_t wlan_network::btm_mode
```

BTM mode

4.92.2.33 bss_transition_supported

```
bool wlan_network::bss_transition_supported
```

bss transition support (For internal use only)

4.92.2.34 neighbor_report_supported

```
bool wlan_network::neighbor_report_supported
```

Neighbor report support (For internal use only)

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

- [wlan.h](#)

4.93 wlan_network_security Struct Reference

Data Fields

- enum [wlan_security_type](#) type
- int [key_mgmt](#)
- struct [wlan_cipher](#) mcstCipher
- struct [wlan_cipher](#) ucstCipher
- unsigned [pkc](#): 1
- int [group_cipher](#)
- int [pairwise_cipher](#)
- int [group_mgmt_cipher](#)
- bool [is_pmf_required](#)
- char [psk](#) [WLAN_PSK_MAX_LENGTH]
- uint8_t [psk_len](#)
- char [password](#) [WLAN_PASSWORD_MAX_LENGTH+1]
- size_t [password_len](#)
- char * [sae_groups](#)
- uint8_t [pwe_derivation](#)
- uint8_t [transition_disable](#)
- char [pmk](#) [WLAN_PMK_LENGTH]
- bool [pmk_valid](#)
- bool [mfpc](#)
- bool [mfpr](#)
- unsigned [wpa3_sb](#): 1
- unsigned [wpa3_sb_192](#): 1
- unsigned [eap_ver](#): 1
- unsigned [peap_label](#): 1
- uint8_t [eap_crypto_binding](#)
- unsigned [eap_result_ind](#): 1
- unsigned char [tls_cipher](#)
- char [identity](#) [IDENTITY_MAX_LENGTH]
- char [anonymous_identity](#) [IDENTITY_MAX_LENGTH]
- char [eap_password](#) [PASSWORD_MAX_LENGTH]
- unsigned char * [ca_cert_data](#)

- size_t [ca_cert_len](#)
- unsigned char * [client_cert_data](#)
- size_t [client_cert_len](#)
- unsigned char * [client_key_data](#)
- size_t [client_key_len](#)
- char [client_key_passwd](#) [PASSWORD_MAX_LENGTH]
- char [ca_cert_hash](#) [HASH_MAX_LENGTH]
- char [domain_match](#) [DOMAIN_MATCH_MAX_LENGTH]
- char [domain_suffix_match](#) [DOMAIN_MATCH_MAX_LENGTH]
- unsigned char * [ca_cert2_data](#)
- size_t [ca_cert2_len](#)
- unsigned char * [client_cert2_data](#)
- size_t [client_cert2_len](#)
- unsigned char * [client_key2_data](#)
- size_t [client_key2_len](#)
- char [client_key2_passwd](#) [PASSWORD_MAX_LENGTH]
- unsigned char * [dh_data](#)
- size_t [dh_len](#)
- unsigned char * [server_cert_data](#)
- size_t [server_cert_len](#)
- unsigned char * [server_key_data](#)
- size_t [server_key_len](#)
- char [server_key_passwd](#) [PASSWORD_MAX_LENGTH]
- size_t [nusers](#)
- char [identities](#) [MAX_USERS][IDENTITY_MAX_LENGTH]
- char [passwords](#) [MAX_USERS][PASSWORD_MAX_LENGTH]
- char [pac_opaque_encr_key](#) [PAC_OPAQUE_ENCR_KEY_MAX_LENGTH]
- char [a_id](#) [A_ID_MAX_LENGTH]
- uint8_t [fast_prov](#)
- unsigned char * [dpp_connector](#)
- unsigned char * [dpp_c_sign_key](#)
- unsigned char * [dpp_net_access_key](#)

4.93.1 Detailed Description

Network security configuration

4.93.2 Field Documentation

4.93.2.1 type

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

Type of network security to use specified by enum wlan_security_type.

4.93.2.2 key_mgmt

```
int wlan_network_security::key_mgmt
```

Key management type

4.93.2.3 mcstCipher

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

Type of network security Group Cipher suite used internally

4.93.2.4 ucstCipher

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

Type of network security Pairwise Cipher suite used internally

4.93.2.5 pkc

```
unsigned wlan_network_security::pkc
```

Proactive Key Caching

4.93.2.6 group_cipher

```
int wlan_network_security::group_cipher
```

Type of network security Group Cipher suite

4.93.2.7 pairwise_cipher

```
int wlan_network_security::pairwise_cipher
```

Type of network security Pairwise Cipher suite

4.93.2.8 group_mgmt_cipher

```
int wlan_network_security::group_mgmt_cipher
```

Type of network security Pairwise Cipher suite

4.93.2.9 is_pmf_required

```
bool wlan_network_security::is_pmf_required
```

Is PMF required

4.93.2.10 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.

4.93.2.11 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.

4.93.2.12 password

```
char wlan_network_security::password[WLAN_PASSWORD_MAX_LENGTH+1]
```

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.

4.93.2.13 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.

4.93.2.14 sae_groups

```
char* wlan_network_security::sae_groups
```

SAE Groups

4.93.2.15 pwe_derivation

```
uint8_t wlan_network_security::pwe_derivation
```

PWE derivation

4.93.2.16 transition_disable

```
uint8_t wlan_network_security::transition_disable
```

transition disable

4.93.2.17 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.

4.93.2.18 pmk_valid

```
bool wlan_network_security::pmk_valid
```

Flag reporting whether pmk is valid or not.

4.93.2.19 mfpc

```
bool wlan_network_security::mfpc
```

Management Frame Protection Capable (MFPC)

4.93.2.20 mfpr

```
bool wlan_network_security::mfpr
```

Management Frame Protection Required (MFPR)

4.93.2.21 wpa3_sb

```
unsigned wlan_network_security::wpa3_sb
```

WPA3 Enterprise mode

4.93.2.22 wpa3_sb_192

```
unsigned wlan_network_security::wpa3_sb_192
```

WPA3 Enterprise Suite B 192 mode

4.93.2.23 eap_ver

```
unsigned wlan_network_security::eap_ver
```

PEAP version

4.93.2.24 peap_label

```
unsigned wlan_network_security::peap_label
```

PEAP label

4.93.2.25 eap_crypto_binding

```
uint8_t wlan_network_security::eap_crypto_binding
```

crypto_binding option can be used to control [WLAN_SECURITY_EAP_PEAP_MSCHAPV2](#), [WLAN_SECURITY_EAP_PEAP_TLS](#) and [WLAN_SECURITY_EAP_PEAP_GTC](#) version 0 cryptobinding behavior: 0 = do not use cryptobinding (default) 1 = use cryptobinding if server supports it 2 = require cryptobinding

4.93.2.26 eap_result_ind

```
unsigned wlan_network_security::eap_result_ind
```

eap_result_ind=1 can be used to enable [WLAN_SECURITY_EAP_SIM](#), [WLAN_SECURITY_EAP_AKA](#) and [WLAN_SECURITY_EAP_AKA_PRIME](#) to use protected result indication.

4.93.2.27 tls_cipher

```
unsigned char wlan_network_security::tls_cipher
```

Cipher for EAP TLS

4.93.2.28 identity

```
char wlan_network_security::identity[IDENTITY_MAX_LENGTH]
```

Identity string for EAP

4.93.2.29 anonymous_identity

```
char wlan_network_security::anonymous_identity[IDENTITY_MAX_LENGTH]
```

Anonymous identity string for EAP

4.93.2.30 eap_password

```
char wlan_network_security::eap_password[PASSWORD_MAX_LENGTH]
```

Password string for EAP. This field can include either the plaintext password (using ASCII or hex string)

4.93.2.31 ca_cert_data

```
unsigned char* wlan_network_security::ca_cert_data
```

CA cert blob in PEM/DER format

4.93.2.32 ca_cert_len

```
size_t wlan_network_security::ca_cert_len
```

CA cert blob len

4.93.2.33 client_cert_data

```
unsigned char* wlan_network_security::client_cert_data
```

Client cert blob in PEM/DER format

4.93.2.34 client_cert_len

```
size_t wlan_network_security::client_cert_len
```

Client cert blob len

4.93.2.35 client_key_data

```
unsigned char* wlan_network_security::client_key_data
```

Client key blob

4.93.2.36 client_key_len

```
size_t wlan_network_security::client_key_len
```

Client key blob len

4.93.2.37 client_key_passwd

```
char wlan_network_security::client_key_passwd[PASSWORD_MAX_LENGTH]
```

Client key password

4.93.2.38 ca_cert_hash

```
char wlan_network_security::ca_cert_hash[HASH_MAX_LENGTH]
```

CA cert HASH

4.93.2.39 domain_match

```
char wlan_network_security::domain_match[DOMAIN_MATCH_MAX_LENGTH]
```

Domain

4.93.2.40 domain_suffix_match

```
char wlan_network_security::domain_suffix_match[DOMAIN_MATCH_MAX_LENGTH]
```

Domain Suffix

4.93.2.41 ca_cert2_data

```
unsigned char* wlan_network_security::ca_cert2_data
```

CA cert2 blob in PEM/DER format

4.93.2.42 ca_cert2_len

```
size_t wlan_network_security::ca_cert2_len
```

CA cert2 blob len

4.93.2.43 client_cert2_data

```
unsigned char* wlan_network_security::client_cert2_data
```

Client cert2 blob in PEM/DER format

4.93.2.44 client_cert2_len

```
size_t wlan_network_security::client_cert2_len
```

Client cert2 blob len

4.93.2.45 client_key2_data

```
unsigned char* wlan_network_security::client_key2_data
```

Client key2 blob

4.93.2.46 client_key2_len

```
size_t wlan_network_security::client_key2_len
```

Client key2 blob len

4.93.2.47 client_key2_passwd

```
char wlan_network_security::client_key2_passwd[PASSWORD_MAX_LENGTH]
```

Client key2 password

4.93.2.48 dh_data

```
unsigned char* wlan_network_security::dh_data
```

DH params blob

4.93.2.49 dh_len

```
size_t wlan_network_security::dh_len
```

DH params blob len

4.93.2.50 server_cert_data

```
unsigned char* wlan_network_security::server_cert_data
```

Server cert blob in PEM/DER format

4.93.2.51 server_cert_len

```
size_t wlan_network_security::server_cert_len
```

Server cert blob len

4.93.2.52 server_key_data

```
unsigned char* wlan_network_security::server_key_data
```

Server key blob

4.93.2.53 server_key_len

```
size_t wlan_network_security::server_key_len
```

Server key blob len

4.93.2.54 server_key_passwd

```
char wlan_network_security::server_key_passwd[PASSWORD_MAX_LENGTH]
```

Server key password

4.93.2.55 nusers

```
size_t wlan_network_security::nusers
```

Number of EAP users

4.93.2.56 identities

```
char wlan_network_security::identities[MAX_USERS][IDENTITY_MAX_LENGTH]
```

User Identities

4.93.2.57 passwords

```
char wlan_network_security::passwords[MAX_USERS][PASSWORD_MAX_LENGTH]
```

User Passwords

4.93.2.58 pac_opaque_encr_key

```
char wlan_network_security::pac_opaque_encr_key[PAC_OPAQUE_ENCR_KEY_MAX_LENGTH]
```

Encryption key for EAP-FAST PAC-Opaque values

4.93.2.59 a_id

```
char wlan_network_security::a_id[A_ID_MAX_LENGTH]
```

EAP-FAST authority identity (A-ID)

4.93.2.60 fast_prov

```
uint8_t wlan_network_security::fast_prov
```

EAP-FAST provisioning modes: 0 = provisioning disabled 1 = only anonymous provisioning allowed 2 = only authenticated provisioning allowed 3 = both provisioning modes allowed (default)

4.93.2.61 dpp_connector

```
unsigned char* wlan_network_security::dpp_connector
```

4.93.2.62 dpp_c_sign_key

```
unsigned char* wlan_network_security::dpp_c_sign_key
```

4.93.2.63 dpp_net_access_key

```
unsigned char* wlan_network_security::dpp_net_access_key
```

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

- [wlan.h](#)

4.94 wlan_nlist_report_param Struct Reference

Data Fields

- enum [wlan_nlist_mode](#) [nlist_mode](#)
- [t_u8](#) [num_channels](#)
- [t_u8](#) [channels](#) [[MAX_NUM_CHANS_IN_NBOR_RPT](#)]
- [t_u8](#) [btm_mode](#)
- [t_u8](#) [bssid](#) [[MLAN_MAC_ADDR_LENGTH](#)]
- [t_u8](#) [dialog_token](#)
- [t_u8](#) [dst_addr](#) [[MLAN_MAC_ADDR_LENGTH](#)]
- [t_u8](#) [protect](#)

4.94.1 Field Documentation

4.94.1.1 nlist_mode

```
enum wlan\_nlist\_mode wlan_nlist_report_param::nlist_mode
```

4.94.1.2 num_channels

```
t\_u8 wlan_nlist_report_param::num_channels
```

4.94.1.3 channels

t_u8 wlan_nlist_report_param::channels[[MAX_NUM_CHANS_IN_NBOR_RPT](#)]

4.94.1.4 btm_mode

t_u8 wlan_nlist_report_param::btm_mode

4.94.1.5 bssid

t_u8 wlan_nlist_report_param::bssid[[MLAN_MAC_ADDR_LENGTH](#)]

4.94.1.6 dialog_token

t_u8 wlan_nlist_report_param::dialog_token

4.94.1.7 dst_addr

t_u8 wlan_nlist_report_param::dst_addr[[MLAN_MAC_ADDR_LENGTH](#)]

4.94.1.8 protect

t_u8 wlan_nlist_report_param::protect

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

- [wifi.h](#)

4.95 wlan_rrm_beacon_report_data Struct Reference

Data Fields

- t_u8 [token](#)
- t_u8 [ssid](#) [[MLAN_MAX_SSID_LENGTH](#)]
- t_u8 [ssid_length](#)
- t_u8 [bssid](#) [[MLAN_MAC_ADDR_LENGTH](#)]
- t_u8 [channel](#) [[MAX_CHANNEL_LIST](#)]
- t_u8 [channel_num](#)
- t_u8 [last_ind](#)
- t_u16 [duration](#)
- enum [wlan_rrm_beacon_reporting_detail](#) [report_detail](#)
- t_u8 [bits_field](#) [32]

4.95.1 Field Documentation

4.95.1.1 token

t_u8 wlan_rrm_beacon_report_data::token

4.95.1.2 ssid

t_u8 wlan_rrm_beacon_report_data::ssid[MLAN_MAX_SSID_LENGTH]

4.95.1.3 ssid_length

t_u8 wlan_rrm_beacon_report_data::ssid_length

4.95.1.4 bssid

t_u8 wlan_rrm_beacon_report_data::bssid[MLAN_MAC_ADDR_LENGTH]

4.95.1.5 channel

t_u8 wlan_rrm_beacon_report_data::channel[MAX_CHANNEL_LIST]

4.95.1.6 channel_num

t_u8 wlan_rrm_beacon_report_data::channel_num

4.95.1.7 last_ind

t_u8 wlan_rrm_beacon_report_data::last_ind

4.95.1.8 duration

```
t_u16 wlan_rrm_beacon_report_data::duration
```

4.95.1.9 report_detail

```
enum wlan_rrm_beacon_reporting_detail wlan_rrm_beacon_report_data::report_detail
```

4.95.1.10 bits_field

```
t_u8 wlan_rrm_beacon_report_data::bits_field[32]
```

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

- [wifi.h](#)

4.96 wlan_rrm_neighbor_ap_t Struct Reference

Data Fields

- char [ssid](#) [[MLAN_MAX_SSID_LENGTH](#)]
- t_u8 [bssid](#) [[MLAN_MAX_SSID_LENGTH](#)]
- t_u8 [bssidInfo](#) [32]
- int [op_class](#)
- int [channel](#)
- int [phy_type](#)
- int [freq](#)

4.96.1 Field Documentation

4.96.1.1 ssid

```
char wlan_rrm_neighbor_ap_t::ssid[MLAN\_MAX\_SSID\_LENGTH]
```

4.96.1.2 bssid

```
t_u8 wlan_rrm_neighbor_ap_t::bssid[MLAN\_MAX\_SSID\_LENGTH]
```

4.96.1.3 bssidInfo

```
t_u8 wlan_rrm_neighbor_ap_t::bssidInfo[32]
```

4.96.1.4 op_class

```
int wlan_rrm_neighbor_ap_t::op_class
```

4.96.1.5 channel

```
int wlan_rrm_neighbor_ap_t::channel
```

4.96.1.6 phy_type

```
int wlan_rrm_neighbor_ap_t::phy_type
```

4.96.1.7 freq

```
int wlan_rrm_neighbor_ap_t::freq
```

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

- [wifi.h](#)

4.97 wlan_rrm_neighbor_report_t Struct Reference

Data Fields

- [wlan_rrm_neighbor_ap_t neighbor_ap](#) [MAX_NEIGHBOR_AP_LIMIT]
- int [neighbor_cnt](#)

4.97.1 Field Documentation

4.97.1.1 neighbor_ap

`wlan_rrm_neighbor_ap_t` `wlan_rrm_neighbor_report_t::neighbor_ap` [`MAX_NEIGHBOR_AP_LIMIT`]

4.97.1.2 neighbor_cnt

`int` `wlan_rrm_neighbor_report_t::neighbor_cnt`

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

- [wifi.h](#)

4.98 wlan_rrm_scan_cb_param Struct Reference

Data Fields

- [wlan_rrm_beacon_report_data](#) `rep_data`
- `t_u8` [dialog_tok](#)
- `t_u8` [dst_addr](#) [`MLAN_MAC_ADDR_LENGTH`]
- `t_u8` [protect](#)

4.98.1 Field Documentation

4.98.1.1 rep_data

`wlan_rrm_beacon_report_data` `wlan_rrm_scan_cb_param::rep_data`

4.98.1.2 dialog_tok

`t_u8` `wlan_rrm_scan_cb_param::dialog_tok`

4.98.1.3 dst_addr

`t_u8` `wlan_rrm_scan_cb_param::dst_addr` [`MLAN_MAC_ADDR_LENGTH`]

4.98.1.4 protect

```
t_u8 wlan_rrm_scan_cb_param::protect
```

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

- [wifi.h](#)

4.99 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 [dot11n](#): 1
- unsigned [dot11ac](#): 1
- unsigned [dot11ax](#): 1
- unsigned [wmm](#): 1
- unsigned [wps](#): 1
- unsigned int [wps_session](#)
- unsigned [wep](#): 1
- unsigned [wpa](#): 1
- unsigned [wpa2](#): 1
- unsigned [wpa2_sha256](#): 1
- unsigned [wpa3_sae](#): 1
- unsigned [wpa2_entp](#): 1
- unsigned [wpa2_entp_sha256](#): 1
- unsigned [wpa3_1x_sha256](#): 1
- unsigned [wpa3_1x_sha384](#): 1
- unsigned [ft_1x](#): 1
- unsigned [ft_1x_sha384](#): 1
- unsigned [ft_psk](#): 1
- unsigned [ft_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](#)
- t_u8 [ap_mfpc](#)
- t_u8 [ap_mfpr](#)
- t_u8 [ap_pwe](#)
- bool [neighbor_report_supported](#)
- bool [bss_transition_supported](#)

4.99.1 Detailed Description

Scan Result

4.99.2 Field Documentation

4.99.2.1 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.

4.99.2.2 ssid_len

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

SSID length

4.99.2.3 bssid

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

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

4.99.2.4 channel

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

The network channel.

4.99.2.5 type

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

The network wireless type.

4.99.2.6 role

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

The network wireless mode.

4.99.2.7 dot11n

```
unsigned wlan_scan_result::dot11n
```

The network supports 802.11N. This is set to 0 if the network does not support 802.11N or if the system does not have 802.11N support enabled.

4.99.2.8 dot11ac

```
unsigned wlan_scan_result::dot11ac
```

The network supports 802.11AC. This is set to 0 if the network does not support 802.11AC or if the system does not have 802.11AC support enabled.

4.99.2.9 dot11ax

```
unsigned wlan_scan_result::dot11ax
```

The network supports 802.11AX. This is set to 0 if the network does not support 802.11AX or if the system does not have 802.11AX support enabled.

4.99.2.10 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.

4.99.2.11 wps

```
unsigned wlan_scan_result::wps
```

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

4.99.2.12 wps_session

```
unsigned int wlan_scan_result::wps_session
```

WPS Type PBC/PIN

4.99.2.13 wep

```
unsigned wlan_scan_result::wep
```

The network uses WEP security.

4.99.2.14 wpa

```
unsigned wlan_scan_result::wpa
```

The network uses WPA security.

4.99.2.15 wpa2

```
unsigned wlan_scan_result::wpa2
```

The network uses WPA2 security

4.99.2.16 wpa2_sha256

```
unsigned wlan_scan_result::wpa2_sha256
```

The network uses WPA2 SHA256 security

4.99.2.17 wpa3_sae

```
unsigned wlan_scan_result::wpa3_sae
```

The network uses WPA3 SAE security

4.99.2.18 wpa2_entp

```
unsigned wlan_scan_result::wpa2_entp
```

The network uses WPA2 Enterprise security

4.99.2.19 wpa2_entp_sha256

```
unsigned wlan_scan_result::wpa2_entp_sha256
```

The network uses WPA2 Enterprise SHA256 security

4.99.2.20 wpa3_1x_sha256

```
unsigned wlan_scan_result::wpa3_1x_sha256
```

The network uses WPA3 Enterprise SHA256 security

4.99.2.21 wpa3_1x_sha384

```
unsigned wlan_scan_result::wpa3_1x_sha384
```

The network uses WPA3 Enterprise SHA384 security

4.99.2.22 ft_1x

```
unsigned wlan_scan_result::ft_1x
```

The network uses FT 802.1x security (For internal use only)

4.99.2.23 ft_1x_sha384

```
unsigned wlan_scan_result::ft_1x_sha384
```

The network uses FT 892.1x SHA384 security

4.99.2.24 ft_psk

```
unsigned wlan_scan_result::ft_psk
```

The network uses FT PSK security (For internal use only)

4.99.2.25 ft_sae

```
unsigned wlan_scan_result::ft_sae
```

The network uses FT SAE security (For internal use only)

4.99.2.26 rssi

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

The signal strength of the beacon

4.99.2.27 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.

4.99.2.28 trans_ssid_len

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

SSID length

4.99.2.29 trans_bssid

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

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

4.99.2.30 beacon_period

```
uint16_t wlan_scan_result::beacon_period
```

Beacon Period

4.99.2.31 dtim_period

```
uint8_t wlan_scan_result::dtim_period
```

DTIM Period

4.99.2.32 ap_mfpc

```
t_u8 wlan_scan_result::ap_mfpc
```

MFPC bit of AP

4.99.2.33 ap_mfpr

```
t_u8 wlan_scan_result::ap_mfpr
```

MFPR bit of AP

4.99.2.34 ap_pwe

```
t_u8 wlan_scan_result::ap_pwe
```

PWE bit of AP

4.99.2.35 neighbor_report_supported

```
bool wlan_scan_result::neighbor_report_supported
```

Neighbor report support (For internal use only)

4.99.2.36 bss_transition_supported

```
bool wlan_scan_result::bss_transition_supported
```

bss transition support (For internal use only)

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

- [wlan.h](#)

4.100 wnm_sleep_result_t Struct Reference

Data Fields

- [uint8_t action](#)
- [uint8_t result](#)

4.100.1 Field Documentation

4.100.1.1 action

```
uint8_t wnm_sleep_result_t::action
```

4.100.1.2 result

```
uint8_t wnm_sleep_result_t::result
```

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 lookup_command()

```
const struct cli_command* lookup_command (
    char * name,
    int len )
```

5.1.3.2 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

5.1.3.3 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

5.1.3.4 cli_init()

```
int cli_init (
    void )
```

Initialize the CLI module

Returns

WM_SUCCESS on success
error code otherwise.

5.1.3.5 cli_deinit()

```
int cli_deinit (
    void )
```

Deinitialize the CLI module

Returns

WM_SUCCESS on success
error code otherwise.

5.1.3.6 cli_stop()

```
int cli_stop (
    void )
```

Stop the CLI thread and carry out the cleanup

Returns

WM_SUCCESS on success
error code otherwise.

5.1.3.7 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

5.1.3.8 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

5.1.3.9 cli_get_cmd_buffer()

```
int cli_get_cmd_buffer (
    char ** buff )
```

Get a command buffer

If an external input task wants to use the CLI, it can use [cli_get_cmd_buffer\(\)](#) to get a command buffer that it can then submit to the CLI later using [cli_submit_cmd_buffer\(\)](#).

Parameters

<i>buff</i>	Pointer to a char * to place the buffer pointer in.
-------------	---

Returns

WM_SUCCESS on success
error code otherwise.

5.1.3.10 cli_submit_cmd_buffer()

```
int cli_submit_cmd_buffer (
    char ** buff )
```

Submit a command buffer to the CLI

Sends the command buffer to the CLI for processing.

Parameters

<i>buff</i>	Pointer to a char * buffer.
-------------	-----------------------------

Returns

WM_SUCCESS on success
error code otherwise.

5.1.3.11 cli_add_history_hook()

```
int cli_add_history_hook (
    cli_name_val_get get_cb,
    cli_name_val_set set_cb )
```

5.1.3.12 help_command()

```
void help_command (
    int argc,
    char ** argv )
```

5.1.4 Macro Documentation

5.1.4.1 CONFIG_APP_FRM_CLI_HISTORY

```
#define CONFIG_APP_FRM_CLI_HISTORY 1
```

5.1.5 Typedef Documentation

5.1.5.1 cli_name_val_get

```
typedef int(* cli_name_val_get) (const char *name, char *value, int max_len)
```

5.1.5.2 cli_name_val_set

```
typedef int(* cli_name_val_set) (const char *name, const char *value)
```

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 dhcpcd_cli_init()

```
int dhcpcd_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 dhcpcd_cli_deinit()

```
int dhcpcd_cli_deinit (  
    void )
```

Unregister DHCP server commands

This function unregisters 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.

5.2.2.3 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

5.2.2.4 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.
----	---------------------	--

5.2.2.5 dhcp_server_stop()

```
void dhcp_server_stop (
    void )
```

Stop DHCP server

5.2.2.6 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	val	Number of seconds, use (60U*60U*number of hours) for clarity. Max value is (60U*60U*24U*49700U)
----	-----	---

Returns

Error status code

5.2.2.7 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	client_mac	Pointer to a six byte array containing the MAC address of the client
out	client_ip	Pointer to IP address of the client

Returns

WM_SUCCESS on success or -WM_FAIL.

5.2.2.8 dhcp_stat()

```
void dhcp_stat (
    void )
```

Print DHCP stats on the console

This API prints DHCP stats on the console

5.2.3 Macro Documentation

5.2.3.1 MAX_QNAME_SIZE

```
#define MAX_QNAME_SIZE 32
```

5.2.4 Enumeration Type Documentation

5.2.4.1 wm_dhcpd_errno

```
enum wm_dhcpd_errno
```

DHCPD Error Codes

Enumerator

WM_E_DHCPD_ERRNO_BASE	
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
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

5.3.2 Macro Documentation

5.3.2.1 iperf_e

```
#define iperf_e(  
    ... ) wmlog_e("iperf", ##__VA_ARGS__)
```

5.3.2.2 iperf_w

```
#define iperf_w(
    ... ) wmlog_w("iperf", ##__VA_ARGS__)
```

5.4 osa.h File Reference

5.4.1 Function Documentation

5.4.1.1 OSA_TimerCreate()

```
osa_status_t OSA_TimerCreate (
    osa_timer_handle_t timerHandle,
    osa_timer_tick ticks,
    void(*) (osa_timer_arg_t) call_back,
    void * cb_arg,
    osa_timer_t reload,
    osa_timer_activate_t activate )
```

Create timer

This function creates a timer.

Parameters

in	<i>timerHandle</i>	Pointer to the timer handle
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 KOSA_TimerOnce or KOSA_TimerPeriodic.
in	<i>activate</i>	Activate Options, valid values include OSA_TIMER_AUTO_ACTIVATE or OSA_TIMER_NO_ACTIVATE

Returns

KOSA_StatusSuccess if timer created successfully
KOSA_StatusError if timer creation fails

5.4.1.2 OSA_TimerActivate()

```
osa_status_t OSA_TimerActivate (
    osa_timer_handle_t timerHandle )
```

Activate timer

This function activates (or starts) a timer that was previously created using [OSA_TimerCreate\(\)](#). If the timer had already started and was already in the active state, then this call is equivalent to [OSA_TimerReset\(\)](#).

Parameters

in	<i>timerHandle</i>	Pointer to a timer handle
----	--------------------	---------------------------

Returns

KOSA_StatusSuccess if timer activated successfully
 KOSA_StatusError if timer activation fails

5.4.1.3 OSA_TimerChange()

```
osa_status_t OSA_TimerChange (
    osa_timer_handle_t timerHandle,
    osa_timer_tick ntime,
    osa_timer_tick block_time )
```

Change timer period

This function changes the period of a timer that was previously created using [OSA_TimerCreate\(\)](#). This function changes the period of an active or dormant state timer.

Parameters

in	<i>timerHandle</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

KOSA_StatusSuccess if timer change successfully
 KOSA_StatusError if timer change fails

5.4.1.4 OSA_TimerIsRunning()

```
bool OSA_TimerIsRunning (
    osa_timer_handle_t timerHandle )
```

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>timerHandle</i>	Pointer to a timer handle
----	--------------------	---------------------------

Returns

true if timer is active
false if time is not active

5.4.1.5 OSA_TimerGetContext()

```
void* OSA_TimerGetContext (
    osa_timer_handle_t timerHandle )
```

Get the timer context

This function helps to retrieve the timer context i.e. 'cb_arg' passed to [OSA_TimerCreate\(\)](#).

Parameters

in	<i>timerHandle</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 [OSA_TimerCreate\(\)](#).

5.4.1.6 OSA_TimerReset()

```
osa_status_t OSA_TimerReset (
    osa_timer_handle_t timerHandle )
```

Reset timer

This function resets a timer that was previously created using [OSA_TimerCreate\(\)](#). 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 [OSA_TimerReset\(\)](#) was called. If the timer was in the dormant state then this call behaves in the same way as [OSA_TimerActivate\(\)](#).

Parameters

in	<i>timerHandle</i>	Pointer to a timer handle
----	--------------------	---------------------------

Returns

KOSA_StatusSuccess if timer reset successfully
KOSA_StatusError if timer reset fails

5.4.1.7 OSA_TimerDeactivate()

```
osa_status_t OSA_TimerDeactivate (
    osa_timer_handle_t timerHandle )
```

Deactivate timer

This function deactivates (or stops) a timer that was previously started.

Parameters

in	<i>timerHandle</i>	handle populated by OSA_TimerCreate() .
----	--------------------	---

Returns

KOSA_StatusSuccess if timer deactivate successfully
KOSA_StatusError if timer deactivate fails

5.4.1.8 OSA_TimerDestroy()

```
osa_status_t OSA_TimerDestroy (
    osa_timer_handle_t timerHandle )
```

Destroy timer

This function deletes a timer.

Parameters

in	<i>timerHandle</i>	Pointer to a timer handle
----	--------------------	---------------------------

Returns

KOSA_StatusSuccess if timer destroy successfully
KOSA_StatusError if timer destroy fails

5.4.1.9 OSA_RWLockCreateWithCB()

```
int OSA_RWLockCreateWithCB (
    osa_rw_lock_t * plock,
    const char * mutex_name,
    const char * lock_name,
    cb_fn r_fn )
```

5.4.1.10 OSA_RWLockCreate()

```
int OSA_RWLockCreate (
    osa_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

5.4.1.11 OSA_RWLockDestroy()

```
void OSA_RWLockDestroy (
    osa_rw_lock_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
----	-------------	--

5.4.1.12 OSA_RWLockWriteLock()

```
int OSA_RWLockWriteLock (
    osa_rw_lock_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 special values <code>osaWaitForever_c</code> and <code>osaWaitNone_c</code> are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS on success
 -WM_FAIL on error

5.4.1.13 OSA_RWLockWriteUnlock()

```
void OSA_RWLockWriteUnlock (
    osa_rw_lock_t * lock )
```

Release writer lock

This function releases a writer lock previously acquired using [OSA_RWLockWriteLock\(\)](#).

Parameters

in	<i>lock</i>	Pointer to the reader-writer lock handle
----	-------------	--

5.4.1.14 OSA_RWLockReadLock()

```
int OSA_RWLockReadLock (
    osa_rw_lock_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 special values <code>osaWaitForever_c</code> and <code>osaWaitNone_c</code> are provided to respectively wait infinitely or return immediately.

Returns

WM_SUCCESS on success
-WM_FAIL on error

5.4.1.15 OSA_RWLockReadUnlock()

```
int OSA_RWLockReadUnlock (
    osa_rw_lock_t * lock )
```

Release reader lock

This function releases a reader lock previously acquired using [OSA_RWLockReadLock\(\)](#).

Parameters

in	lock	pointer to the reader-writer lock handle
----	------	--

Returns

WM_SUCCESS if unlock operation successful.
-WM_FAIL if unlock operation failed.

5.4.1.16 OSA_SetupIdleFunction()

```
int OSA_SetupIdleFunction (
    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	func	The callback function
----	------	-----------------------

Returns

WM_SUCCESS on success
-WM_FAIL on error

5.4.1.17 OSA_SetupTickFunction()

```
int OSA_SetupTickFunction (
    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

5.4.1.18 OSA_RemoveIdleFunction()

```
int OSA_RemoveIdleFunction (
    void(*) (void) func )
```

Remove idle function

This function removes an idle callback function that was registered previously using [OSA_SetupIdleFunction\(\)](#).

Parameters

in	<i>func</i>	The callback function
----	-------------	-----------------------

Returns

WM_SUCCESS on success
-WM_FAIL on error

5.4.1.19 OSA_RemoveTickFunction()

```
int OSA_RemoveTickFunction (
    void(*) (void) func )
```

Remove tick function

This function removes a tick callback function that was registered previously using [OSA_SetupTickFunction\(\)](#).

Parameters

in	<i>func</i>	Callback function
----	-------------	-------------------

Returns

WM_SUCCESS on success
-WM_FAIL on error

5.4.1.20 OSA_Srand()

```
static void OSA_Srand (  
    uint32_t seed ) [inline], [static]
```

This function initialize the seed for rand generator

Returns

a uint32_t random numer

5.4.1.21 OSA_Rand()

```
static uint32_t OSA_Rand ( ) [inline], [static]
```

This function generate a random number

Returns

a uint32_t random numer

5.4.1.22 OSA_RandRange()

```
static uint32_t OSA_RandRange (  
    uint32_t low,  
    uint32_t high ) [inline], [static]
```

This function generate a random number in a range

Parameters

in	<i>low</i>	range low
in	<i>high</i>	range high

Returns

a uint32_t random numer

5.4.1.23 OSA_DumpThreadInfo()

```
void OSA_DumpThreadInfo (
    char * name )
```

5.4.1.24 OSA_ThreadSelfComplete()

```
void OSA_ThreadSelfComplete (
    osa_task_handle_t taskHandle )
```

Suspend the given thread

- The function [OSA_ThreadSelfComplete\(\)](#) 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 [OSA_TaskDestroy\(\)](#) needs to be called separately.

Parameters

in	<i>taskHandle</i>	Pointer to thread handle
----	-------------------	--------------------------

5.4.1.25 OSA_MsgQWaiting()

```
uint32_t OSA_MsgQWaiting (
    osa_msgq_handle_t msgqHandle )
```

Return the number of messages stored in queue.

Parameters

in	<i>msgqHandle</i>	Pointer to handle of the queue to be queried.
----	-------------------	---

Returns

Number of items in the queue

5.4.2 Macro Documentation

5.4.2.1 MAX_CUSTOM_HOOKS

```
#define MAX_CUSTOM_HOOKS 4U
```

5.4.3 Typedef Documentation

5.4.3.1 cb_fn

```
typedef int(* cb_fn) (osa_rw_lock_t *plock, unsigned int wait_time)
```

This is prototype of reader callback

5.4.4 Variable Documentation

5.4.4.1 g_osa_tick_hooks

```
void(* g_osa_tick_hooks[MAX_CUSTOM_HOOKS]) (void)
```

5.4.4.2 g_osa_idle_hooks

```
void(* g_osa_idle_hooks[MAX_CUSTOM_HOOKS]) (void)
```

5.4.4.3 wm_rand_seed

```
uint32_t wm_rand_seed
```

5.5 README.txt File Reference

5.6 wifi-decl.h File Reference

Wifi structure declarations.

5.6.1 Macro Documentation

5.6.1.1 MLAN_MAC_ADDR_LENGTH

```
#define MLAN_MAC_ADDR_LENGTH (6U)
```

5.6.1.2 MLAN_MAX_VER_STR_LEN

```
#define MLAN_MAX_VER_STR_LEN 128
```

Version string buffer length

5.6.1.3 WIFI_MAX_CHANNEL_NUM

```
#define WIFI_MAX_CHANNEL_NUM 42
```

5.6.1.4 PMK_BIN_LEN

```
#define PMK_BIN_LEN 32
```

5.6.1.5 PMK_HEX_LEN

```
#define PMK_HEX_LEN 64
```

5.6.1.6 MOD_GROUPS

```
#define MOD_GROUPS 7
```

5.6.1.7 WIFI_SUPPORT_11AX

```
#define WIFI_SUPPORT_11AX (1 << 3)
```

5.6.1.8 WIFI_SUPPORT_11AC

```
#define WIFI_SUPPORT_11AC (1 << 2)
```

5.6.1.9 WIFI_SUPPORT_11N

```
#define WIFI_SUPPORT_11N (1 << 1)
```

5.6.1.10 WIFI_SUPPORT_LEGACY

```
#define WIFI_SUPPORT_LEGACY (1 << 0)
```

5.6.1.11 BSS_TYPE_STA

```
#define BSS_TYPE_STA 0U
```

BSS type : STA

5.6.1.12 BSS_TYPE_UAP

```
#define BSS_TYPE_UAP 1U
```

BSS type : UAP

5.6.1.13 UAP_DEFAULT_CHANNEL

```
#define UAP_DEFAULT_CHANNEL 0
```

5.6.1.14 UAP_DEFAULT_BANDWIDTH

```
#define UAP_DEFAULT_BANDWIDTH 2
```

5.6.1.15 UAP_DEFAULT_BEACON_PERIOD

```
#define UAP_DEFAULT_BEACON_PERIOD 100
```

5.6.1.16 UAP_DEFAULT_HIDDEN_SSID

```
#define UAP_DEFAULT_HIDDEN_SSID 0
```

5.6.1.17 MLAN_MAX_SSID_LENGTH

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

MLAN Maximum SSID Length

5.6.1.18 MLAN_MAX_PASS_LENGTH

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

MLAN Maximum PASSPHRASE Length

5.6.1.19 BIT

```
#define BIT(  
    n ) (1U << (n))
```

5.6.1.20 WOWLAN_MAX_PATTERN_LEN

```
#define WOWLAN_MAX_PATTERN_LEN 20
```

5.6.1.21 WOWLAN_MAX_OFFSET_LEN

```
#define WOWLAN_MAX_OFFSET_LEN 50
```

5.6.1.22 MAX_NUM_FILTERS

```
#define MAX_NUM_FILTERS 10
```

5.6.1.23 MEF_MODE_HOST_SLEEP

```
#define MEF_MODE_HOST_SLEEP (1 << 0)
```


5.6.1.24 MEF_MODE_NON_HOST_SLEEP

```
#define MEF_MODE_NON_HOST_SLEEP (1 << 1)
```

5.6.1.25 MEF_ACTION_WAKE

```
#define MEF_ACTION_WAKE (1 << 0)
```

5.6.1.26 MEF_ACTION_ALLOW

```
#define MEF_ACTION_ALLOW (1 << 1)
```

5.6.1.27 MEF_ACTION_ALLOW_AND_WAKEUP_HOST

```
#define MEF_ACTION_ALLOW_AND_WAKEUP_HOST 3
```

5.6.1.28 MEF_AUTO_ARP

```
#define MEF_AUTO_ARP 0x10
```

5.6.1.29 MEF_AUTO_PING

```
#define MEF_AUTO_PING 0x20
```

5.6.1.30 MEF_NS_RESP

```
#define MEF_NS_RESP 0x40
```

5.6.1.31 MEF_MAGIC_PKT

```
#define MEF_MAGIC_PKT 0x80
```

5.6.1.32 CRITERIA_BROADCAST

```
#define CRITERIA_BROADCAST MBIT(0)
```

5.6.1.33 CRITERIA_UNICAST

```
#define CRITERIA_UNICAST MBIT(1)
```

5.6.1.34 CRITERIA_MULTICAST

```
#define CRITERIA_MULTICAST MBIT(3)
```

5.6.1.35 MAX_NUM_ENTRIES

```
#define MAX_NUM_ENTRIES 8
```

5.6.1.36 MAX_NUM_BYTE_SEQ

```
#define MAX_NUM_BYTE_SEQ 6
```

5.6.1.37 MAX_NUM_MASK_SEQ

```
#define MAX_NUM_MASK_SEQ 6
```

5.6.1.38 OPERAND_DNUM

```
#define OPERAND_DNUM 1
```

5.6.1.39 OPERAND_BYTE_SEQ

```
#define OPERAND_BYTE_SEQ 2
```

5.6.1.40 MAX_OPERAND

```
#define MAX_OPERAND 0x40
```

5.6.1.41 TYPE_BYTE_EQ

```
#define TYPE_BYTE_EQ (MAX_OPERAND + 1)
```

5.6.1.42 TYPE_DNUM_EQ

```
#define TYPE_DNUM_EQ (MAX_OPERAND + 2)
```

5.6.1.43 TYPE_BIT_EQ

```
#define TYPE_BIT_EQ (MAX_OPERAND + 3)
```

5.6.1.44 RPN_TYPE_AND

```
#define RPN_TYPE_AND (MAX_OPERAND + 4)
```

5.6.1.45 RPN_TYPE_OR

```
#define RPN_TYPE_OR (MAX_OPERAND + 5)
```

5.6.1.46 ICMP_OF_IP_PROTOCOL

```
#define ICMP_OF_IP_PROTOCOL 0x01
```

5.6.1.47 TCP_OF_IP_PROTOCOL

```
#define TCP_OF_IP_PROTOCOL 0x06
```

5.6.1.48 UDP_OF_IP_PROTOCOL

```
#define UDP_OF_IP_PROTOCOL 0x11
```

5.6.1.49 IPV4_PKT_OFFSET

```
#define IPV4_PKT_OFFSET 20
```

5.6.1.50 IP_PROTOCOL_OFFSET

```
#define IP_PROTOCOL_OFFSET 31
```

5.6.1.51 PORT_PROTOCOL_OFFSET

```
#define PORT_PROTOCOL_OFFSET 44
```

5.6.1.52 FILLING_TYPE

```
#define FILLING_TYPE MBIT(0)
```

5.6.1.53 FILLING_PATTERN

```
#define FILLING_PATTERN MBIT(1)
```

5.6.1.54 FILLING_OFFSET

```
#define FILLING_OFFSET MBIT(2)
```

5.6.1.55 FILLING_NUM_BYTES

```
#define FILLING_NUM_BYTES MBIT(3)
```

5.6.1.56 FILLING_REPEAT

```
#define FILLING_REPEAT MBIT(4)
```

5.6.1.57 FILLING_BYTE_SEQ

```
#define FILLING_BYTE_SEQ MBIT(5)
```

5.6.1.58 FILLING_MASK_SEQ

```
#define FILLING_MASK_SEQ MBIT(6)
```

5.6.1.59 MKEEP_ALIVE_IP_PKT_MAX

```
#define MKEEP_ALIVE_IP_PKT_MAX 256
```

5.6.1.60 WLAN_BTWT_REPORT_LEN

```
#define WLAN_BTWT_REPORT_LEN 9
```

5.6.1.61 WLAN_BTWT_REPORT_MAX_NUM

```
#define WLAN_BTWT_REPORT_MAX_NUM 4
```

5.6.1.62 BAND_SPECIFIED

```
#define BAND_SPECIFIED 0x80U
```

Scan all the channels in specified band

5.6.1.63 MAX_CHANNEL_LIST

```
#define MAX_CHANNEL_LIST 6
```

5.6.1.64 MAX_NUM_SSID

```
#define MAX_NUM_SSID 2
```

5.6.1.65 MAX_FUNC_SYMBOL_LEN

```
#define MAX_FUNC_SYMBOL_LEN 64
```

5.6.1.66 OS_MEM_STAT_TABLE_SIZE

```
#define OS_MEM_STAT_TABLE_SIZE 128
```

5.6.1.67 CSI_FILTER_MAX

```
#define CSI_FILTER_MAX 16
```

5.6.2 Enumeration Type Documentation

5.6.2.1 wifi_bss_security

```
enum wifi_bss_security
```

Enumerator

WIFI_SECURITY_NONE	
WIFI_SECURITY_WEP_STATIC	
WIFI_SECURITY_WEP_DYNAMIC	
WIFI_SECURITY_WPA	
WIFI_SECURITY_WPA2	

5.6.2.2 wifi_bss_features

```
enum wifi_bss_features
```

Enumerator

WIFI_BSS_FEATURE_WMM	
WIFI_BSS_FEATURE_WPS	

5.6.2.3 wlan_type

```
enum wlan_type
```

Enumerator

WLAN_TYPE_NORMAL	
WLAN_TYPE_WIFI_CALIB	
WLAN_TYPE_FCC_CERTIFICATION	

5.6.2.4 wifi_ds_command_type

```
enum wifi_ds_command_type
```

Enumerator

WIFI_DS_RATE_CFG	
WIFI_DS_GET_DATA_RATE	

5.6.2.5 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

5.6.2.6 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

5.7 wifi.h File Reference

This file contains interface to wifi driver.

5.7.1 Function Documentation

5.7.1.1 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.

5.7.1.2 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.

5.7.1.3 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.

5.7.1.4 wifi_set_tx_status()

```
void wifi_set_tx_status (
    t_u8 status )
```

This API can be used to set wifi driver tx status.

Parameters

in	<i>status</i>	Status to set for TX
----	---------------	----------------------

5.7.1.5 wifi_set_rx_status()

```
void wifi_set_rx_status (
    t_u8 status )
```

This API can be used to set wifi driver rx status.

Parameters

in	<i>status</i>	Status to set for RX
----	---------------	----------------------

5.7.1.6 reset_ie_index()

```
void reset_ie_index ( )
```

This API can be used to reset mgmt_ie_index_bitmap.

5.7.1.7 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

5.7.1.8 wifi_deregister_data_input_callback()

```
void wifi_deregister_data_input_callback (
    void )
```

Deregister Data callback function from Wi-Fi Driver

5.7.1.9 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_SUCESS

5.7.1.10 wifi_deregister_amsdu_data_input_callback()

```
void wifi_deregister_amsdu_data_input_callback (
    void )
```

Deregister Data callback function from Wi-Fi Driver

5.7.1.11 wifi_register_deliver_packet_above_callback()

```
int wifi_register_deliver_packet_above_callback (
    void(*) (void *rxpd, uint8_t interface, void *lwip_pbuf) deliver_packet_above_↵
callback )
```

5.7.1.12 wifi_deregister_deliver_packet_above_callback()

```
void wifi_deregister_deliver_packet_above_callback (
    void )
```

5.7.1.13 wifi_register_wrapper_net_is_ip_or_ipv6_callback()

```
int wifi_register_wrapper_net_is_ip_or_ipv6_callback (
    bool(*) (const t_u8 *buffer) wrapper_net_is_ip_or_ipv6_callback )
```

5.7.1.14 wifi_deregister_wrapper_net_is_ip_or_ipv6_callback()

```
void wifi_deregister_wrapper_net_is_ip_or_ipv6_callback (
    void )
```

5.7.1.15 wifi_add_to_bypassq()

```
int wifi_add_to_bypassq (
    const t_u8 interface,
    void * pkt,
    t_u32 len )
```

5.7.1.16 wifi_low_level_output()

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

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.
in	<i>pkt_prio</i>	Priority for sending packet.
in	<i>tid</i>	TID for tx.

Returns

WM_SUCCESS on success or -WM_E_NOMEM if memory is not available or -WM_E_BUSY if SDIO is busy.

5.7.1.17 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.
----	--------------	-----------------------

5.7.1.18 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.

5.7.1.19 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.

5.7.1.20 wifi_sta_ampdu_tx_enable_per_tid()

```
void wifi_sta_ampdu_tx_enable_per_tid (
    t_u8 tid )
```

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

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

5.7.1.21 wifi_sta_ampdu_tx_enable_per_tid_is_allowed()

```
t_u8 wifi_sta_ampdu_tx_enable_per_tid_is_allowed (  
    t_u8 tid )
```

This API can be used to check if tid to enable AMPDU is allowed when station is a transmitter.

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

Returns

MTRUE or MFALSE

5.7.1.22 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.

5.7.1.23 wifi_sta_ampdu_rx_enable_per_tid()

```
void wifi_sta_ampdu_rx_enable_per_tid (  
    t_u8 tid )
```

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

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

5.7.1.24 wifi_sta_ampdu_rx_enable_per_tid_is_allowed()

```
t_u8 wifi_sta_ampdu_rx_enable_per_tid_is_allowed (  
    t_u8 tid )
```

This API can be used to check if tid to enable AMPDU is allowed when station is a receiver.

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

Returns

MTRUE or MFALSE

5.7.1.25 wifi_uap_ampdu_rx_enable()

```
void wifi_uap_ampdu_rx_enable (  
    void )
```

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

5.7.1.26 wifi_uap_ampdu_rx_enable_per_tid()

```
void wifi_uap_ampdu_rx_enable_per_tid (  
    t_u8 tid )
```

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

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

5.7.1.27 wifi_uap_ampdu_rx_enable_per_tid_is_allowed()

```
t_u8 wifi_uap_ampdu_rx_enable_per_tid_is_allowed (  
    t_u8 tid )
```

This API can be used to check if tid to enable AMPDU is allowed when uap is a receiver.

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

Returns

MTRUE or MFALSE

5.7.1.28 wifi_uap_ampdu_rx_disable()

```
void wifi_uap_ampdu_rx_disable (  
    void )
```

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

5.7.1.29 wifi_uap_ampdu_tx_enable()

```
void wifi_uap_ampdu_tx_enable (  
    void )
```

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

5.7.1.30 wifi_uap_ampdu_tx_enable_per_tid()

```
void wifi_uap_ampdu_tx_enable_per_tid (  
    t_u8 tid )
```

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

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

5.7.1.31 wifi_uap_ampdu_tx_enable_per_tid_is_allowed()

```
t_u8 wifi_uap_ampdu_tx_enable_per_tid_is_allowed (  
    t_u8 tid )
```

This API can be used to check if tid to enable AMPDU is allowed when uap is a transmitter.

Parameters

in	<i>tid</i>	tid value
----	------------	-----------

Returns

MTRUE or MFALSE

5.7.1.32 wifi_uap_ampdu_tx_disable()

```
void wifi_uap_ampdu_tx_disable (  
    void )
```

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

5.7.1.33 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.

5.7.1.34 wifi_get_device_mac_addr()

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

Get the device sta MAC address

Parameters

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

Returns

WM_SUCESS

5.7.1.35 wifi_get_device_uap_mac_addr()

```
int wifi_get_device_uap_mac_addr (
    wifi_mac_addr_t * mac_addr_uap )
```

Get the device uap MAC address

Parameters

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

Returns

WM_SUCESS

5.7.1.36 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

5.7.1.37 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

5.7.1.38 wifi_get_value1()

```
uint32_t wifi_get_value1 (  
    void )
```

5.7.1.39 wifi_get_outbuf()

```
uint8_t* wifi_get_outbuf (  
    uint32_t * outbuf_len )
```

5.7.1.40 wifi_config_roaming()

```
int wifi_config_roaming (  
    const int enable,  
    uint8_t * rssi_low )
```

5.7.1.41 wifi_config_bgscan_and_rssi()

```
int wifi_config_bgscan_and_rssi (  
    const char * ssid )
```

5.7.1.42 wifi_stop_bgscan()

```
mlan_status wifi_stop_bgscan ( )
```

5.7.1.43 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.

5.7.1.44 wifi_register_event_queue()

```
int wifi_register_event_queue (
    osa_msgq_handle_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

5.7.1.45 wifi_unregister_event_queue()

```
int wifi_unregister_event_queue (
    osa_msgq_handle_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

5.7.1.46 `wifi_get_scan_result()`

```
int wifi_get_scan_result (
    unsigned int index,
    struct wifi_scan_result2 ** desc )
```

Get scan list

Parameters

in	<i>index</i>	Index
out	<i>desc</i>	Descriptor of type <code>wifi_scan_result2</code>

Returns

WM_SUCCESS on success or error code.

5.7.1.47 `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	<i>count</i>	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.

5.7.1.48 `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

<i>in, out</i>	<i>list</i>	After this call returns this points to the structure <code>wifi_sta_list_t</code> allocated by the callee. This is variable length structure and depends on count variable inside it. The caller needs to free this buffer after use.. 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

5.7.1.49 wifi_sta_deauth()

```
int wifi_sta_deauth (
    uint8_t * mac_addr,
    uint16_t reason_code )
```

Dsiconnect ex-sta which is connected to in-uap.

Parameters

<i>in</i>	<i>mac_addr</i>	Mac address of external station.
<i>in</i>	<i>reason_code</i>	Deauth reason code.

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.50 wifi_enable_low_pwr_mode()

```
void wifi_enable_low_pwr_mode ( )
```

5.7.1.51 wifi_set_cal_data()

```
void wifi_set_cal_data (
    const uint8_t * cdata,
    const 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

5.7.1.52 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
----	------------	---------------------

5.7.1.53 _wifi_set_mac_addr()

```
void _wifi_set_mac_addr (
    const uint8_t * mac,
    mlan_bss_type bss_type )
```

Set wifi MAC address in firmware at run time.

This function may be used to set wifi MAC address in firmware as per passed bss type.

Parameters

in	<i>mac</i>	The new MAC Address
in	<i>bss_type</i>	BSS Type

5.7.1.54 wifi_get_wpa_ie_in_assoc()

```
int wifi_get_wpa_ie_in_assoc (
    uint8_t * wpa_ie )
```

5.7.1.55 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

5.7.1.56 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

5.7.1.57 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)

5.7.1.58 wifi_get_ipv6_multicast_mac()

```
void wifi_get_ipv6_multicast_mac (
    uint32_t ipaddr,
    uint8_t * mac_addr )
```

Get Multicast Mapped Mac address from IPv6 address

This function will generate Multicast Mapped MAC address from IPv6 address. Multicast Mapped MAC address will be in following format: 1) Higher 16-bits filled with IANA Multicast OUI (33-33) 2) Lower 32-bits filled with last 4 bytes of IPv6 address

Parameters

in	<i>ipaddr</i>	last 4 bytes of IPv6 address
in	<i>mac_addr</i>	multicast mapped MAC address

5.7.1.59 wifi_set_antenna()

```
int wifi_set_antenna (
    t_u32 ant_mode,
    t_u16 evaluate_time )
```

5.7.1.60 wifi_get_antenna()

```
int wifi_get_antenna (
    t_u32 * ant_mode,
    t_u16 * evaluate_time,
    t_u16 * current_antenna )
```


5.7.1.61 wifi_process_hs_cfg_resp()

```
void wifi_process_hs_cfg_resp (
    t_u8 * cmd_res_buffer )
```

5.7.1.62 wifi_process_ps_enh_response()

```
enum wifi_event_reason wifi_process_ps_enh_response (
    t_u8 * cmd_res_buffer,
    t_ul6 * ps_event,
    t_ul6 * action )
```

5.7.1.63 wifi_uap_rates_getset()

```
int wifi_uap_rates_getset (
    uint8_t action,
    char * rates,
    uint8_t num_rates )
```

5.7.1.64 wifi_uap_sta_ageout_timer_getset()

```
int wifi_uap_sta_ageout_timer_getset (
    uint8_t action,
    uint32_t * sta_ageout_timer )
```

5.7.1.65 wifi_uap_ps_sta_ageout_timer_getset()

```
int wifi_uap_ps_sta_ageout_timer_getset (
    uint8_t action,
    uint32_t * ps_sta_ageout_timer )
```

5.7.1.66 wifi_mem_access()

```
int wifi_mem_access (
    uint16_t action,
    uint32_t addr,
    uint32_t * value )
```

5.7.1.67 wifi_scan_process_results()

```
void wifi_scan_process_results (
    void )
```

5.7.1.68 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.

5.7.1.69 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.

5.7.1.70 `wifi_set_country_code()`

```
int wifi_set_country_code (
    const char * alpha2 )
```

Set/Get country code

Parameters

in	<i>alpha2</i>	country code in 3bytes string, 2bytes country code and 1byte 0 WW : World Wide Safe US : US FCC CA : IC Canada SG : Singapore EU : ETSI AU : Australia KR : Republic Of Korea FR : France JP : Japan CN : China
----	---------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.71 `wifi_get_country_code()`

```
int wifi_get_country_code (
    char * alpha2 )
```

5.7.1.72 `wifi_set_country_ie_ignore()`

```
int wifi_set_country_ie_ignore (
    uint8_t * ignore )
```

5.7.1.73 `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

5.7.1.74 wifi_uap_pmf_getset()

```
int wifi_uap_pmf_getset (
    uint8_t action,
    uint8_t * mfpc,
    uint8_t * mfpr )
```

Get/Set the uAP mfpc and mfpr

Parameters

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

Returns

cmd response status

5.7.1.75 wifi_uap_enable_11d_support()

```
int wifi_uap_enable_11d_support ( )
```

enable/disable 80211d domain feature for the uAP.

Note

This API only set 80211d domain feature. The actual application 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.

Returns

WM_SUCCESS on success or error code.

5.7.1.76 wifi_11d_is_channel_allowed()

```
bool wifi_11d_is_channel_allowed (
    int channel )
```

5.7.1.77 get_sub_band_from_region_code()

```
wifi_sub_band_set_t* get_sub_band_from_region_code (
    int region_code,
    t_u8 * nr_sb )
```

5.7.1.78 get_sub_band_from_region_code_5ghz()

```
wifi_sub_band_set_t* get_sub_band_from_region_code_5ghz (
    int region_code,
    t_u8 * nr_sb )
```

5.7.1.79 wifi_enable_11d_support()

```
int wifi_enable_11d_support ( )
```

5.7.1.80 wifi_enable_uap_11d_support()

```
int wifi_enable_uap_11d_support ( )
```

5.7.1.81 wifi_disable_11d_support()

```
int wifi_disable_11d_support ( )
```

5.7.1.82 wifi_disable_uap_11d_support()

```
int wifi_disable_uap_11d_support ( )
```

5.7.1.83 wifi_set_region_power_cfg()

```
int wifi_set_region_power_cfg (
    const t_u8 * data,
    t_u16 len )
```

5.7.1.84 wifi_set_txbfcap()

```
int wifi_set_txbfcap (
    unsigned int tx_bf_cap )
```

5.7.1.85 wifi_set_htcapinfo()

```
int wifi_set_htcapinfo (
    unsigned int htcapinfo )
```

5.7.1.86 wifi_set_httxcfg()

```
int wifi_set_httxcfg (
    unsigned short httxcfg )
```

5.7.1.87 wifi_uap_set_httxcfg()

```
void wifi_uap_set_httxcfg (
    const t_u16 ht_tx_cfg )
```

5.7.1.88 wifi_uap_set_httxcfg_int()

```
int wifi_uap_set_httxcfg_int (
    unsigned short httxcfg )
```

5.7.1.89 wifi_get_tx_power()

```
int wifi_get_tx_power (
    t_u32 * power_level )
```

5.7.1.90 wifi_set_tx_power()

```
int wifi_set_tx_power (
    t_u32 power_level )
```

5.7.1.91 wrapper_wlan_cmd_get_hw_spec()

```
int wrapper_wlan_cmd_get_hw_spec (
    void )
```

5.7.1.92 set_event_chanswann()

```
void set_event_chanswann (
    void )
```

5.7.1.93 clear_event_chanswann()

```
void clear_event_chanswann (
    void )
```

5.7.1.94 wifi_set_ps_cfg()

```
void wifi_set_ps_cfg (
    t_u16 multiple_dtims,
    t_u16 bcn_miss_timeout,
    t_u16 local_listen_interval,
    t_u16 adhoc_wake_period,
    t_u16 mode,
    t_u16 delay_to_ps )
```

5.7.1.95 wifi_send_hs_cfg_cmd()

```
int wifi_send_hs_cfg_cmd (
    mlan_bss_type interface,
    t_u32 ipv4_addr,
    t_u16 action,
    t_u32 conditions )
```

5.7.1.96 wrapper_wlan_11d_support_is_enabled()

```
bool wrapper_wlan_11d_support_is_enabled (
    void )
```

5.7.1.97 wrapper_wlan_11d_clear_parsedtable()

```
void wrapper_wlan_11d_clear_parsedtable (
    void )
```

5.7.1.98 wrapper_clear_media_connected_event()

```
void wrapper_clear_media_connected_event (
    void )
```

5.7.1.99 wifi_uap_ps_inactivity_sleep_exit()

```
int wifi_uap_ps_inactivity_sleep_exit (
    mlan_bss_type type )
```

5.7.1.100 wifi_uap_ps_inactivity_sleep_enter()

```
int wifi_uap_ps_inactivity_sleep_enter (
    mlan_bss_type type,
    unsigned int ctrl_bitmap,
    unsigned int min_sleep,
    unsigned int max_sleep,
    unsigned int inactivity_to,
    unsigned int min_awake,
    unsigned int max_awake )
```

5.7.1.101 wifi_enter_ieee_power_save()

```
int wifi_enter_ieee_power_save (
    void )
```


5.7.1.102 wifi_exit_ieee_power_save()

```
int wifi_exit_ieee_power_save (
    void )
```

5.7.1.103 wifi_enter_wnm_power_save()

```
int wifi_enter_wnm_power_save (
    t_ul6 wnm_sleep_time )
```

5.7.1.104 wifi_exit_wnm_power_save()

```
int wifi_exit_wnm_power_save (
    void )
```

5.7.1.105 wifi_enter_deepsleep_power_save()

```
int wifi_enter_deepsleep_power_save (
    void )
```

5.7.1.106 wifi_exit_deepsleep_power_save()

```
int wifi_exit_deepsleep_power_save (
    void )
```

5.7.1.107 wifi_set_power_save_mode()

```
int wifi_set_power_save_mode (
    void )
```

5.7.1.108 wifi_get_wakeup_reason()

```
int wifi_get_wakeup_reason (
    t_ul6 * hs_wakeup_reason )
```

5.7.1.109 send_sleep_confirm_command()

```
void send_sleep_confirm_command (
    mlan_bss_type interface )
```

5.7.1.110 wifi_configure_listen_interval()

```
void wifi_configure_listen_interval (
    int listen_interval )
```

5.7.1.111 wifi_configure_delay_to_ps()

```
void wifi_configure_delay_to_ps (
    unsigned int timeout_ms )
```

5.7.1.112 wifi_get_listen_interval()

```
unsigned short wifi_get_listen_interval ( )
```

5.7.1.113 wifi_get_delay_to_ps()

```
unsigned int wifi_get_delay_to_ps ( )
```

5.7.1.114 wifi_configure_null_pkt_interval()

```
void wifi_configure_null_pkt_interval (
    unsigned int null_pkt_interval )
```

5.7.1.115 wrapper_wifi_assoc()

```
int wrapper_wifi_assoc (
    const unsigned char * bssid,
    int wlan_security,
    bool is_wpa_tkip,
    unsigned int owe_trans_mode,
    bool is_ft )
```

5.7.1.116 wifi_uap_enable_sticky_bit()

```
void wifi_uap_enable_sticky_bit (
    const uint8_t * mac_addr )
```

5.7.1.117 wifi_get_xfer_pending()

```
bool wifi_get_xfer_pending (
    void )
```

5.7.1.118 wifi_set_xfer_pending()

```
void wifi_set_xfer_pending (
    bool xfer_val )
```

5.7.1.119 wrapper_wlan_cmd_11n_ba_stream_timeout()

```
int wrapper_wlan_cmd_11n_ba_stream_timeout (
    void * saved_event_buff )
```

5.7.1.120 wifi_set_txratecfg()

```
int wifi_set_txratecfg (
    wifi_ds_rate ds_rate,
    mlan_bss_type bss_type )
```

5.7.1.121 wifi_get_txratecfg()

```
int wifi_get_txratecfg (
    wifi_ds_rate * ds_rate,
    mlan_bss_type bss_type )
```

5.7.1.122 wifi_wake_up_card()

```
void wifi_wake_up_card (
    uint32_t * resp )
```

5.7.1.123 wifi_tx_card_awake_lock()

```
void wifi_tx_card_awake_lock (
    void )
```

5.7.1.124 wifi_tx_card_awake_unlock()

```
void wifi_tx_card_awake_unlock (
    void )
```

5.7.1.125 wrapper_wlan_11d_enable()

```
int wrapper_wlan_11d_enable (
    t_u32 state )
```

5.7.1.126 wrapper_wlan_uap_11d_enable()

```
int wrapper_wlan_uap_11d_enable (
    t_u32 state )
```

5.7.1.127 wifi_11h_enable()

```
int wifi_11h_enable (
    void )
```

5.7.1.128 wrapper_wlan_cmd_11n_addba_rspgen()

```
int wrapper_wlan_cmd_11n_addba_rspgen (
    void * saved_event_buff )
```

5.7.1.129 wrapper_wlan_cmd_11n_delba_rspgen()

```
int wrapper_wlan_cmd_11n_delba_rspgen (
    void * saved_event_buff )
```

5.7.1.130 wrapper_wlan_ecsa_enable()

```
int wrapper_wlan_ecsa_enable (
    void )
```

5.7.1.131 wifi_uap_start()

```
int wifi_uap_start (
    mlan_bss_type type,
    char * ssid,
    uint8_t * mac_addr,
    int security,
    int key_mgmt,
    char * passphrase,
    char * password,
    int channel,
    wifi_scan_chan_list_t scan_chan_list,
    uint8_t pwe_derivation,
    uint8_t transition_disable,
    bool mfpc,
    bool mfpr )
```

5.7.1.132 wrapper_wlan_sta_ampdu_enable()

```
int wrapper_wlan_sta_ampdu_enable (
    t_u8 tid )
```

5.7.1.133 wrapper_wlan_uap_ampdu_enable()

```
int wrapper_wlan_uap_ampdu_enable (
    uint8_t * addr,
    t_u8 tid )
```

5.7.1.134 wifi_set_packet_filters()

```
int wifi_set_packet_filters (
    wififlt_cfg_t * flt_cfg )
```

5.7.1.135 wifi_uap_stop()

```
int wifi_uap_stop ( )
```

5.7.1.136 wifi_uap_do_acs()

```
int wifi_uap_do_acs (
    const int * freq_list )
```

5.7.1.137 wifi_uap_config_wifi_capa()

```
void wifi_uap_config_wifi_capa (
    uint8_t wlan_capa )
```

Set uAP capability

User can set uAP capability of 11ax/11ac/11n/legacy. Default is 11ax.

Parameters

in	<i>wlan_capa</i>	uAP capability bitmap. 1111 - 11AX 0111 - 11AC 0011 - 11N 0001 - legacy
----	------------------	---

5.7.1.138 wifi_get_fw_info()

```
void wifi_get_fw_info (
    mlan_bss_type type,
    t_u16 * fw_bands )
```

5.7.1.139 wifi_get_data_rate()

```
int wifi_get_data_rate (
    wifi_ds_rate * ds_rate,
    mlan_bss_type bss_type )
```

5.7.1.140 wifi_uap_set_bandwidth()

```
int wifi_uap_set_bandwidth (
    const t_u8 bandwidth )
```

5.7.1.141 wifi_uap_get_bandwidth()

```
t_u8 wifi_uap_get_bandwidth ( )
```

5.7.1.142 wifi_uap_get_pmfcfg()

```
int wifi_uap_get_pmfcfg (
    t_u8 * mfpc,
    t_u8 * mfpr )
```

5.7.1.143 wifi_set_rts()

```
int wifi_set_rts (
    int rts,
    mlan_bss_type bss_type )
```

5.7.1.144 wifi_set_frag()

```
int wifi_set_frag (
    int frag,
    mlan_bss_type bss_type )
```

5.7.1.145 wifi_same_ess_ft()

```
bool wifi_same_ess_ft ( )
```

5.7.1.146 wifi_host_11k_cfg()

```
int wifi_host_11k_cfg (
    int enable_11k )
```

5.7.1.147 wifi_host_11k_neighbor_req()

```
int wifi_host_11k_neighbor_req (
    const char * ssid )
```

5.7.1.148 wifi_host_11v_bss_trans_query()

```
int wifi_host_11v_bss_trans_query (
    t_u8 query_reason )
```

5.7.1.149 wifi_clear_mgmt_ie()

```
int wifi_clear_mgmt_ie (
    mlan_bss_type bss_type,
    IEEEtypes_ElementId_t index,
    int mgmt_bitmap_index )
```

5.7.1.150 wifi_set_sta_mac_filter()

```
int wifi_set_sta_mac_filter (
    int filter_mode,
    int mac_count,
    unsigned char * mac_addr )
```

5.7.1.151 wifi_set_auto_arp()

```
int wifi_set_auto_arp (
    t_u32 * ipv4_addr )
```

5.7.1.152 wifi_tcp_keep_alive()

```
int wifi_tcp_keep_alive (
    wifi_tcp_keep_alive_t * keep_alive,
    t_u8 * src_mac,
    t_u32 src_ip )
```

5.7.1.153 wifi_cloud_keep_alive()

```
int wifi_cloud_keep_alive (
    wifi_cloud_keep_alive_t * keep_alive,
    t_u16 action,
    t_u8 * enable )
```


5.7.1.154 wifi_raw_packet_send()

```
int wifi_raw_packet_send (
    const t_u8 * packet,
    t_u32 length )
```

5.7.1.155 wifi_raw_packet_rcv()

```
int wifi_raw_packet_rcv (
    t_u8 ** data,
    t_u32 * pkt_type )
```

5.7.1.156 wifi_set_11ax_tx_omi()

```
int wifi_set_11ax_tx_omi (
    const mlan_bss_type bss_type,
    const t_u16 tx_omi,
    const t_u8 tx_option,
    const t_u8 num_data_pkts )
```

5.7.1.157 wifi_set_11ax_tol_time()

```
int wifi_set_11ax_tol_time (
    const t_u32 tol_time )
```

5.7.1.158 wifi_set_11ax_rtxpowerlimit()

```
int wifi_set_11ax_rtxpowerlimit (
    const void * rtx_pwr_cfg,
    uint32_t rtx_pwr_cfg_len )
```

5.7.1.159 wifi_set_11ax_rtxpowerlimit_legacy()

```
int wifi_set_11ax_rtxpowerlimit_legacy (
    const wifi_rtxpwrlimit_t * ru_pwr_cfg )
```

5.7.1.160 `wifi_get_11ax_rutxpowerlimit_legacy()`

```
int wifi_get_11ax_rutxpowerlimit_legacy (
    wifi_rutxpwrlimit_t * ru_pwr_cfg )
```

5.7.1.161 `wifi_set_11ax_cfg()`

```
int wifi_set_11ax_cfg (
    wifi_11ax_config_t * ax_config )
```

Set 11ax config params

Parameters

in, out	<i>ax_config</i>	11AX config parameters to be sent to Firmware
---------	------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.162 `wifi_set_btwt_cfg()`

```
int wifi_set_btwt_cfg (
    const wifi_btwt_config_t * btwt_config )
```

Set btwt config params

Parameters

in	<i>btwt_config</i>	Broadcast TWT setup parameters to be sent to Firmware
----	--------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.163 `wifi_set_twt_setup_cfg()`

```
int wifi_set_twt_setup_cfg (
    const wifi_twt_setup_config_t * twt_setup )
```

Set twt setup config params

Parameters

in	<i>twi_setup</i>	TWT Setup parameters to be sent to Firmware
----	------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.164 `wifi_set_twt_teardown_cfg()`

```
int wifi_set_twt_teardown_cfg (
    const wifi_twt_teardown_config_t * teardown_config )
```

Set twt teardown config params

Parameters

in	<i>teardown_config</i>	TWT Teardown parameters to be sent to Firmware
----	------------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.165 `wifi_get_twt_report()`

```
int wifi_get_twt_report (
    wifi_twt_report_t * twt_report )
```

Get twt report

Parameters

out	<i>twt_report</i>	TWT Report parameters to be sent to Firmware
-----	-------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.166 `wifi_twt_information()`

```
int wifi_twt_information (
    wifi_twt_information_t * information )
```

Twt information

Confidential

Parameters

out	<i>twi_report</i>	TWT Information
-----	-------------------	-----------------

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.167 `wifi_set_clocksync_cfg()`

```
int wifi_set_clocksync_cfg (
    const wifi\_clock\_sync\_gpio\_tsf\_t * tsf_latch,
    mlan_bss_type bss_type )
```

5.7.1.168 `wifi_get_tsf_info()`

```
int wifi_get_tsf_info (
    wifi\_tsf\_info\_t * tsf_info )
```

5.7.1.169 `wifi_set_rf_test_mode()`

```
int wifi_set_rf_test_mode (
    void )
```

5.7.1.170 `wifi_unset_rf_test_mode()`

```
int wifi_unset_rf_test_mode (
    void )
```

5.7.1.171 `wifi_set_rf_channel()`

```
int wifi_set_rf_channel (
    const uint8_t channel )
```

5.7.1.172 wifi_set_rf_radio_mode()

```
int wifi_set_rf_radio_mode (
    const uint8_t mode )
```

5.7.1.173 wifi_get_rf_channel()

```
int wifi_get_rf_channel (
    uint8_t * channel )
```

5.7.1.174 wifi_get_rf_radio_mode()

```
int wifi_get_rf_radio_mode (
    uint8_t * mode )
```

5.7.1.175 wifi_set_rf_band()

```
int wifi_set_rf_band (
    const uint8_t band )
```

5.7.1.176 wifi_get_rf_band()

```
int wifi_get_rf_band (
    uint8_t * band )
```

5.7.1.177 wifi_set_rf_bandwidth()

```
int wifi_set_rf_bandwidth (
    const uint8_t bandwidth )
```

5.7.1.178 wifi_get_rf_bandwidth()

```
int wifi_get_rf_bandwidth (
    uint8_t * bandwidth )
```

5.7.1.179 wifi_get_rf_per()

```
int wifi_get_rf_per (
    uint32_t * rx_tot_pkt_count,
    uint32_t * rx_mcast_bcast_count,
    uint32_t * rx_pkt_fcs_error )
```

5.7.1.180 wifi_set_rf_tx_cont_mode()

```
int wifi_set_rf_tx_cont_mode (
    const uint32_t enable_tx,
    const uint32_t cw_mode,
    const uint32_t payload_pattern,
    const uint32_t cs_mode,
    const uint32_t act_sub_ch,
    const uint32_t tx_rate )
```

5.7.1.181 wifi_set_rf_tx_antenna()

```
int wifi_set_rf_tx_antenna (
    const uint8_t antenna )
```

5.7.1.182 wifi_get_rf_tx_antenna()

```
int wifi_get_rf_tx_antenna (
    uint8_t * antenna )
```

5.7.1.183 wifi_set_rf_rx_antenna()

```
int wifi_set_rf_rx_antenna (
    const uint8_t antenna )
```

5.7.1.184 wifi_get_rf_rx_antenna()

```
int wifi_get_rf_rx_antenna (
    uint8_t * antenna )
```

5.7.1.185 wifi_set_rf_tx_power()

```
int wifi_set_rf_tx_power (
    const uint32_t power,
    const uint8_t mod,
    const uint8_t path_id )
```

5.7.1.186 wifi_cfg_rf_he_tb_tx()

```
int wifi_cfg_rf_he_tb_tx (
    uint16_t enable,
    uint16_t qnum,
    uint16_t aid,
    uint16_t axq_mu_timer,
    int16_t tx_power )
```

5.7.1.187 wifi_rf_trigger_frame_cfg()

```
int wifi_rf_trigger_frame_cfg (
    uint32_t Enable_tx,
    uint32_t Standalone_hetb,
    uint8_t FRAME_CTRL_TYPE,
    uint8_t FRAME_CTRL_SUBTYPE,
    uint16_t FRAME_DURATION,
    uint64_t TriggerType,
    uint64_t U1Len,
    uint64_t MoreTF,
    uint64_t CSRequired,
    uint64_t U1Bw,
    uint64_t LTFTType,
    uint64_t LTFMode,
    uint64_t LTFSymbol,
    uint64_t U1STBC,
    uint64_t LdpcESS,
    uint64_t ApTxPwr,
    uint64_t PreFecPadFct,
    uint64_t PeDisambig,
    uint64_t SpatialReuse,
    uint64_t Doppler,
    uint64_t HeSig2,
    uint32_t AID12,
    uint32_t RUAllocReg,
    uint32_t RUAlloc,
    uint32_t U1CodingType,
    uint32_t U1MCS,
    uint32_t U1DCM,
    uint32_t SSAlloc,
    uint8_t U1TargetRSSI,
    uint8_t MPDU_MU_SF,
    uint8_t TID_AL,
    uint8_t AC_PL,
    uint8_t Pref_AC )
```


5.7.1.188 wifi_set_rf_tx_frame()

```
int wifi_set_rf_tx_frame (
    const uint32_t enable,
    const uint32_t data_rate,
    const uint32_t frame_pattern,
    const uint32_t frame_length,
    const uint16_t adjust_burst_sifs,
    const uint32_t burst_sifs_in_us,
    const uint32_t short_preamble,
    const uint32_t act_sub_ch,
    const uint32_t short_gi,
    const uint32_t adv_coding,
    const uint32_t tx_bf,
    const uint32_t gf_mode,
    const uint32_t stbc,
    const uint8_t * bssid )
```

5.7.1.189 wifi_set_rf_otp_mac_addr()

```
int wifi_set_rf_otp_mac_addr (
    uint8_t * mac )
```

5.7.1.190 wifi_get_rf_otp_mac_addr()

```
int wifi_get_rf_otp_mac_addr (
    uint8_t * mac )
```

5.7.1.191 wifi_set_rf_otp_cal_data()

```
int wifi_set_rf_otp_cal_data (
    const uint8_t * cal_data,
    uint32_t cal_data_len )
```

5.7.1.192 wifi_get_rf_otp_cal_data()

```
int wifi_get_rf_otp_cal_data (
    uint8_t * cal_data )
```

5.7.1.193 wifi_register_fw_dump_cb()

```
void wifi_register_fw_dump_cb (
    int(*)() wifi_usb_mount_cb,
    int(*) (char *test_file_name) wifi_usb_file_open_cb,
    int(*) (uint8_t *data, size_t data_len) wifi_usb_file_write_cb,
    int(*)() wifi_usb_file_close_cb )
```

This function registers callbacks which are used to generate FW Dump on USB device.

Parameters

in	<i>wifi_usb_mount_cb</i>	Callback to mount usb device.
in	<i>wifi_usb_file_open_cb</i>	Callback to open file on usb device for FW dump.
in	<i>wifi_usb_file_write_cb</i>	Callback to write FW dump data to opened file.
in	<i>wifi_usb_file_close_cb</i>	Callback to close FW dump file.

5.7.1.194 **wifi_wmm_init()**

```
void wifi_wmm_init ( )
```

5.7.1.195 **wifi_wmm_get_pkt_prio()**

```
t_u32 wifi_wmm_get_pkt_prio (
    void * buf,
    t_u8 * tid )
```

5.7.1.196 **wifi_wmm_get_packet_cnt()**

```
t_u8 wifi_wmm_get_packet_cnt (
    void )
```

5.7.1.197 **wifi_handle_event_data_pause()**

```
void wifi_handle_event_data_pause (
    void * data )
```

5.7.1.198 **wifi_wmm_tx_stats_dump()**

```
void wifi_wmm_tx_stats_dump (
    int bss_type )
```

5.7.1.199 `wifi_set_rssi_low_threshold()`

```
int wifi_set_rssi_low_threshold (
    uint8_t * low_rssi )
```

5.7.1.200 `wifi_show_os_mem_stat()`

```
void wifi_show_os_mem_stat ( )
```

Show os mem alloc and free info.

5.7.1.201 `wifi_inject_frame()`

```
int wifi_inject_frame (
    const enum wlan_bss_type bss_type,
    const uint8_t * buff,
    const size_t len )
```

Frame Tx - Injecting Wireless frames from Host

This function is used to Inject Wireless frames from application directly.

Note

All injected frames will be sent on station interface. Application needs minimum of 2 KBytes stack for successful operation. Also application have to take care of allocating buffer for 802.11 Wireless frame (Header + Data) and freeing allocated buffer. Also this API may not work when Power Save is enabled on station interface.

Parameters

in	<i>bss_type</i>	The interface on which management frame needs to be send.
in	<i>buff</i>	Buffer holding 802.11 Wireless frame (Header + Data).
in	<i>len</i>	Length of the 802.11 Wireless frame.

Returns

WM_SUCCESS on success or error code.

5.7.1.202 `wifi_supp_inject_frame()`

```
int wifi_supp_inject_frame (
    const unsigned int bss_type,
    const uint8_t * buff,
    const size_t len )
```

5.7.1.203 wifi_is_wpa_supPLICant_input()

```
void wifi_is_wpa_supPLICant_input (
    const uint8_t interface,
    const uint8_t * buffer,
    const uint16_t len )
```

5.7.1.204 wifi_wpa_supPLICant_eapol_input()

```
void wifi_wpa_supPLICant_eapol_input (
    const uint8_t interface,
    const uint8_t * src_addr,
    const uint8_t * buffer,
    const uint16_t len )
```

5.7.1.205 wifi_get_sec_channel_offset()

```
t_u8 wifi_get_sec_channel_offset (
    unsigned int chan )
```

5.7.1.206 wifi_nxp_scan_res_get()

```
int wifi_nxp_scan_res_get (
    void )
```

5.7.1.207 wifi_nxp_survey_res_get()

```
int wifi_nxp_survey_res_get (
    void )
```

5.7.1.208 wifi_nxp_set_default_scan_ies()

```
int wifi_nxp_set_default_scan_ies (
    const u8 * ies,
    size_t ies_len )
```

5.7.1.209 wifi_nxp_reset_scan_flag()

```
void wifi_nxp_reset_scan_flag ( )
```

5.7.1.210 wifi_host_mbo_cfg()

```
int wifi_host_mbo_cfg (
    int enable_mbo )
```

5.7.1.211 wifi_mbo_preferch_cfg()

```
int wifi_mbo_preferch_cfg (
    t_u8 ch0,
    t_u8 prefer0,
    t_u8 ch1,
    t_u8 prefer1 )
```

5.7.1.212 wifi_mbo_send_preferch_wnm()

```
int wifi_mbo_send_preferch_wnm (
    t_u8 * src_addr,
    t_u8 * target_bssid,
    t_u8 ch0,
    t_u8 prefer0,
    t_u8 ch1,
    t_u8 prefer1 )
```

5.7.1.213 wifi_csi_cfg()

```
int wifi_csi_cfg (
    wifi_csi_config_params_t * csi_params )
```

Send the csi config parameter to FW.

Parameters

in	<i>csi_params</i>	Csi config parameter
----	-------------------	----------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.7.1.214 register_csi_user_callback()

```
int register_csi_user_callback (
    int(*) (void *buffer, size_t len) csi_data_rcv_callback )
```

5.7.1.215 unregister_csi_user_callback()

```
int unregister_csi_user_callback (
    void )
```

5.7.1.216 csi_local_buff_init()

```
void csi_local_buff_init ( )
```

5.7.1.217 csi_save_data_to_local_buff()

```
void csi_save_data_to_local_buff (
    void * data )
```

5.7.1.218 csi_deliver_data_to_user()

```
void csi_deliver_data_to_user ( )
```

5.7.1.219 wifi_send_mgmt_auth_request()

```
int wifi_send_mgmt_auth_request (
    const t_u8 channel,
    const t_u8 auth_alg,
    const t_u8 * auth_seq_num,
    const t_u8 * status_code,
    const t_u8 * dest,
    const t_u8 * sae_data,
    const t_u16 sae_data_len )
```

5.7.1.220 wifi_send_scan_cmd()

```
int wifi_send_scan_cmd (
    t_u8 bss_mode,
    const t_u8 * specific_bssid,
    const char * ssid,
    uint8_t ssid_num,
    const t_u8 num_channels,
    const wifi_scan_channel_list_t * chan_list,
    const t_u8 num_probes,
    const t_u16 scan_chan_gap,
    const bool keep_previous_scan,
    const bool active_scan_triggered )
```

5.7.1.221 wifi_deauthenticate()

```
int wifi_deauthenticate (
    uint8_t * bssid )
```

5.7.1.222 wifi_get_turbo_mode()

```
int wifi_get_turbo_mode (
    t_u8 * mode )
```

5.7.1.223 wifi_get_uap_turbo_mode()

```
int wifi_get_uap_turbo_mode (
    t_u8 * mode )
```

5.7.1.224 wifi_set_turbo_mode()

```
int wifi_set_turbo_mode (
    t_u8 mode )
```

5.7.1.225 wifi_set_uap_turbo_mode()

```
int wifi_set_uap_turbo_mode (
    t_u8 mode )
```

5.7.1.226 wifi_get_default_ht_capab()

```
t_u16 wifi_get_default_ht_capab ( )
```

5.7.1.227 wifi_get_default_vht_capab()

```
t_u32 wifi_get_default_vht_capab ( )
```

5.7.1.228 wifi_uap_client_assoc()

```
void wifi_uap_client_assoc (
    t_u8 * sta_addr,
    unsigned char is_11n_enabled )
```

5.7.1.229 wifi_uap_client_deauth()

```
void wifi_uap_client_deauth (
    t_u8 * sta_addr )
```

5.7.1.230 region_string_2_region_code()

```
t_u8 region_string_2_region_code (
    t_u8 * region_string )
```

Parameters

<i>region_string</i>	Region string
----------------------	---------------

Returns

Region code

5.7.1.231 wifi_set_indrst_cfg()

```
int wifi_set_indrst_cfg (
    const wifi_indrst_cfg_t * indrst_cfg,
    mlan_bss_type bss_type )
```


5.7.1.232 wifi_get_indrst_cfg()

```
int wifi_get_indrst_cfg (
    wifi_indrst_cfg_t * indrst_cfg,
    mlan_bss_type bss_type )
```

5.7.1.233 wifi_test_independent_reset()

```
int wifi_test_independent_reset ( )
```

5.7.1.234 wifi_trigger_oob_indrst()

```
int wifi_trigger_oob_indrst ( )
```

5.7.1.235 hostapd_connected_sta_list()

```
void hostapd_connected_sta_list (
    wifi_sta_info_t * si,
    wifi_sta_list_t * sl )
```

5.7.1.236 wifi_is_remain_on_channel()

```
bool wifi_is_remain_on_channel (
    void )
```

5.7.2 Macro Documentation**5.7.2.1 CONFIG_STA_AUTO_DHCPV4**

```
#define CONFIG_STA_AUTO_DHCPV4 1
```

5.7.2.2 CONFIG_WIFI_STA_RECONNECT

```
#define CONFIG_WIFI_STA_RECONNECT 1
```

5.7.2.3 CONFIG_WIFI_AUTO_POWER_SAVE

```
#define CONFIG_WIFI_AUTO_POWER_SAVE 1
```

5.7.2.4 CONFIG_GTK_REKEY_OFFLOAD

```
#define CONFIG_GTK_REKEY_OFFLOAD 0
```

5.7.2.5 CONFIG_TCP_ACK_ENH

```
#define CONFIG_TCP_ACK_ENH 1
```

5.7.2.6 CONFIG_FW_VDLL

```
#define CONFIG_FW_VDLL 1
```

5.7.2.7 WIFI_REG8

```
#define WIFI_REG8(  
    x ) (*(volatile unsigned char *) (x))
```

5.7.2.8 WIFI_REG16

```
#define WIFI_REG16(  
    x ) (*(volatile unsigned short *) (x))
```

5.7.2.9 WIFI_REG32

```
#define WIFI_REG32(  
    x ) (*(volatile unsigned int *) (x))
```

5.7.2.10 WIFI_WRITE_REG8

```
#define WIFI_WRITE_REG8(  
    reg,  
    val ) (WIFI_REG8(reg) = (val))
```

5.7.2.11 WIFI_WRITE_REG16

```
#define WIFI_WRITE_REG16(  
    reg,  
    val ) (WIFI_REG16(reg) = (val))
```

5.7.2.12 WIFI_WRITE_REG32

```
#define WIFI_WRITE_REG32(  
    reg,  
    val ) (WIFI_REG32(reg) = (val))
```

5.7.2.13 WIFI_COMMAND_RESPONSE_WAIT_MS

```
#define WIFI_COMMAND_RESPONSE_WAIT_MS 20000
```

5.7.2.14 BANDWIDTH_20MHZ

```
#define BANDWIDTH_20MHZ 1U
```

5.7.2.15 BANDWIDTH_40MHZ

```
#define BANDWIDTH_40MHZ 2U
```

5.7.2.16 BANDWIDTH_80MHZ

```
#define BANDWIDTH_80MHZ 3U
```

5.7.2.17 MAX_NUM_CHANS_IN_NBOR_RPT

```
#define MAX_NUM_CHANS_IN_NBOR_RPT 6U
```

5.7.2.18 MBIT

```
#define MBIT(  
    x ) ((t_u32)1) << (x)
```

BIT value

5.7.2.19 WIFI_MGMT_DIASOC

```
#define WIFI_MGMT_DIASOC MBIT(10)
```

5.7.2.20 WIFI_MGMT_AUTH

```
#define WIFI_MGMT_AUTH MBIT(11)
```

5.7.2.21 WIFI_MGMT_DEAUTH

```
#define WIFI_MGMT_DEAUTH MBIT(12)
```

5.7.2.22 WIFI_MGMT_ACTION

```
#define WIFI_MGMT_ACTION MBIT(13)
```

BITMAP for Action frame

5.7.2.23 BEACON_REPORT_BUF_SIZE

```
#define BEACON_REPORT_BUF_SIZE 1400
```

5.7.2.24 MAX_NEIGHBOR_AP_LIMIT

```
#define MAX_NEIGHBOR_AP_LIMIT 6U
```

5.7.3 Enumeration Type Documentation

5.7.3.1 anonymous enum

```
anonymous enum
```

WiFi Error Code

Enumerator

WM_E_WIFI_ERRNO_START	
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.

5.7.3.2 anonymous enum

```
anonymous enum
```

WiFi driver TX/RX data status

Enumerator

WIFI_DATA_RUNNING	Data in running status
WIFI_DATA_BLOCK	Data in block status

5.7.3.3 IEEEtypes_ElementId_t

```
enum IEEEtypes_ElementId_t
```

Enumerator

MGMT_RSN_IE	
MGMT_RRM_ENABLED_CAP	
MGMT_VENDOR_SPECIFIC_221	
MGMT_WPA_IE	
MGMT_WPS_IE	
MGMT_MBO_IE	

5.7.3.4 wifi_reg_t

```
enum wifi_reg_t
```

Enumerator

REG_MAC	
REG_BBP	
REG_RF	
REG_CAU	

5.7.3.5 wlan_rrm_beacon_reporting_detail

```
enum wlan_rrm_beacon_reporting_detail
```

Enumerator

WLAN_RRM_REPORTING_DETAIL_NONE	
WLAN_RRM_REPORTING_DETAIL_AS_REQUEST	
WLAN_RRM_REPORTING_DETAIL_ALL_FIELDS_AND_ELEMENTS	

5.7.3.6 wlan_nlist_mode

```
enum wlan_nlist_mode
```

Enumerator

WLAN_NLIST_11K	
WLAN_NLIST_11V	
WLAN_NLIST_11V_PREFERRED	

5.7.4 Variable Documentation

5.7.4.1 wifi_tx_status

t_u8 wifi_tx_status

5.7.4.2 wifi_tx_block_cnt

t_u8 wifi_tx_block_cnt

5.7.4.3 wifi_rx_status

t_u8 wifi_rx_status

5.7.4.4 wifi_rx_block_cnt

t_u8 wifi_rx_block_cnt

5.7.4.5 g_bcn_nf_last

int16_t g_bcn_nf_last

5.7.4.6 g_rssi

uint8_t g_rssi

5.7.4.7 g_data_nf_last

uint16_t g_data_nf_last

5.7.4.8 g_data_snr_last

```
uint16_t g_data_snr_last
```

5.7.4.9 wifi_shutdown_enable

```
bool wifi_shutdown_enable
```

5.7.4.10 csi_event_cnt

```
int csi_event_cnt
```

5.7.4.11 csi_event_data_len

```
t_u64 csi_event_data_len
```

5.8 wifi_events.h File Reference

Wi-Fi events.

5.8.1 Enumeration Type Documentation

5.8.1.1 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_CONN	uAP Client connected
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

Enumerator

WIFI_EVENT_SCAN_START	Scan start event when scan is started
WIFI_EVENT_SCAN_RESULT	Scan Result
WIFI_EVENT_SURVEY_RESULT_GET	Survey Result Get
WIFI_EVENT_GET_HW_SPEC	Get hardware spec
WIFI_EVENT_ASSOCIATION	Association
WIFI_EVENT_ASSOCIATION_NOTIFY	Association Notify
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_RSSI_LOW	
WIFI_EVENT_FW_HANG	Firmware Hang event
WIFI_EVENT_FW_RESET	Firmware Reset event
WIFI_EVENT_NET_STA_ADDR_CONFIG	Network station address configuration
WIFI_EVENT_NET_INTERFACE_CONFIG	Network interface configuration
WIFI_EVENT_WEP_CONFIG	WEP configuration
WIFI_EVENT_STA_MAC_ADDR_CONFIG	STA MAC address configuration
WIFI_EVENT_UAP_MAC_ADDR_CONFIG	UAP MAC address configuration
WIFI_EVENT_NET_DHCP_CONFIG	Network DHCP configuration
WIFI_EVENT_SUPPLICANT_PMK	Supplicant PMK
WIFI_EVENT_SLEEP	Sleep
WIFI_EVENT_IEEE_PS	IEEE PS
WIFI_EVENT_DEEP_SLEEP	Deep Sleep
WIFI_EVENT_WNM_PS	WNM ps
WIFI_EVENT_IEEE_DEEP_SLEEP	IEEE and Deep Sleep
WIFI_EVENT_WNM_DEEP_SLEEP	WNM and Deep Sleep
WIFI_EVENT_PS_INVALID	PS Invalid
WIFI_EVENT_ERR_MULTICAST	Error Multicast
WIFI_EVENT_ERR_UNICAST	error Unicast
WIFI_EVENT_NLIST_REPORT	802.11K/11V neighbor report
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_NET_IPV6_CONFIG	IPv6 address state change
WIFI_EVENT_BG_SCAN_REPORT	
WIFI_EVENT_BG_SCAN_STOPPED	
WIFI_EVENT_MGMT_FRAME	
WIFI_EVENT_REMAIN_ON_CHANNEL	
WIFI_EVENT_MGMT_TX_STATUS	
WIFI_EVENT_CSI	
WIFI_EVENT_SYNC_REGION_CODE	Event to sync region code with connected AP
WIFI_EVENT_REGION_POWER_CFG	Event to set region power
WIFI_EVENT_LAST	Event to indicate end of Wi-Fi events

5.8.1.2 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

5.8.1.3 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

5.8.1.4 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

5.8.1.5 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

5.9 wifi_ping.h File Reference

This file provides the support for network utility ping.

5.9.1 Function Documentation

5.9.1.1 ping_cli_init()

```
int ping_cli_init (
    void )
```

Register Network Utility CLI commands.

Register the Network Utility CLI commands. Currently, only ping command is supported.

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)

5.9.1.2 ping_stats()

```
void ping_stats (
    int * total,
    int * recvd )
```

5.9.1.3 ping_cli_deinit()

```
int ping_cli_deinit (
    void )
```

Unregister Network Utility CLI commands.

Unregister the Network Utility CLI commands.

Returns

WM_SUCCESS if the CLI commands are unregistered
-WM_FAIL otherwise

5.9.2 Macro Documentation

5.9.2.1 ping_e

```
#define ping_e(
    ... ) wmlog_e("ping", ##__VA_ARGS__)
```

5.9.2.2 ping_w

```
#define ping_w(
    ... ) wmlog_w("ping", ##__VA_ARGS__)
```

5.9.2.3 PING_ID

```
#define PING_ID 0xAFAFU
```

5.9.2.4 PING_INTERVAL

```
#define PING_INTERVAL 1000
```

5.9.2.5 PING_DEFAULT_TIMEOUT_SEC

```
#define PING_DEFAULT_TIMEOUT_SEC 2
```

5.9.2.6 PING_DEFAULT_COUNT

```
#define PING_DEFAULT_COUNT 10
```

5.9.2.7 PING_DEFAULT_SIZE

```
#define PING_DEFAULT_SIZE 56
```

5.9.2.8 PING_MAX_SIZE

```
#define PING_MAX_SIZE 65507U
```

5.9.2.9 PING_MAX_COUNT

```
#define PING_MAX_COUNT 65535U
```

5.10 wlan.h File Reference

WLAN Connection Manager.

5.10.1 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.

5.10.2 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.

5.10.3 Function Documentation

5.10.3.1 is_valid_security()

```
static int is_valid_security (
    int security ) [inline], [static]
```

5.10.3.2 is_ep_valid_security()

```
static int is_ep_valid_security (
    int security ) [inline], [static]
```

5.10.3.3 verify_scan_duration_value()

```
int verify_scan_duration_value (
    int scan_duration )
```

5.10.3.4 verify_scan_channel_value()

```
int verify_scan_channel_value (
    int channel )
```

5.10.3.5 verify_split_scan_delay()

```
int verify_split_scan_delay (
    int delay )
```

5.10.3.6 set_scan_params()

```
int set_scan_params (
    struct wifi_scan_params_t * wifi_scan_params )
```

5.10.3.7 get_scan_params()

```
int get_scan_params (
    struct wifi_scan_params_t * wifi_scan_params )
```

5.10.3.8 wlan_get_current_rssi()

```
int wlan_get_current_rssi (
    short * rssi )
```

5.10.3.9 wlan_get_current_nf()

```
int wlan_get_current_nf (
    void )
```

5.10.3.10 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.

5.10.3.11 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	cb	A pointer to a callback function that handles WLAN events. All further WLCMGR events will be notified in this callback. Refer to enum wlan_event_reason 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.

5.10.3.12 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.

5.10.3.13 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 WLAN_ACTIVE: no action to be taken
---------------	--

5.10.3.14 wlan_remove_all_network_profiles()

```
int wlan_remove_all_network_profiles (
    void )
```

Stop and Remove all wireless network profiles.

Returns

WM_SUCCESS if successful.

5.10.3.15 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
-----	------------	---

5.10.3.16 wlan_initialize_sta_network()

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

WLAN initialize station network information

This API initializes a default station network. The network ssid, passphrase is initialized to NULL. Channel is set to auto.

Parameters

out	net	Pointer to the initialized micro-AP network
-----	-----	---

5.10.3.17 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	network	A pointer to the wlan_network 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.

5.10.3.18 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.

5.10.3.19 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.

Confidential

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.

5.10.3.20 wlan_connect_opt()

```
int wlan_connect_opt (
    char * name,
    bool skip_dfs )
```

Connect to a wireless network (Access Point) with options.

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.
in	<i>skip_dfs</i>	Option to skip DFS channel when doing scan.

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.

5.10.3.21 wlan_reassociate()

```
int wlan_reassociate ( )
```

Reassociate to a wireless network (Access Point).

When this function is called, WLAN Connection Manager starts reassociation attempts using same SSID as currently connected network . 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 have no effect.

Calling this function when the station interface is in the [WLAN_CONNECTED](#) state will, if successful, cause the interface to reassociate to another network(AP).

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_AUTH_FAILED](#), [WLAN_REASON_CONNECT_FAILED](#) or [WLAN_REASON_ADDRESS_FAILED](#) are reported as appropriate.

Returns

- WM_SUCCESS if a reassociation attempt was started successfully
- WLAN_ERROR_STATE if the WLAN Connection Manager was not running. or WLAN Connection Manager was not in [WLAN_CONNECTED](#) state.
- WM_E_INVALID if there are no known networks to connect to
- WM_FAIL if an internal error has occurred.

5.10.3.22 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

5.10.3.23 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.

5.10.3.24 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.

5.10.3.25 wlan_get_mac_address()

```
int wlan_get_mac_address (
    uint8_t * dest )
```

Retrieve the wireless MAC address of station interface.

This function copies the MAC address of the station interface to *sta_mac* address and uAP interface to *uap_mac* address.

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 *sta_mac* or *uap_mac* is NULL.

5.10.3.26 wlan_get_mac_address_uap()

```
int wlan_get_mac_address_uap (
    uint8_t * dest )
```

Retrieve the wireless MAC address of 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.

5.10.3.27 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 wlan_ip_config .
-----	-------------	---

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.

5.10.3.28 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 wlan_ip_config .
-----	-------------	---

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.

5.10.3.29 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.

5.10.3.30 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 wlan_network .
-----	----------------	---

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.

5.10.3.31 wlan_get_current_network_ssid()

```
int wlan_get_current_network_ssid (
    char * ssid )
```

Retrieve the current network ssid of station interface.

This function retrieves the current network ssid of station interface when the station interface is in the [WLAN_CONNECTED](#) state.

Parameters

out	ssid	A pointer to the ssid.
-----	------	------------------------

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.

5.10.3.32 wlan_get_current_network_bssid()

```
int wlan_get_current_network_bssid (
    char * bssid )
```

Retrieve the current network bssid of station interface.

This function retrieves the current network bssid of station interface when the station interface is in the [WLAN_CONNECTED](#) state.

Parameters

out	bssid	A pointer to the bssid.
-----	-------	-------------------------

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.

5.10.3.33 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 wlan_network .
-----	----------------	---

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.

5.10.3.34 wlan_get_current_uap_network_ssid()

```
int wlan_get_current_uap_network_ssid (
    char * ssid )
```

Retrieve the current network ssid of micro-AP interface.

This function retrieves the current network ssid of micro-AP interface when the micro-AP interface is in the [WLAN_UAP_STARTED](#) state.

Parameters

out	<i>ssid</i>	A pointer to the ssid.
-----	-------------	------------------------

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.

5.10.3.35 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.

5.10.3.36 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.

5.10.3.37 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.

5.10.3.38 is_sta_ipv6_connected()

```
bool is_sta_ipv6_connected (
    void )
```

Retrieve the status information of the ipv6 network of station interface.

Returns

TRUE if ipv6 network of station interface is in [WLAN_CONNECTED](#) state.
FALSE otherwise.

5.10.3.39 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 wlan_network 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.

5.10.3.40 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 wlan_network 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.

5.10.3.41 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.

5.10.3.42 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 wlan_connection_state 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.

5.10.3.43 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 wlan_connection_state 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.

5.10.3.44 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.

5.10.3.45 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>t_wlan_scan_param</i>	A wlan_scan_params_v2_t 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.

5.10.3.46 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 wlan_scan_result 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).

5.10.3.47 wlan_enable_low_pwr_mode()

```
int wlan_enable_low_pwr_mode ( )
```

Enable Low Power Mode in Wireless Firmware.

Note

When low power mode is enabled, the output power will be clipped at ~+10dBm and the expected PA current is expected to be in the 80-90 mA range for b/g/n modes.

This function may be called to enable low power mode in firmware. This should be called before [wlan_init\(\)](#) function.

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

5.10.3.48 wlan_set_ed_mac_mode()

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

Configure ED MAC mode for Station 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.

5.10.3.49 wlan_set_uap_ed_mac_mode()

```
int wlan_set_uap_ed_mac_mode (
    wlan_ed_mac_ctrl_t wlan_ed_mac_ctrl )
```

Configure ED MAC mode for Micro AP 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.

5.10.3.50 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 for Station.

Parameters

out	<i>wlan_ed_mac_ctrl</i>	A pointer to wlan_ed_mac_ctrl_t with parameters mentioned in above set API.
-----	-------------------------	---

Returns

WM_SUCCESS if the call was successful.
 -WM_FAIL if failed.

5.10.3.51 wlan_get_uap_ed_mac_mode()

```
int wlan_get_uap_ed_mac_mode (
    wlan_ed_mac_ctrl_t * wlan_ed_mac_ctrl )
```

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

Parameters

out	<i>wlan_ed_mac_ctrl</i>	A pointer to wlan_ed_mac_ctrl_t with parameters mentioned in above set API.
-----	-------------------------	---

Returns

WM_SUCCESS if the call was successful.
 -WM_FAIL if failed.

5.10.3.52 wlan_set_cal_data()

```
void wlan_set_cal_data (
    const uint8_t * cal_data,
    const 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.

Confidential

Parameters

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

5.10.3.53 wlan_set_mac_addr()

```
int 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. When called after wlan init done, the incoming mac is treated as the sta mac address directly. And mac[4] plus 1 the modified mac as the UAP mac address.

Parameters

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

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

5.10.3.54 wlan_set_sta_mac_addr()

```
int wlan_set_sta_mac_addr (
    uint8_t * mac )
```

Set wireless MAC Address for STA in WLAN firmware.

This function may be called to set wireless MAC Address in firmware. This should be call before [wlan_init\(\)](#) function. When called after wlan init done, it will set only STA MAC address.

Parameters

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

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

5.10.3.55 wlan_set_uap_mac_addr()

```
int wlan_set_uap_mac_addr (
    uint8_t * mac )
```

Set wireless MAC Address for uAP in WLAN firmware.

This function may be called to set wireless MAC Address in firmware. This should be call before [wlan_init\(\)](#) function. When called after wlan init done, it will set only uAP MAC address.

Parameters

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

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

5.10.3.56 wlan_set_roaming()

```
int wlan_set_roaming (
    const int enable,
    const uint8_t rssi_low_threshold )
```

Set soft roaming config.

This function may be called to enable/disable soft roaming by specifying the RSSI threshold.

Note

RSSI Threshold setting for soft roaming: The provided RSSI low threshold value is used to subscribe RSSI low event from firmware, on reception of this event background scan is started in firmware with same RSSI threshold to find out APs with better signal strength than RSSI threshold.

If AP is found then roam attempt is initiated, otherwise background scan started again till limit reaches to BG_SCAN_LIMIT.

If still AP is not found then WLAN connection manager sends [WLAN_REASON_BGSCAN_NETWORK_NOT_FOUND](#) event to application. In this case, if application again wants to use soft roaming then it can call this API again or use [wlan_set_rssi_low_threshold](#) API to set RSSI low threshold again.

Parameters

in	<i>enable</i>	Enable/disable roaming.
in	<i>rssi_low_threshold</i>	RSSI low threshold value

Returns

WM_SUCCESS if the call was successful.
 -WM_FAIL if failed.

5.10.3.57 wlan_get_roaming_status()

```
int wlan_get_roaming_status ( )
```

5.10.3.58 wlan_set_ieeeeps_cfg()

```
int wlan_set_ieeeeps_cfg (
    struct wlan_ieeeeps_config * ps_cfg )
```

Set configuration parameters of IEEE power save mode.

Parameters

in	<i>ps_cfg</i>	: powersave configuration includes multiple parameters.
----	---------------	---

Returns

WM_SUCCESS if the call was successful.
 -WM_FAIL if failed.

5.10.3.59 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 sent 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
----	------------------------	--

5.10.3.60 wlan_configure_delay_to_ps()

```
void wlan_configure_delay_to_ps (
    unsigned int timeout_ms )
```

5.10.3.61 wlan_get_listen_interval()

```
unsigned short wlan_get_listen_interval ( )
```

5.10.3.62 wlan_get_delay_to_ps()

```
unsigned int wlan_get_delay_to_ps ( )
```

5.10.3.63 wlan_is_power_save_enabled()

```
bool wlan_is_power_save_enabled ( )
```

5.10.3.64 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.
----	---------------------	---

5.10.3.65 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.

5.10.3.66 wlan_get_antcfg()

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

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.
out	<i>current_antenna</i>	pointer to current antenna.

Returns

WM_SUCCESS if successful.
WLAN_ERROR_STATE if unsuccessful.

5.10.3.67 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

5.10.3.68 wlan_version_extended()

```
void wlan_version_extended (
    void )
```

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

5.10.3.69 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.

5.10.3.70 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 wlan_wakeup_event_t . Typically devices would want to wake up on WAKE_ON_ALL_BROADCAST , WAKE_ON_UNICAST , WAKE_ON_MAC_EVENT , WAKE_ON_MULTICAST , WAKE_ON_ARP_BROADCAST , WAKE_ON_MGMT_FRAME
----	--------------------------	---

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL otherwise.

5.10.3.71 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.
-WM_FAIL otherwise.

5.10.3.72 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.
-WM_FAIL otherwise.

5.10.3.73 wlan_deepsleeps_off()

```
int wlan_deepsleeps_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.
-WM_FAIL otherwise.

5.10.3.74 wlan_tcp_keep_alive()

```
int wlan_tcp_keep_alive (
    wlan_tcp_keep_alive_t * keep_alive )
```

Use this API to configure the TCP Keep alive parameters in Wi-Fi firmware. [wlan_tcp_keep_alive_t](#) provides the parameters which are available for configuration.

Note

To reset current TCP Keep alive configuration just pass the reset with value 1, all other parameters are ignored in this case.

Please note that this API must be called after successful connection and before putting Wi-Fi card in IEEE power save mode.

Parameters

in	<i>keep_alive</i>	A pointer to wlan_tcp_keep_alive_t with following parameters. enable Enable keep alive reset Reset keep alive timeout Keep alive timeout interval Keep alive interval max_keep_alives Maximum keep alives dst_mac Destination MAC address dst_ip Destination IP dst_tcp_port Destination TCP port src_tcp_port Source TCP port seq_no Sequence number
----	-------------------	---

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.75 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.

5.10.3.76 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](#).

5.10.3.77 wlan_get_data_rate()

```
int wlan_get_data_rate (
    wlan_ds_rate * ds_rate,
    mlan_bss_type bss_type )
```

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.
in	<i>bss_type</i>	0: STA, 1: uAP

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.

5.10.3.78 wlan_get_pmfcfg()

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

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

Parameters

out	<i>mfpc</i>	Management Frame Protection Capable (MFPC) 1: Management Frame Protection Capable 0: Management Frame Protection not Capable
out	<i>mfpr</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.

5.10.3.79 wlan_uap_get_pmfcfg()

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

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

Parameters

out	<i>mfpc</i>	Management Frame Protection Capable (MFPC) 1: Management Frame Protection Capable 0: Management Frame Protection not Capable
out	<i>mfpr</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.

5.10.3.80 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 "&&" and " " operator,space can not be removed between operator. RPN=Filter_0 && 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(&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.

5.10.3.81 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.

5.10.3.82 wlan_wowlan_cfg_ptn_match()

```
int wlan_wowlan_cfg_ptn_match (
    wlan_wowlan_ptn_cfg_t * ptn_cfg )
```

Use this API to enable WOWLAN on magic pkt rx in Wi-Fi firmware

Parameters

in	<i>ptn_cfg</i>	A pointer to wlan_wowlan_ptn_cfg_t containing Wake on WLAN pattern configuration
----	----------------	--

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails

5.10.3.83 wlan_set_ipv6_ns_offload()

```
int wlan_set_ipv6_ns_offload ( )
```

Use this API to enable NS Offload in Wi-Fi firmware.

Returns

WM_SUCCESS if operation is successful.
 -WM_FAIL if command fails.

5.10.3.84 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.

5.10.3.85 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.

5.10.3.86 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 wlan_ps_mode where station interface power save mode will be stored.
-----	----------------	---

Returns

WM_SUCCESS if successful.
-WM_E_INVALID if *ps_mode* was NULL.

5.10.3.87 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 wifi_event .
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.

5.10.3.88 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).

5.10.3.89 wlan_wfa_basic_cli_deinit()

```
int wlan_wfa_basic_cli_deinit (
    void )
```

Unregister WFA basic WLAN CLI commands

This function unregisters 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 unregistered or
WLAN_ERROR_ACTION if they were not unregistered

5.10.3.90 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).

5.10.3.91 wlan_basic_cli_deinit()

```
int wlan_basic_cli_deinit (  
    void )
```

Unregister basic WLAN CLI commands

This function unregisters 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 unregistered or
WLAN_ERROR_ACTION if they were not unregistered (for example if this function was called while the CLI commands were already registered).

5.10.3.92 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).

5.10.3.93 wlan_cli_deinit()

```
int wlan_cli_deinit (
    void )
```

Unregister WLAN CLI commands.

Try to unregister 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_deinit\(\)](#), hence only one function out of these two functions should be called in the application.

Returns

WM_SUCCESS if the CLI commands were unregistered or

-WM_FAIL if they were not (for example if this function was called while the CLI commands were already unregistered).

5.10.3.94 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).

5.10.3.95 wlan_enhanced_cli_deinit()

```
int wlan_enhanced_cli_deinit (
    void )
```

Unregister WLAN enhanced CLI commands.

Unregister 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 unregistered or
-WM_FAIL if they were not unregistered.

5.10.3.96 wlan_test_mode_cli_init()

```
int wlan_test_mode_cli_init (
    void )
```

Register WLAN Test Mode CLI commands.

Register the WLAN Test Mode CLI commands like set or get channel, band, bandwidth, PER and more 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).

5.10.3.97 wlan_test_mode_cli_deinit()

```
int wlan_test_mode_cli_deinit (
    void )
```

Unregister WLAN Test Mode CLI commands.

Unregister the WLAN Test Mode CLI commands like set or get channel, band, bandwidth, PER and more 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 unregistered or
-WM_FAIL if they were not unregistered

5.10.3.98 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.

5.10.3.99 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.

5.10.3.100 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.

5.10.3.101 wlan_set_htcapinfo()

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

This API can be used to configure some of parameters in HTCapiInfo 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> <p>Bit 29: Green field enable/disable</p> <p>Bit 26: Rx STBC Support enable/disable. (As we support single spatial stream only 1 bit is used for Rx STBC)</p> <p>Bit 25: Tx STBC support enable/disable.</p> <p>Bit 24: Short GI in 40 Mhz enable/disable</p> <p>Bit 23: Short GI in 20 Mhz enable/disable</p> <p>Bit 22: Rx LDPC enable/disable</p> <p>Bit 17: 20/40 Mhz enable disable.</p> <p>Bit 8: Enable/disable 40Mhz Intolarent bit in ht capinfo.</p> <p>0 will reset this bit and 1 will set this bit in htcapinfo attached in assoc request.</p> <p>All others are reserved and should be set to 0.</p>
----	------------------	--

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

5.10.3.102 wlan_set_httxcf()

```
int wlan_set_httxcf (
    unsigned short httxcf )
```

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

Parameters

in	<i>httxcf</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.

5.10.3.103 wlan_set_txratecfg()

```
int wlan_set_txratecfg (  
    wlan_ds_rate ds_rate,  
    mlan_bss_type bss_type )
```

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>
----	----------------	--

Parameters

in	<i>bss_type</i>	0: STA, 1: uAP
----	-----------------	----------------

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

5.10.3.104 wlan_get_txratecfg()

```
int wlan_get_txratecfg (
    wlan_ds_rate * ds_rate,
    wlan_bss_type bss_type )
```

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

Parameters

in	<i>ds_rate</i>	A pointer to wlan_ds_rate where Tx Rate configuration will be stored.
in	<i>bss_type</i>	0: STA, 1: uAP

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

5.10.3.105 wlan_get_sta_tx_power()

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

Get Station interface transmit power

Parameters

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

Returns

WM_SUCCESS if successful.
-WM_FAIL if unsuccessful.

5.10.3.106 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.

5.10.3.107 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.

5.10.3.108 wlan_get_wlan_region_code()

```
const char* wlan_get_wlan_region_code (
    void )
```

Get wlan region code from tx power config

Returns

wlan region code in string format.

5.10.3.109 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>	0: STA, 1: uAP
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.

5.10.3.110 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>	0: STA, 1: uAP
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.

5.10.3.111 wlan_get_ext_coex_stats()

```
int wlan_get_ext_coex_stats (
    wlan_ext_coex_stats_t * ext_coex_stats )
```

Get External Radio Coex statistics.

Parameters

out	<i>ext_coex_stats</i>	A pointer to structure to get coex statistics.
-----	-----------------------	--

Returns

WM_SUCCESS if successful.
 -WM_FAIL if unsuccessful.

5.10.3.112 wlan_set_ext_coex_config()

```
int wlan_set_ext_coex_config (
    const wlan_ext_coex_config_t ext_coex_config )
```

Set External Radio Coex configuration.

Parameters

in	<i>ext_coex_config</i>	to apply coex configuration.
----	------------------------	------------------------------

Returns

IE index if successful.
 -WM_FAIL if unsuccessful.

5.10.3.113 wlan_clear_mgmt_ie()

```
int wlan_clear_mgmt_ie (
    enum wlan_bss_type bss_type,
    IEEEtypes_ElementId_t index,
    int mgmt_bitmap_index )
```

Clear Management IE for given BSS type (interface) and index.

Parameters

in	<i>bss_type</i>	0: STA, 1: uAP
in	<i>index</i>	IE index.
in	<i>mgmt_bitmap_index</i>	mgmt bitmap index.

Returns

WM_SUCCESS if successful.
 -WM_FAIL if unsuccessful.

5.10.3.114 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.

5.10.3.115 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.

5.10.3.116 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.

5.10.3.117 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 bss_type 0: STA, 1: uAP
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.

5.10.3.118 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.

5.10.3.119 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.

5.10.3.120 wlan_set_region_power_cfg()

```
int wlan_set_region_power_cfg (
    const t_u8 * data,
    t_u16 len )
```

Set the compressed Tx PWR Limit configuration.

Parameters

in	<i>data</i>	A pointer to TX PWR Limit configuration.
in	<i>len</i>	Length of TX PWR Limit configuration.

Returns

WM_SUCCESS on success, error otherwise.

5.10.3.121 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	chanlist	A pointer to wlan_chanlist_t Channel List configuration.
in	txpwrlimit	A pointer to wlan_txpwrlimit_t TX PWR Limit configuration.

Returns

WM_SUCCESS on success, error otherwise.

5.10.3.122 wlan_set_chanlist()

```
int wlan_set_chanlist (
    wlan_chanlist_t * chanlist )
```

Set the Channel List configuration.

Parameters

in	chanlist	A pointer to wlan_chanlist_t 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.

5.10.3.123 wlan_get_chanlist()

```
int wlan_get_chanlist (
    wlan_chanlist_t * chanlist )
```

Get the Channel List configuration.

Parameters

out	chanlist	A pointer to wlan_chanlist_t Channel List configuration.
-----	----------	--

Returns

WM_SUCCESS on success, error otherwise.

Note

The [wlan_chanlist_t](#) struct allocates memory for a maximum of 54 channels.

5.10.3.124 wlan_set_txpwrlimit()

```
int wlan_set_txpwrlimit (
    wlan_txpwrlimit_t * txpwrlimit )
```

Set the TRPC channel configuration.

Parameters

in	txpwrlimit	A pointer to wlan_txpwrlimit_t TX PWR Limit configuration.
----	------------	--

Returns

WM_SUCCESS on success, error otherwise.

5.10.3.125 wlan_get_txpwrlimit()

```
int wlan_get_txpwrlimit (
    wifi_SubBand_t subband,
    wifi_txpwrlimit_t * txpwrlimit )
```

Get the TRPC channel configuration.

Parameters

in	<i>subband</i>	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	<i>txpwrlimit</i>	A pointer to wlan_txpwrlimit_t 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.

5.10.3.126 `wlan_auto_reconnect_enable()`

```
int wlan_auto_reconnect_enable (
    wlan_auto_reconnect_config_t auto_reconnect_config )
```

Enable Auto Reconnect feature in WLAN firmware.

Parameters

in	<i>auto_reconnect_config</i>	Auto Reconnect configuration structure holding following parameters: <ol style="list-style-type: none"> 1. reconnect counter(0x1-0xff) - The number of times the WLAN firmware retries connection attempt with AP. The value 0xff means retry forever. (default 0xff). 2. reconnect interval(0x0-0xff) - Time gap in seconds between each connection attempt (default 10). 3. flags - Bit 0: Set to 1: Firmware should report link-loss to host if AP rejects authentication/association while reconnecting. Set to 0: Default behaviour: Firmware does not report link-loss to host on AP rejection and continues internally. Bit 1-15: Reserved.
----	------------------------------	---

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.127 wlan_auto_reconnect_disable()

```
int wlan_auto_reconnect_disable (  
    void )
```

Disable Auto Reconnect feature in WLAN firmware.

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.128 wlan_get_auto_reconnect_config()

```
int wlan_get_auto_reconnect_config (  
    wlan_auto_reconnect_config_t * auto_reconnect_config )
```

Get Auto Reconnect configuration from WLAN firmware.

Parameters

out	<i>auto_reconnect_config</i>	Auto Reconnect configuration structure where response from WLAN firmware will get stored.
-----	------------------------------	---

Returns

WM_SUCCESS if operation is successful.
-WM_E_INVALID if *auto_reconnect_config* is not valid.
-WM_FAIL if command fails.

5.10.3.129 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	<i>reassoc_control</i>	Reassociation enable/disable
----	------------------------	------------------------------

5.10.3.130 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.

5.10.3.131 wlan_uap_set_bandwidth()

```
int wlan_uap_set_bandwidth (
    const uint8_t bandwidth )
```

API to set the bandwidth of uAP

Parameters

in	<i>bandwidth</i>	Wi-Fi AP Bandwidth (20MHz/40MHz) 1: 20 MHz 2: 40 MHz 3: 80 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.

5.10.3.132 wlan_uap_get_bandwidth()

```
int wlan_uap_get_bandwidth (
    uint8_t * bandwidth )
```

API to Get the bandwidth of uAP

Confidential

Parameters

out	<i>bandwidth</i>	Wi-Fi AP Bandwidth (20MHz/40MHz) 1: 20 MHz 2: 40 MHz 3: 80 MHz
-----	------------------	--

Returns

WM_SUCCESS if successful otherwise failure.
-WM_FAIL if command fails.

Note

Please call this API before calling uAP start API.

5.10.3.133 wlan_uap_set_hidden_ssid()

```
int wlan_uap_set_hidden_ssid (
    const t_u8 hidden_ssid )
```

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>hidden_ssid</i>	Hidden SSID control hidden_ssid=0: broadcast SSID in beacons. hidden_ssid=1: send empty SSID (length=0) in beacon. hidden_ssid=2: clear SSID (ACSI 0), but keep the original length
----	--------------------	---

Returns

WM_SUCCESS if successful otherwise failure.
-WM_FAIL if command fails.

Note

Please call this API before calling uAP start API.

5.10.3.134 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.

5.10.3.135 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.

5.10.3.136 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>	<p>- This is a bitmap and should be used as following</p> <p>Bit 15: L Sig TxOP protection - reserved, set to 0</p> <p>Bit 14: 40 MHz intolerant - reserved, set to 0</p> <p>Bit 13: PSMP - reserved, set to 0</p> <p>Bit 12: DSSS Cck40MHz mode</p> <p>Bit 11: Maximal AMSDU size - reserved, set to 0</p> <p>Bit 10: Delayed BA - reserved, set to 0</p> <p>Bits 9:8: Rx STBC - reserved, set to 0</p> <p>Bit 7: Tx STBC - reserved, set to 0</p> <p>Bit 6: Short GI 40 MHz</p> <p>Bit 5: Short GI 20 MHz</p> <p>Bit 4: GF preamble</p> <p>Bits 3:2: MIMO power save - reserved, set to 0</p> <p>Bit 1: SuppChanWidth - set to 0 for 2.4 GHz band</p> <p>Bit 0: LDPC coding - reserved, set to 0</p>
----	--------------------	--

Note

Please call this API before calling uAP start API.

5.10.3.137 wlan_uap_set_httxcf()

```
void wlan_uap_set_httxcf (
    unsigned short httxcf )
```

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>httxcf</i>	<p>This is a bitmap and should be used as following</p> <ul style="list-style-type: none"> 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 <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>
----	---------------	--

Note

Please call this API before calling uAP start API.

5.10.3.138 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.

5.10.3.139 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.

5.10.3.140 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.

5.10.3.141 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.

5.10.3.142 wlan_uap_ampdu_tx_enable()

```
void wlan_uap_ampdu_tx_enable (  
    void )
```

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

Note

By default the uap AMPDU TX support is on if configuration option is enabled in defconfig.

5.10.3.143 wlan_uap_ampdu_tx_disable()

```
void wlan_uap_ampdu_tx_disable (  
    void )
```

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

Note

By default the uap AMPDU RX support is on if configuration option is enabled in defconfig.

5.10.3.144 wlan_uap_ampdu_rx_enable()

```
void wlan_uap_ampdu_rx_enable (
    void )
```

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

5.10.3.145 wlan_uap_ampdu_rx_disable()

```
void wlan_uap_ampdu_rx_disable (
    void )
```

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

5.10.3.146 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.

5.10.3.147 wlan_set_rts()

```
int wlan_set_rts (
    int rts )
```

Set the rts threshold of sta in WLAN firmware.

Parameters

in	<i>rts</i>	the value of rts threshold configuration.
----	------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.148 wlan_set_uap_rts()

```
int wlan_set_uap_rts (  
    int rts )
```

Set the rts threshold of uap in WLAN firmware.

Parameters

in	<i>rts</i>	the value of rts threshold configuration.
----	------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.149 wlan_set_frag()

```
int wlan_set_frag (  
    int frag )
```

Set the fragment threshold of sta in WLAN firmware.

Parameters

in	<i>frag</i>	the value of fragment threshold configuration.
----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.150 wlan_set_uap_frag()

```
int wlan_set_uap_frag (  
    int frag )
```

Set the fragment threshold of uap in WLAN firmware.

Parameters

in	<i>frag</i>	the value of fragment threshold configuration.
----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.151 wlan_set_sta_mac_filter()

```
int wlan_set_sta_mac_filter (
    int filter_mode,
    int mac_count,
    unsigned char * mac_addr )
```

Set the sta mac filter in Wi-Fi firmware.

Parameters

in	<i>filter_mode</i>	channel filter mode (disable/white/black list)
in	<i>mac_count</i>	the count of mac list
in	<i>mac_addr</i>	the pointer to mac address list

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.152 print_mac()

```
static void print_mac (
    const char * mac ) [inline], [static]
```

5.10.3.153 wlan_set_rf_test_mode()

```
int wlan_set_rf_test_mode (
    void )
```

Set the RF Test Mode on in Wi-Fi firmware.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.154 wlan_unset_rf_test_mode()

```
int wlan_unset_rf_test_mode (  
    void )
```

UnSet the RF Test Mode on in Wi-Fi firmware.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.155 wlan_set_rf_channel()

```
int wlan_set_rf_channel (  
    const uint8_t channel )
```

Set the RF Channel in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>channel</i>	The channel number to be set in Wi-Fi firmware.
----	----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.156 wlan_set_rf_radio_mode()

```
int wlan_set_rf_radio_mode (  
    const uint8_t mode )
```

Set the RF radio mode in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>mode</i>	The radio mode number to be set in Wi-Fi firmware.
----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.157 wlan_get_rf_channel()

```
int wlan_get_rf_channel (
    uint8_t * channel )
```

Get the RF Channel from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>channel</i>	A Pointer to a variable where channel number to get.
-----	----------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.158 wlan_get_rf_radio_mode()

```
int wlan_get_rf_radio_mode (
    uint8_t * mode )
```

Get the RF Radio mode from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>mode</i>	A Pointer to a variable where radio mode number to get.
-----	-------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.159 wlan_set_rf_band()

```
int wlan_set_rf_band (
    const uint8_t band )
```

Set the RF Band in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>band</i>	The bandwidth to be set in Wi-Fi firmware.
----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.160 wlan_get_rf_band()

```
int wlan_get_rf_band (
    uint8_t * band )
```

Get the RF Band from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>band</i>	A Pointer to a variable where RF Band is to be stored.
-----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.161 wlan_set_rf_bandwidth()

```
int wlan_set_rf_bandwidth (
    const uint8_t bandwidth )
```

Set the RF Bandwidth in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>bandwidth</i>	The bandwidth to be set in Wi-Fi firmware.
----	------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.162 wlan_get_rf_bandwidth()

```
int wlan_get_rf_bandwidth (
    uint8_t * bandwidth )
```

Get the RF Bandwidth from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>bandwidth</i>	A Pointer to a variable where bandwidth to get.
-----	------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.163 wlan_get_rf_per()

```
int wlan_get_rf_per (
    uint32_t * rx_tot_pkt_count,
    uint32_t * rx_mcast_bcast_count,
    uint32_t * rx_pkt_fcs_error )
```

Get the RF PER from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>rx_tot_pkt_count</i>	A Pointer to a variable where Rx Total packet count to get.
out	<i>rx_mcast_bcast_count</i>	A Pointer to a variable where Rx Total Multicast/Broadcast packet count to get.
out	<i>rx_pkt_fcs_error</i>	A Pointer to a variable where Rx Total packet count with FCS error to get.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.164 wlan_set_rf_tx_cont_mode()

```
int wlan_set_rf_tx_cont_mode (
    const uint32_t enable_tx,
    const uint32_t cw_mode,
    const uint32_t payload_pattern,
    const uint32_t cs_mode,
    const uint32_t act_sub_ch,
    const uint32_t tx_rate )
```

Set the RF Tx continuous mode in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>enable_tx</i>	Enable Tx.
in	<i>cw_mode</i>	Set CW Mode.
in	<i>payload_pattern</i>	Set Payload Pattern.
in	<i>cs_mode</i>	Set CS Mode.
in	<i>act_sub_ch</i>	Act Sub Ch
in	<i>tx_rate</i>	Set Tx Rate.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.165 wlan_cfg_rf_he_tb_tx()

```
int wlan_cfg_rf_he_tb_tx (
    uint16_t enable,
    uint16_t qnum,
    uint16_t aid,
    uint16_t axq_mu_timer,
    int16_t tx_power )
```

Set the RF HE TB TX in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>enable</i>	Enable/Disable trigger response mode
in	<i>qnum</i>	AXQ to be used for the trigger response frame
in	<i>aid</i>	AID of the peer to which response is to be generated
in	<i>axq_mu_timer</i>	MU timer for the AXQ on which response is sent
in	<i>tx_power</i>	TxPwr to be configured for the response

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.166 wlan_rf_trigger_frame_cfg()

```
int wlan_rf_trigger_frame_cfg (
    uint32_t Enable_tx,
    uint32_t Standalone_hetb,
    uint8_t FRAME_CTRL_TYPE,
    uint8_t FRAME_CTRL_SUBTYPE,
    uint16_t FRAME_DURATION,
    uint64_t TriggerType,
    uint64_t ULen,
    uint64_t MoreTF,
    uint64_t CSRequired,
    uint64_t ULBw,
    uint64_t LTFTType,
    uint64_t LTFTMode,
    uint64_t LTFSymbol,
    uint64_t ULSTBC,
    uint64_t LdpcESS,
    uint64_t ApTxPwr,
    uint64_t PreFecPadFct,
    uint64_t PeDisambig,
    uint64_t SpatialReuse,
    uint64_t Doppler,
    uint64_t HeSig2,
    uint32_t AID12,
    uint32_t RUAllocReg,
    uint32_t RUAlloc,
    uint32_t ULCodingType,
    uint32_t ULMCS,
    uint32_t ULDCM,
    uint32_t SSAlloc,
    uint8_t ULTargetRSSI,
    uint8_t MPDU_MU_SF,
    uint8_t TID_AL,
    uint8_t AC_PL,
    uint8_t Pref_AC )
```

Set the RF Trigger Frame Config in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>Enable_tx</i>	Enable or Disable trigger frame transmission.
in	<i>Standalone_hetb</i>	Enable or Disable Standalone HE TB support.
in	<i>FRAME_CTRL_TYPE</i>	Frame control type.
in	<i>FRAME_CTRL_SUBTYPE</i>	Frame control subtype.
in	<i>FRAME_DURATION</i>	Max Duration time.
in	<i>TriggerType</i>	Identifies the Trigger frame variant and its encoding.
in	<i>Uilen</i>	Indicates the value of the L-SIG LENGTH field of the solicited HE TB PPDU.
in	<i>MoreTF</i>	Indicates whether a subsequent Trigger frame is scheduled for transmission.
in	<i>CSRequired</i>	Required to use ED to sense the medium and to consider the medium state and the NAV in determining whether to respond.
in	<i>UIBw</i>	Indicates the bandwidth in the HE-SIG-A field of the HE TB PPDU.
in	<i>LTFTType</i>	Indicates the LTF type of the HE TB PPDU response.
in	<i>LTFMode</i>	Indicates the LTF mode for an HE TB PPDU.
in	<i>LTFSymbol</i>	Indicates the number of LTF symbols present in the HE TB PPDU.
in	<i>UISTBC</i>	Indicates the status of STBC encoding for the solicited HE TB PPDU.
in	<i>LdpcESS</i>	Indicates the status of the LDPC extra symbol segment.
in	<i>ApTxPwr</i>	Indicates the AP's combined transmit power at the transmit antenna connector of all the antennas used to transmit the triggering PPDU.
in	<i>PreFecPadFct</i>	Indicates the pre-FEC padding factor.
in	<i>PeDisambig</i>	Indicates PE disambiguity.
in	<i>SpatialReuse</i>	Carries the values to be included in the Spatial Reuse fields in the HE-SIG-A field of the solicited HE TB PPDU.
in	<i>Doppler</i>	Indicate that a midamble is present in the HE TB PPDU.
in	<i>HeSig2</i>	Carries the value to be included in the Reserved field in the HE-SIG-A2 subfield of the solicited HE TB PPDU.
in	<i>AID12</i>	If set to 0 allocates one or more contiguous RA-RUs for associated STAs.
in	<i>RUAllocReg</i>	RUAllocReg.
in	<i>RUAlloc</i>	Identifies the size and the location of the RU.
in	<i>UICodingType</i>	Indicates the code type of the solicited HE TB PPDU.
in	<i>UIMCS</i>	Indicates the HE-MCS of the solicited HE TB PPDU.
in	<i>UIDCM</i>	Indicates DCM of the solicited HE TB PPDU.
in	<i>SSAlloc</i>	Indicates the spatial streams of the solicited HE TB PPDU.
in	<i>UITargetRSSI</i>	Indicates the expected receive signal power.
in	<i>MPDU_MU_SF</i>	Used for calculating the value by which the minimum MPDU start spacing is multiplied.
in	<i>TID_AL</i>	Indicates the MPDUs allowed in an A-MPDU carried in the HE TB PPDU and the maximum number of TIDs that can be aggregated by the STA in the A-MPDU.
in	<i>AC_PL</i>	Reserved.
in	<i>Pref_AC</i>	Indicates the lowest AC that is recommended for aggregation of MPDUs in the A-MPDU contained in the HE TB PPDU sent as a response to the Trigger frame.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.167 wlan_set_rf_tx_antenna()

```
int wlan_set_rf_tx_antenna (
    const uint8_t antenna )
```

Set the RF Tx Antenna in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>antenna</i>	The Tx antenna to be set in Wi-Fi firmware.
----	----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.168 wlan_get_rf_tx_antenna()

```
int wlan_get_rf_tx_antenna (
    uint8_t * antenna )
```

Get the RF Tx Antenna from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>antenna</i>	A Pointer to a variable where Tx antenna is to be stored.
-----	----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.169 wlan_set_rf_rx_antenna()

```
int wlan_set_rf_rx_antenna (
    const uint8_t antenna )
```

Set the RF Rx Antenna in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>antenna</i>	The Rx antenna to be set in Wi-Fi firmware.
----	----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.170 wlan_get_rf_rx_antenna()

```
int wlan_get_rf_rx_antenna (
    uint8_t * antenna )
```

Get the RF Rx Antenna from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>antenna</i>	A Pointer to a variable where Rx antenna is to be stored.
-----	----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.171 wlan_set_rf_tx_power()

```
int wlan_set_rf_tx_power (
    const uint32_t power,
    const uint8_t mod,
    const uint8_t path_id )
```

Set the RF Tx Power in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>power</i>	The RF Tx Power to be set in Wi-Fi firmware. For RW610, 20M bandwidth max linear output power is 20db per data sheet.
in	<i>mod</i>	The modulation to be set in Wi-Fi firmware.
in	<i>path_id</i>	The Path ID to be set in Wi-Fi firmware.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.172 wlan_set_rf_tx_frame()

```
int wlan_set_rf_tx_frame (
    const uint32_t enable,
    const uint32_t data_rate,
    const uint32_t frame_pattern,
    const uint32_t frame_length,
    const uint16_t adjust_burst_sifs,
    const uint32_t burst_sifs_in_us,
    const uint32_t short_preamble,
    const uint32_t act_sub_ch,
    const uint32_t short_gi,
    const uint32_t adv_coding,
    const uint32_t tx_bf,
    const uint32_t gf_mode,
    const uint32_t stbc,
    const uint8_t * bssid )
```

Set the RF Tx Frame in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>enable</i>	Enable/Disable RF Tx Frame
in	<i>data_rate</i>	Rate Index corresponding to legacy/HT/VHT rates
in	<i>frame_pattern</i>	Payload Pattern
in	<i>frame_length</i>	Payload Length
in	<i>adjust_burst_sifs</i>	Enabl/Disable Adjust Burst SIFS3 Gap
in	<i>burst_sifs_in_us</i>	Burst SIFS in us
in	<i>short_preamble</i>	Enable/Disable Short Preamble
in	<i>act_sub_ch</i>	Enable/Disable Active SubChannel
in	<i>short_gi</i>	Short Guard Interval
in	<i>adv_coding</i>	Enable/Disable Adv Coding
in	<i>tx_bf</i>	Enable/Disable Beamforming
in	<i>gf_mode</i>	Enable/Disable GreenField Mode
in	<i>stbc</i>	Enable/Disable STBC
in	<i>bssid</i>	BSSID

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.173 wlan_set_rf_otp_mac_addr()

```
int wlan_set_rf_otp_mac_addr (
    uint8_t * mac )
```

Set the RF OTP MAC address in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	mac	A Pointer to a variable where OTP MAC address is to be stored.
----	-----	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.174 wlan_get_rf_otp_mac_addr()

```
int wlan_get_rf_otp_mac_addr (
    uint8_t * mac )
```

Get the RF OTP MAC address from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	mac	A Pointer to a variable where OTP MAC address is to be stored.
-----	-----	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.175 wlan_set_rf_otp_cal_data()

```
int wlan_set_rf_otp_cal_data (
    const uint8_t * cal_data,
    uint32_t cal_data_len )
```

Set the RF OTP cal data in Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

in	<i>cal_data</i>	A Pointer to a variable where OTP cal data is to be stored.
in	<i>cal_data_len</i>	The length of OTP cal data.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.176 wlan_get_rf_otp_cal_data()

```
int wlan_get_rf_otp_cal_data (
    uint8_t * cal_data )
```

Get the RF OTP cal data from Wi-Fi firmware.

Note

Please call [wlan_set_rf_test_mode](#) API before using this API.

Parameters

out	<i>cal_data</i>	A Pointer to a variable where OTP cal data is to be stored.
-----	-----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.177 wlan_register_fw_dump_cb()

```
void wlan_register_fw_dump_cb (
    void(*) (void) wlan_usb_init_cb,
    int(*) () wlan_usb_mount_cb,
    int(*) (char *test_file_name) wlan_usb_file_open_cb,
    int(*) (uint8_t *data, size_t data_len) wlan_usb_file_write_cb,
    int(*) () wlan_usb_file_close_cb )
```

This function registers callbacks which are used to generate FW Dump on USB device.

Parameters

in	<i>wlan_usb_init_cb</i>	Callback to initialize usb device.
in	<i>wlan_usb_mount_cb</i>	Callback to mount usb device.
in	<i>wlan_usb_file_open_cb</i>	Callback to open file on usb device for FW dump.
in	<i>wlan_usb_file_write_cb</i>	Callback to write FW dump data to opened file.
in	<i>wlan_usb_file_close_cb</i>	Callback to close FW dump file.

5.10.3.178 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.179 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.180 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.181 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.182 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.183 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.184 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.185 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.186 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.187 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.
 -WM_E_PERM if not supported.
 -WM_FAIL if 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.

5.10.3.188 wlan_send_hostcmd()

```
int wlan_send_hostcmd (
    const 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>host_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.

5.10.3.189 wlan_send_debug_htc()

```
int wlan_send_debug_htc (
    const uint8_t count,
    const uint8_t vht,
    const uint8_t he,
    const uint8_t rxNss,
    const uint8_t channelWidth,
    const uint8_t ulMuDisable,
    const uint8_t txNSTS,
    const uint8_t erSuDisable,
    const uint8_t dlResoundRecomm,
    const uint8_t ulMuDataDisable )
```

This function is used to set HTC parameter.

Parameters

in	<i>count</i>	
in	<i>vht</i>	
in	<i>he</i>	
in	<i>rxNss</i>	
in	<i>channelWidth</i>	
in	<i>ulMuDisable</i>	
in	<i>txNSTS</i>	
in	<i>erSuDisable</i>	
in	<i>dlResoundRecomm</i>	
in	<i>ulMuDataDisable</i>	

Returns

WM_SUCCESS if operation is successful, otherwise failure

5.10.3.190 wlan_enable_disable_htc()

```
int wlan_enable_disable_htc (
    uint8_t option )
```

This function is used to enable/disable HTC.

Parameters

in	<i>option</i>	1 => Enable; 0 => Disable
----	---------------	---------------------------

Returns

WM_SUCCESS if operation is successful, otherwise failure

5.10.3.191 wlan_set_11ax_tx_omi()

```
int wlan_set_11ax_tx_omi (
    const t_u8 interface,
    const t_u16 tx_omi,
    const t_u8 tx_option,
    const t_u8 num_data_pkts )
```

Use this API to set the set 11AX Tx OMI.

Parameters

in	<i>interface</i>	Interface type STA or uAP.
in	<i>tx_omi</i>	value to be sent to Firmware
in	<i>tx_option</i>	value to be sent to Firmware 1: send OMI in QoS data.
in	<i>num_data_pkts</i>	value to be sent to Firmware num_data_pkts is applied only if OMI is sent in QoS data frame. It specifies the number of consecutive data frames containing the OMI. Minimum value is 1 Maximum value is 16

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.192 wlan_set_11ax_tol_time()

```
int wlan_set_11ax_tol_time (
    const t_u32 tol_time )
```

Set 802_11 AX OBSS Narrow Bandwidth RU Tolerance Time In uplink transmission, AP sends a trigger frame to all the stations that will be involved in the upcoming transmission, and then these stations transmit Trigger-based(TB) PPDU in response to the trigger frame. If STA connects to AP which channel is set to 100,STA doesn't support 26 tones RU. The API should be called when station is in disconnected state.

Parameters

in	<i>tol_time</i>	Valid range [1...3600] tolerance time is in unit of seconds. STA periodically check AP's beacon for ext cap bit79 (OBSS Narrow bandwidth RU in ofdma tolerance support) and set 20 tone RU tolerance time if ext cap bit79 is not set
----	-----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.193 wlan_set_11ax_rutxpowerlimit()

```
int wlan_set_11ax_rutxpowerlimit (
    const void * rutx_pwr_cfg,
    uint32_t rutx_pwr_cfg_len )
```

Use this API to set the RU tx power limit.

Parameters

in	<i>rutx_pwr_cfg</i>	11AX rutxpwr of sub-bands to be sent to Firmware.
in	<i>rutx_pwr_cfg_len</i>	Size of rutx_pwr_cfg buffer.

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.194 wlan_set_11ax_rutxpowerlimit_legacy()

```
int wlan_set_11ax_rutxpowerlimit_legacy (
    const wlan_rutxpwrlimit_t * ru_pwr_cfg )
```

Use this API to set the RU tx power limit by channel based approach.

Parameters

in	<i>ru_pwr_cfg</i>	11AX rutxpwr of channels to be sent to Firmware.
----	-------------------	--

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

5.10.3.195 wlan_get_11ax_rutxpowerlimit_legacy()

```
int wlan_get_11ax_rutxpowerlimit_legacy (
    wlan_rutxpwrlimit_t * ru_pwr_cfg )
```

Use this API to get the RU tx power limit by channel based approach.

Parameters

in	<i>ru_pwr_cfg</i>	11AX rtxpwr of channels to be get from Firmware
----	-------------------	---

Returns

WM_SUCCESS if operation is successful.
 -WM_FAIL if command fails.

5.10.3.196 wlan_set_11ax_cfg()

```
int wlan_set_11ax_cfg (
    wlan_11ax_config_t * ax_config )
```

Set 11ax config params

Parameters

in, out	<i>ax_config</i>	11AX config parameters to be sent to Firmware
---------	------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.197 wlan_get_11ax_cfg()

```
uint8_t* wlan_get_11ax_cfg ( )
```

Get default 11ax config params

Returns

11AX config parameters default array.

5.10.3.198 wlan_set_btwt_cfg()

```
int wlan_set_btwt_cfg (
    const wlan_btwt_config_t * btwt_config )
```

Set btwt config params

Parameters

in	<i>btwt_config</i>	Broadcast TWT Setup parameters to be sent to Firmware
----	--------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.199 wlan_get_btwt_cfg()

```
uint8_t* wlan_get_btwt_cfg ( )
```

Get btwt config params

Returns

Broadcast TWT Setup parameters default config array.

5.10.3.200 wlan_set_twt_setup_cfg()

```
int wlan_set_twt_setup_cfg (
    const wlan_twt_setup_config_t * twt_setup )
```

Set twt setup config params

Parameters

in	<i>twt_setup</i>	TWT Setup parameters to be sent to Firmware
----	------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.201 wlan_get_twt_setup_cfg()

```
uint8_t* wlan_get_twt_setup_cfg ( )
```

Get twt setup config params

Returns

TWT Setup parameters default array.

5.10.3.202 wlan_set_twt_teardown_cfg()

```
int wlan_set_twt_teardown_cfg (
    const wlan_twt_teardown_config_t * teardown_config )
```

Set twt teardown config params

Parameters

in	<i>teardown_config</i>	TWT Teardown parameters sent to Firmware
----	------------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.203 wlan_get_twt_teardown_cfg()

```
uint8_t* wlan_get_twt_teardown_cfg ( )
```

Get twt teardown config params

Returns

TWT Teardown parameters default array

5.10.3.204 wlan_get_twt_report()

```
int wlan_get_twt_report (
    wlan_twt_report_t * twt_report )
```

Get twt report

Parameters

out	<i>twt_report</i>	TWT Report parameter.
-----	-------------------	-----------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.205 wlan_twt_information()

```
int wlan_twt_information (
    wlan_twt_information_t * twt_information )
```

Twt information

Parameters

out	<i>twt_information</i>	TWT information.
-----	------------------------	------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.206 wlan_set_clocksync_cfg()

```
int wlan_set_clocksync_cfg (
    const wlan_clock_sync_gpio_tsf_t * tsf_latch )
```

Set Clock Sync GPIO based TSF

Parameters

in	<i>tsf_latch</i>	Clock Sync TSF latch parameters to be sent to Firmware
----	------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.207 wlan_get_tsf_info()

```
int wlan_get_tsf_info (
    wlan_tsf_info_t * tsf_info )
```

Get TSF info from firmware using GPIO latch

Parameters

out	<i>tsf_info</i>	TSF info parameter received from Firmware
-----	-----------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.208 wlan_show_os_mem_stat()

```
void wlan_show_os_mem_stat ( )
```

Show os mem alloc and free info.

5.10.3.209 wlan_ft_roam()

```
int wlan_ft_roam (
    const t_u8 * bssid,
    const t_u8 channel )
```

Start FT roaming : This API is used to initiate fast BSS transition based roaming.

Parameters

in	<i>bssid</i>	BSSID of AP to roam
in	<i>channel</i>	Channel of AP to roam

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.210 wlan_rx_mgmt_indication()

```
int wlan_rx_mgmt_indication (
    const enum wlan_bss_type bss_type,
    const uint32_t mgmt_subtype_mask,
    int (*)(const enum wlan_bss_type bss_type, const wlan_mgmt_frame_t *frame, const
size_t len) rx_mgmt_callback )
```

This API can be used to start/stop the management frame forwards to host through datapath.

Parameters

in	<i>bss_type</i>	The interface from which management frame needs to be collected 0: STA, 1: uAP
in	<i>mgmt_subtype_mask</i>	Management Subtype Mask If Bit X is set in mask, it means that IEEE Management Frame SubType X is to be filtered and passed through to host. Bit Description [31:14] Reserved [13] Action frame [12:9] Reserved [8] Beacon [7:6] Reserved [5] Probe response [4] Probe request [3] Reassociation response [2] Reassociation request [1] Association response [0] Association request Support multiple bits set. 0 = stop forward frame 1 = start forward frame
in	<i>rx_mgmt_callback</i>	The receive callback where the received management frames are passed.

Returns

WM_SUCCESS if operation is successful.
-WM_FAIL if command fails.

Note

Pass Management Subtype Mask all zero to disable all the management frame forward to host.

5.10.3.211 wlan_wmm_tx_stats_dump()

```
void wlan_wmm_tx_stats_dump (
    int bss_type )
```

5.10.3.212 wlan_set_scan_channel_gap()

```
void wlan_set_scan_channel_gap (
    unsigned scan_chan_gap )
```

Set scan channel gap.

Parameters

in	<i>scan_chan_gap</i>	Time gap to be used between two consecutive channels scan.
----	----------------------	--

5.10.3.213 wlan_host_11k_cfg()

```
int wlan_host_11k_cfg (
    int enable_11k )
```

enable/disable host 11k feature

Parameters

in	<i>enable_11k</i>	the value of 11k configuration.
----	-------------------	---------------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.214 wlan_get_host_11k_status()

```
bool wlan_get_host_11k_status ( )
```

5.10.3.215 wlan_host_11k_neighbor_req()

```
int wlan_host_11k_neighbor_req (
    const char * ssid )
```

host send neighbor report request

Parameters

in	ssid	the SSID for neighbor report
----	------	------------------------------

Note

ssid parameter is optional

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.216 wlan_host_11v_bss_trans_query()

```
int wlan_host_11v_bss_trans_query (
    t_u8 query_reason )
```

host send bss transition management query

Parameters

in	query_reason	BTM request query reason code
----	--------------	-------------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.217 wlan_set_okc()

```
int wlan_set_okc (
    t_u8 okc )
```


Opportunistic Key Caching (also known as Proactive Key Caching) default This parameter can be used to set the default behavior for the `proactive_key_caching` parameter. By default, OKC is disabled unless enabled with the global `okc=1` parameter or with the per-network `pkc(proactive_key_caching)=1` parameter. With `okc=1`, OKC is enabled by default, but can be disabled with per-network `pkc(proactive_key_caching)=0` parameter.

Parameters

in	<i>okc</i>	Enable Opportunistic Key Caching
----	------------	----------------------------------

0 = Disable OKC (default) 1 = Enable OKC

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.218 wlan_pmksa_list()

```
int wlan_pmksa_list (
    char * buf,
    size_t buflen )
```

Dump text list of entries in PMKSA cache

Parameters

out	<i>buf</i>	Buffer to save PMKSA cache text list
in	<i>buflen</i>	length of the buffer

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.219 wlan_pmksa_flush()

```
int wlan_pmksa_flush ( )
```

Flush PTKSA cache entries

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.220 wlan_set_scan_interval()

```
int wlan_set_scan_interval (
    int scan_int )
```

Set wpa supplicant scan interval in seconds

Parameters

in	<i>scan_int</i>	Scan interval in seconds
----	-----------------	--------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.221 wlan_tx_ampdu_prot_mode()

```
int wlan_tx_ampdu_prot_mode (
    tx_ampdu_prot_mode_para * prot_mode,
    t_u16 action )
```

Set/Get Tx ampdu prot mode.

Parameters

in, out	<i>prot_mode</i>	Tx ampdu prot mode
in	<i>action</i>	Command action

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.222 wlan_mef_set_auto_arp()

```
int wlan_mef_set_auto_arp (
    t_u8 mef_action )
```

This function set auto ARP configuration.

Parameters

in	<i>mef_action</i>	To be 0—discard and not wake host, 1—discard and wake host 3—allow and wake host.
----	-------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.223 wlan_mef_set_auto_ping()

```
int wlan_mef_set_auto_ping (
    t_u8 mef_action )
```

This function set auto ping configuration.

Parameters

in	<i>mef_action</i>	To be 0—discard and not wake host, 1—discard and wake host 3—allow and wake host.
----	-------------------	---

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.224 wlan_config_mef()

```
int wlan_config_mef (
    int type,
    t_u8 mef_action )
```

This function set/delete mef entries configuration.

Parameters

in	<i>type</i>	MEF type: MEF_TYPE_DELETE, MEF_TYPE_AUTO_PING, MEF_TYPE_AUTO_ARP
in	<i>mef_action</i>	To be 0—discard and not wake host, 1—discard and wake host 3—allow and wake host.

Returns

WM_SUCCESS if the call was successful.
-WM_FAIL if failed.

5.10.3.225 wlan_set_ipv6_ns_mef()

```
int wlan_set_ipv6_ns_mef (
    t_u8 mef_action )
```

Use this API to enable IPv6 Neighbor Solicitation offload in Wi-Fi firmware

Parameters

in	<i>mef_action</i>	0—discard and not wake host, 1—discard and wake host 3—allow and wake host.
----	-------------------	---

Returns

WM_SUCCESS if operation is successful.
 -WM_FAIL if command fails.

5.10.3.226 wlan_csi_cfg()

```
int wlan_csi_cfg (
    wlan_csi_config_params_t * csi_params )
```

Send the csi config parameter to FW.

Parameters

in	<i>csi_params</i>	Csi config parameter
----	-------------------	----------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.227 wlan_register_csi_user_callback()

```
int wlan_register_csi_user_callback (
    int(*) (void *buffer, size_t len) csi_data_recv_callback )
```

This function registers callback which are used to deliver CSI data to user.

Parameters

in	<i>csi_data_recv_callback</i>	Callback to deliver CSI data and max data length is 768 bytes. Pls save data as soon as possible in callback Type of callback return vale is int. Memory layout of buffer: size(byte) items 2 buffer len[bit 0:12] 2 CSI signature, 0xABCD fixed 4 User defined HeaderID 2 Packet info 2 Frame control field for the received packet 8 Timestamp when packet received 6 Received Packet Destination MAC Address 6 Received Packet Source MAC Address 1 RSSI for antenna A 1 RSSI for antenna B 1 Noise floor for antenna A 1 Noise floor for antenna B 1 Rx signal strength above noise floor 1 Channel 2 user defined Chip ID 4 Reserved 4 CSI data length in DWORDs CSI data
----	-------------------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.228 wlan_unregister_csi_user_callback()

```
int wlan_unregister_csi_user_callback (
    void )
```

This function unregisters callback which are used to deliver CSI data to user.

Returns

WM_SUCCESS if successful

5.10.3.229 wlan_set_rssi_low_threshold()

```
void wlan_set_rssi_low_threshold (
    uint8_t threshold )
```

Use this API to set the RSSI threshold value for low RSSI event subscription. When RSSI falls below this threshold firmware will generate the low RSSI event to driver. This low RSSI event is used when either of CONFIG_11R, CONFIG_11K, CONFIG_11V or CONFIG_ROAMING is enabled. NOTE: By default rssi low threshold is set at -70 dbm

Parameters

in	<i>threshold</i>	Threshold rssi value to be set
----	------------------	--------------------------------

5.10.3.230 wlan_wps_generate_pin()

```
void wlan_wps_generate_pin (
    uint32_t * pin )
```

Generate valid PIN for WPS session.

This function generate PIN for WPS PIN session.

Parameters

in	<i>pin</i>	A pointer to WPS pin to be generated.
----	------------	---------------------------------------

5.10.3.231 wlan_start_wps_pin()

```
int wlan_start_wps_pin (
    const char * pin )
```

Start WPS PIN session.

This function starts WPS PIN session.

Parameters

in	<i>pin</i>	Pin for WPS session.
----	------------	----------------------

Returns

WM_SUCCESS if the pin entered is valid.
-WM_FAIL if invalid pin entered.

5.10.3.232 wlan_start_wps_pbc()

```
int wlan_start_wps_pbc (  
    void )
```

Start WPS PBC session.

This function starts WPS PBC session.

Returns

WM_SUCCESS if successful
-WM_FAIL if invalid pin entered.

5.10.3.233 wlan_wps_cancel()

```
int wlan_wps_cancel (  
    void )
```

Cancel WPS session.

This function cancels ongoing WPS session.

Returns

WM_SUCCESS if successful
-WM_FAIL if invalid pin entered.

5.10.3.234 wlan_start_ap_wps_pin()

```
int wlan_start_ap_wps_pin (  
    const char * pin )
```

Start WPS PIN session.

This function starts AP WPS PIN session.

Parameters

in	<i>pin</i>	Pin for WPS session.
----	------------	----------------------

Returns

WM_SUCCESS if the pin entered is valid.
 -WM_FAIL if invalid pin entered.

5.10.3.235 wlan_start_ap_wps_pbc()

```
int wlan_start_ap_wps_pbc (
    void )
```

Start WPS PBC session.

This function starts AP WPS PBC session.

Returns

WM_SUCCESS if successful
 -WM_FAIL if invalid pin entered.

5.10.3.236 wlan_wps_ap_cancel()

```
int wlan_wps_ap_cancel (
    void )
```

Cancel AP's WPS session.

This function cancels ongoing WPS session.

Returns

WM_SUCCESS if successful
 -WM_FAIL if invalid pin entered.

5.10.3.237 wlan_set_entp_cert_files()

```
int wlan_set_entp_cert_files (
    int cert_type,
    t_u8 * data,
    t_u32 data_len )
```

This function specifies the enterprise certificate file. This function must be used before adding network profile. It will store certificate data in "wlan" global structure. When adding new network profile, it will be get by [wlan_get_entp_cert_files\(\)](#), and put into profile security structure after mbedtls parse.

Parameters

in	<i>cert_type</i>	certificate file type: 1 – FILE_TYPE_ENTP_CA_CERT, 2 – FILE_TYPE_ENTP_CLIENT_CERT, 3 – FILE_TYPE_ENTP_CLIENT_KEY.
in	<i>data</i>	raw data
in	<i>data_len</i>	size of raw data

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.238 wlan_get_entp_cert_files()

```
t_u32 wlan_get_entp_cert_files (
    int cert_type,
    t_u8 ** data )
```

This function get enterprise certificate data from "wlan" global structure *

Parameters

in	<i>cert_type</i>	certificate file type: 1 – FILE_TYPE_ENTP_CA_CERT, 2 – FILE_TYPE_ENTP_CLIENT_CERT, 3 – FILE_TYPE_ENTP_CLIENT_KEY.
in	<i>data</i>	raw data

Returns

size of raw data

5.10.3.239 wlan_free_entp_cert_files()

```
void wlan_free_entp_cert_files (
    void )
```

This function free the temporary memory of enterprise certificate data After add new enterprise network profile, the certificate data has been parsed by mbedtls into another data, which can be freed.

5.10.3.240 wlan_check_11n_capa()

```
uint8_t wlan_check_11n_capa (
    unsigned int channel )
```

Check if 11n(2G or 5G) is supported by hardware or not.

Parameters

in	<i>channel</i>	Channel number.
----	----------------	-----------------

Returns

true if 11n is supported or false if not.

5.10.3.241 wlan_check_11ac_capa()

```
uint8_t wlan_check_11ac_capa (
    unsigned int channel )
```

Check if 11ac(2G or 5G) is supported by hardware or not.

Parameters

in	<i>channel</i>	Channel number.
----	----------------	-----------------

Returns

true if 11ac is supported or false if not.

5.10.3.242 wlan_check_11ax_capa()

```
uint8_t wlan_check_11ax_capa (
    unsigned int channel )
```

Check if 11ax(2G or 5G) is supported by hardware or not.

Parameters

in	<i>channel</i>	Channel number.
----	----------------	-----------------

Returns

true if 11ax is supported or false if not.

5.10.3.243 wlan_get_signal_info()

```
int wlan_get_signal_info (
    wlan_rssi_info_t * signal )
```

Get rssi information.

Confidential

Parameters

out	<i>signal</i>	rss information get report buffer
-----	---------------	-----------------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.244 wlan_set_rg_power_cfg()

```
int wlan_set_rg_power_cfg (
    t_u16 region_code )
```

set region power table

Parameters

in	<i>region_code</i>	region code
----	--------------------	-------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.245 wlan_get_turbo_mode()

```
int wlan_get_turbo_mode (
    t_u8 * mode )
```

Get Turbo mode.

Parameters

out	<i>mode</i>	turbo mode 0: disable turbo mode 1: turbo mode 1 2: turbo mode 2 3: turbo mode 3
-----	-------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.246 wlan_get_uap_turbo_mode()

```
int wlan_get_uap_turbo_mode (
    t_u8 * mode )
```

Get UAP Turbo mode.

Confidential

Parameters

out	mode	turbo mode 0: disable turbo mode 1: turbo mode 1 2: turbo mode 2 3: turbo mode 3
-----	------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.247 wlan_set_turbo_mode()

```
int wlan_set_turbo_mode (  
    t_u8 mode )
```

Set Turbo mode.

Parameters

in	mode	turbo mode 0: disable turbo mode 1: turbo mode 1 2: turbo mode 2 3: turbo mode 3
----	------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.248 wlan_set_uap_turbo_mode()

```
int wlan_set_uap_turbo_mode (  
    t_u8 mode )
```

Set UAP Turbo mode.

Parameters

in	mode	turbo mode 0: disable turbo mode 1: turbo mode 1 2: turbo mode 2 3: turbo mode 3
----	------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.249 wlan_set_ps_cfg()

```
void wlan_set_ps_cfg (  
    t_u16 multiple_dtims,
```

```

t_u16 bcn_miss_timeout,
t_u16 local_listen_interval,
t_u16 adhoc_wake_period,
t_u16 mode,
t_u16 delay_to_ps )

```

set ps configuration. Currently only used to modify multiple dtim.

Parameters

in	<i>multiple_dtim</i>	num dtims, range [1,20]
in	<i>bcn_miss_timeout</i>	beacon miss interval
in	<i>local_listen_interval</i>	local listen interval
in	<i>adhoc_wake_period</i>	adhoc awake period
in	<i>mode</i>	mode - (0x01 - firmware to automatically choose PS_POLL or NULL mode, 0x02 - PS_POLL, 0x03 - NULL mode)
in	<i>delay_to_ps</i>	Delay to PS in milliseconds

5.10.3.250 wlan_save_cloud_keep_alive_params()

```

int wlan_save_cloud_keep_alive_params (
    wlan_cloud_keep_alive_t * cloud_keep_alive,
    t_u16 src_port,
    t_u16 dst_port,
    t_u32 seq_number,
    t_u32 ack_number,
    t_u8 enable )

```

Save start cloud keep alive parameters

Parameters

in	<i>cloud_keep_alive</i>	cloud keep alive information
in	<i>src_port</i>	Source port
in	<i>dst_port</i>	Destination port
in	<i>seq_number</i>	Sequence number
in	<i>ack_number</i>	Acknowledgement number
in	<i>enable</i>	Enable

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.251 wlan_cloud_keep_alive_enabled()

```

int wlan_cloud_keep_alive_enabled (
    t_u32 dst_ip,
    t_u16 dst_port )

```

Get cloud keep alive status for given destination ip and port

Parameters

in	<i>dst_ip</i>	Destination ip address
in	<i>dst_port</i>	Destination port

Returns

1 if enabled otherwise 0.

5.10.3.252 wlan_start_cloud_keep_alive()

```
int wlan_start_cloud_keep_alive (
    void )
```

Start cloud keep alive

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.253 wlan_stop_cloud_keep_alive()

```
int wlan_stop_cloud_keep_alive (
    wlan_cloud_keep_alive_t * cloud_keep_alive )
```

Stop cloud keep alive

Parameters

in	<i>cloud_keep_alive</i>	cloud keep alive information
----	-------------------------	------------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.254 wlan_set_country_code()

```
int wlan_set_country_code (
    const char * alpha2 )
```

Set country code

Note

This API should be called after WLAN is initialized but before starting uAP interface.

Parameters

in	<i>alpha2</i>	country code in 3 octets string, 2 octets country code and 1 octet environment 2 octets country code supported: WW : World Wide Safe US : US FCC CA : IC Canada SG : Singapore EU : ETSI AU : Australia KR : Republic Of Korea FR : France JP : Japan CN : China
----	---------------	--

For the third octet, STA is always 0. For uAP environment: All environments of the current frequency band and country (default) alpha2[2]=0x20 Outdoor environment only alpha2[2]=0x4f Indoor environment only alpha2[2]=0x49 Noncountry entity (country_code=XX) alpha[2]=0x58 IEEE 802.11 standard Annex E table indication: 0x01 .. 0x1f Annex E, Table E-4 (Global operating classes) alpha[2]=0x04

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.255 wlan_set_country_ie_ignore()

```
int wlan_set_country_ie_ignore (
    uint8_t * ignore )
```

Set ignore region code

Parameters

in	<i>ignore</i>	0: Don't ignore 1: ignore
----	---------------	---------------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.256 wlan_set_region_code()

```
int wlan_set_region_code (
    unsigned int region_code )
```

Set region code

Parameters

in	<i>region_code</i>	
----	--------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.257 wlan_get_region_code()

```
int wlan_get_region_code (
    unsigned int * region_code )
```

Get region code

Parameters

out	<i>region_code</i>	pointer
-----	--------------------	---------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.258 wlan_set_11d_state()

```
int wlan_set_11d_state (
    int bss_type,
    int state )
```

Set STA/uAP 80211d feature enable/disable

Parameters

in	<i>bss_type</i>	0: STA, 1: uAP
in	<i>state</i>	0: disable, 1: enable

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.259 wlan_dpp_configurator_add()

```
int wlan_dpp_configurator_add (
    int is_ap,
    const char * cmd )
```

Add a DPP Configurator

If this device is DPP Configurator, add it to get configurator ID.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"curve=P-256"

Returns

configurator ID if successful otherwise failure.

5.10.3.260 wlan_dpp_configurator_params()

```
void wlan_dpp_configurator_params (
    int is_ap,
    const char * cmd )
```

Set DPP Configurator parameter

set DPP configurator params. for example:" conf=<sta-dpp/ap-dpp> ssid=<hex ssid> configurator=conf_id"
#space character exists between " & conf word.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	" conf=<sta-dpp/ap-dpp/sta-psk> ssid=<hex ssid> configurator=conf_id..."

Returns

void

5.10.3.261 wlan_dpp_mud_url()

```
void wlan_dpp_mud_url (
    int is_ap,
    const char * cmd )
```

MUD URL for Enrollee's DPP Configuration Request (optional)

Wi-Fi_CERTIFIED_Easy_Connect_Test_Plan_v3.0.pdf 5.1.23 STAUT sends the MUD URL

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"https://example.com/mud"

Returns

void

5.10.3.262 wlan_dpp_bootstrap_gen()

```
int wlan_dpp_bootstrap_gen (
    int is_ap,
    const char * cmd )
```

Generate QR code

This function generates QR code and return bootstrap-id

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"type=qr code mac=<mac-address-of-device> chan=<operating-class/channel>..."

Returns

bootstrap-id if successful otherwise failure.

5.10.3.263 wlan_dpp_bootstrap_get_uri()

```
const char* wlan_dpp_bootstrap_get_uri (
    int is_ap,
    unsigned int id )
```

Get QR code by bootstrap-id

This function get QR code string by bootstrap-id

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>id</i>	bootstrap-id

Returns

QR code string if successful otherwise NULL.

5.10.3.264 wlan_dpp_qr_code()

```
int wlan_dpp_qr_code (
    int is_ap,
    char * uri )
```

Enter the QR code in the DPP device.

This function set the QR code and return qr-code-id.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>uri</i>	QR code provided by other device.

Returns

qr-code-id if successful otherwise failure.

5.10.3.265 wlan_dpp_auth_init()

```
int wlan_dpp_auth_init (
    int is_ap,
    const char * cmd )
```

Send provisioning Auth request to responder.

This function send Auth request to responder by qr-code-id.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"peer=<qr-code-id> conf=<sta-dpp/ap-dpp/sta-psk>"

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.266 wlan_dpp_listen()

```
int wlan_dpp_listen (
    int is_ap,
    const char * cmd )
```

Make device listen to DPP request.

Responder generates QR code and listening on its operating channel to wait Auth request.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"<frequency>"

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.267 wlan_dpp_stop_listen()

```
int wlan_dpp_stop_listen (  
    int is_ap )
```

DPP stop listen

Stop dpp listen and clear listen frequency

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
----	--------------	--------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.268 wlan_dpp_pkex_add()

```
int wlan_dpp_pkex_add (  
    int is_ap,  
    const char * cmd )
```

Set bootstrapping through PKEX(Public Key Exchange)

Support in-band bootstrapping through PKEX

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"own=<bootstrap_id> identifier=<string> code=<string>"

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.269 wlan_dpp_chirp()

```
int wlan_dpp_chirp (
    int is_ap,
    const char * cmd )
```

sends DPP presence announcement.

Send DPP presence announcement from responder. After the Initiator enters the QRcode URI provided by the Responder, the Responder sends the presence announcement to trigger Auth Request from Initiator.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	"own=<bootstrap id> listen=<freq> ..."

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.270 wlan_dpp_reconfig()

```
int wlan_dpp_reconfig (
    const char * cmd )
```

DPP reconfig

DPP reconfig and make a new DPP connection.

Parameters

in	<i>cmd</i>	"<network id> ..."
----	------------	--------------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.271 wlan_dpp_configurator_sign()

```
int wlan_dpp_configurator_sign (
    int is_ap,
    const char * cmd )
```

Configurator configures itself as an Enrollee AP/STA

Wi-Fi_CERTIFIED_Easy_Connect_Test_Plan_v3.0.pdf 5.3.8 & 5.3.9 Configurator configures itself as an Enrollee AP/STA

for example:" conf=<sta-dpp/ap-dpp> ssid=<hex ssid> configurator=conf_id" #space character exists between " & conf word.

Parameters

in	<i>is_ap</i>	0 is sta, 1 is uap
in	<i>cmd</i>	" conf=<sta-dpp/ap-dpp/sta-psk> ssid=<hex ssid> configurator=conf_id..."

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.272 wlan_host_set_sta_mac_filter()

```
int wlan_host_set_sta_mac_filter (
    int filter_mode,
    int mac_count,
    unsigned char * mac_addr )
```

5.10.3.273 wlan_set_indrst_cfg()

```
int wlan_set_indrst_cfg (
    const wifi_indrst_cfg_t * indrst_cfg )
```

Set GPIO independent reset configuration

Parameters

in	<i>indrst_cfg</i>	GPIO independent reset config to be sent to Firmware
----	-------------------	--

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.274 wlan_get_indrst_cfg()

```
int wlan_get_indrst_cfg (
    wifi_indrst_cfg_t * indrst_cfg )
```

5.10.3.275 wlan_independent_reset()

```
int wlan_independent_reset ( )
```

Test Independent Firmware reset

This function will either send cmd that will cause timeout in firmware or send GPIO pulse that will cause out of band reset in firmware as per configuration int earlier [wlan_set_indrst_cfg](#) API.

Returns

WM_SUCCESS if successful otherwise failure.

5.10.3.276 wlan_set_network_ip_byname()

```
int wlan_set_network_ip_byname (
    char * name,
    struct wlan_ip_config * ip )
```

5.10.3.277 wlan_string_dup()

```
char* wlan_string_dup (
    const char * s )
```

allocate a copy of a string

5.10.3.278 wlan_get_board_type()

```
uint32_t wlan_get_board_type ( )
```

Get board type.

Returns

board type.

5.10.3.279 wlan_uap_disconnect_sta()

```
int wlan_uap_disconnect_sta (
    uint8_t * sta_addr )
```

Disconnect to sta which is connected with internal uap.

Parameters

in	<i>sta_addr</i>	sta mac address
----	-----------------	-----------------

Returns

WM_SUCCESS if successful otherwise failure.

5.10.4 Macro Documentation

5.10.4.1 WLAN_DRV_VERSION

```
#define WLAN_DRV_VERSION "v1.3.r48.p16"
```

5.10.4.2 ARG_UNUSED

```
#define ARG_UNUSED(  
    x ) (void) (x)
```

5.10.4.3 CONFIG_WLAN_KNOWN_NETWORKS

```
#define CONFIG_WLAN_KNOWN_NETWORKS 5U
```

5.10.4.4 wlcm_e

```
#define wlcm_e(  
    ... ) wmlog_e("wlcm", ##__VA_ARGS__)
```

5.10.4.5 wlcm_w

```
#define wlcm_w(  
    ... ) wmlog_w("wlcm", ##__VA_ARGS__)
```

5.10.4.6 wlc_m_d

```
#define wlc_m_d(  
    ... ) wmlog("wlc_m", ##__VA_ARGS__)
```

5.10.4.7 ACTION_GET

```
#define ACTION_GET (0U)
```

Action GET

5.10.4.8 ACTION_SET

```
#define ACTION_SET (1)
```

Action SET

5.10.4.9 IEEEtypes_SSID_SIZE

```
#define IEEEtypes_SSID_SIZE 32U
```

Maximum SSID length

5.10.4.10 IEEEtypes_ADDRESS_SIZE

```
#define IEEEtypes_ADDRESS_SIZE 6
```

MAC Address length

5.10.4.11 WLAN_REASON_CODE_PREV_AUTH_NOT_VALID

```
#define WLAN_REASON_CODE_PREV_AUTH_NOT_VALID 2U
```

5.10.4.12 WLAN_RESCAN_LIMIT

```
#define WLAN_RESCAN_LIMIT 30U
```

The number of times that the WLAN Connection Manager will look for a network before giving up.

5.10.4.13 WLAN_11D_SCAN_LIMIT

```
#define WLAN_11D_SCAN_LIMIT 3U
```

5.10.4.14 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.

5.10.4.15 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](#)

5.10.4.16 WLAN_NETWORK_NAME_MAX_LENGTH

```
#define WLAN_NETWORK_NAME_MAX_LENGTH 32U
```

The space reserved for storing network names, [wlan_network](#)

5.10.4.17 WLAN_PSK_MIN_LENGTH

```
#define WLAN_PSK_MIN_LENGTH 8U
```

The space reserved for storing PSK (password) phrases.

5.10.4.18 WLAN_PSK_MAX_LENGTH

```
#define WLAN_PSK_MAX_LENGTH 65U
```

Max WPA2 passphrase can be upto 63 ASCII chars or 64 hexadecimal digits

5.10.4.19 WLAN_PASSWORD_MIN_LENGTH

```
#define WLAN_PASSWORD_MIN_LENGTH 8U
```

Min WPA3 password can be upto 8 ASCII chars

5.10.4.20 WLAN_PASSWORD_MAX_LENGTH

```
#define WLAN_PASSWORD_MAX_LENGTH 255U
```

Max WPA3 password can be upto 255 ASCII chars

5.10.4.21 IDENTITY_MAX_LENGTH

```
#define IDENTITY_MAX_LENGTH 64U
```

Max WPA2 Enterprise identity can be upto 256 characters

5.10.4.22 PASSWORD_MAX_LENGTH

```
#define PASSWORD_MAX_LENGTH 128U
```

Max WPA2 Enterprise password can be upto 256 unicode characters

5.10.4.23 MAX_USERS

```
#define MAX_USERS 8U
```

Max identities for EAP server users

5.10.4.24 PAC_OPAQUE_ENCR_KEY_MAX_LENGTH

```
#define PAC_OPAQUE_ENCR_KEY_MAX_LENGTH 33U
```

Encryption key for EAP-FAST PAC-Opaque values. This key must be a secret, random value. It is configured as a 16-octet value in hex format.

5.10.4.25 A_ID_MAX_LENGTH

```
#define A_ID_MAX_LENGTH 33U
```

A-ID indicates the identity of the authority that issues PACs. The A-ID should be unique across all issuing servers. A-ID to be 16 octets in length

5.10.4.26 HASH_MAX_LENGTH

```
#define HASH_MAX_LENGTH 40U
```

MAX CA Cert hash len

5.10.4.27 DOMAIN_MATCH_MAX_LENGTH

```
#define DOMAIN_MATCH_MAX_LENGTH 64U
```

MAX domain len

5.10.4.28 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

5.10.4.29 WLAN_PMK_LENGTH

```
#define WLAN_PMK_LENGTH 32
```

Length of a pairwise master key (PMK). It's always 256 bits (32 Bytes)

5.10.4.30 WLAN_MAX_STA_FILTER_NUM

```
#define WLAN_MAX_STA_FILTER_NUM 16
```

5.10.4.31 WLAN_MAC_ADDR_LENGTH

```
#define WLAN_MAC_ADDR_LENGTH 6
```

5.10.4.32 WLAN_ERROR_NONE

```
#define WLAN_ERROR_NONE 0
```

The operation was successful.

5.10.4.33 WLAN_ERROR_PARAM

```
#define WLAN_ERROR_PARAM 1
```

The operation failed due to an error with one or more parameters.

5.10.4.34 WLAN_ERROR_NOMEM

```
#define WLAN_ERROR_NOMEM 2
```

The operation could not be performed because there is not enough memory.

5.10.4.35 WLAN_ERROR_STATE

```
#define WLAN_ERROR_STATE 3
```

The operation could not be performed in the current system state.

5.10.4.36 WLAN_ERROR_ACTION

```
#define WLAN_ERROR_ACTION 4
```

The operation failed due to an internal error.

5.10.4.37 WLAN_ERROR_PS_ACTION

```
#define WLAN_ERROR_PS_ACTION 5
```

The operation to change power state could not be performed

5.10.4.38 WLAN_ERROR_NOT_SUPPORTED

```
#define WLAN_ERROR_NOT_SUPPORTED 6
```

The requested feature is not supported

5.10.4.39 HOST_WAKEUP_GPIO_PIN

```
#define HOST_WAKEUP_GPIO_PIN 17
```

5.10.4.40 CARD_WAKEUP_GPIO_PIN

```
#define CARD_WAKEUP_GPIO_PIN 16
```

5.10.4.41 WLAN_MGMT_DIASSOC

```
#define WLAN_MGMT_DIASSOC MBIT(10)
```

5.10.4.42 WLAN_MGMT_AUTH

```
#define WLAN_MGMT_AUTH MBIT(11)
```

5.10.4.43 WLAN_MGMT_DEAUTH

```
#define WLAN_MGMT_DEAUTH MBIT(12)
```

5.10.4.44 WLAN_MGMT_ACTION

```
#define WLAN_MGMT_ACTION MBIT(13)
```

BITMAP for Action frame

5.10.4.45 WLAN_KEY_MGMT_IEEE8021X

```
#define WLAN_KEY_MGMT_IEEE8021X MBIT(0)
```

5.10.4.46 WLAN_KEY_MGMT_PSK

```
#define WLAN_KEY_MGMT_PSK MBIT(1)
```

5.10.4.47 WLAN_KEY_MGMT_NONE

```
#define WLAN_KEY_MGMT_NONE MBIT(2)
```

5.10.4.48 WLAN_KEY_MGMT_IEEE8021X_NO_WPA

```
#define WLAN_KEY_MGMT_IEEE8021X_NO_WPA MBIT(3)
```

5.10.4.49 WLAN_KEY_MGMT_WPA_NONE

```
#define WLAN_KEY_MGMT_WPA_NONE MBIT(4)
```

5.10.4.50 WLAN_KEY_MGMT_FT_IEEE8021X

```
#define WLAN_KEY_MGMT_FT_IEEE8021X MBIT(5)
```

5.10.4.51 WLAN_KEY_MGMT_FT_PSK

```
#define WLAN_KEY_MGMT_FT_PSK MBIT(6)
```

5.10.4.52 WLAN_KEY_MGMT_IEEE8021X_SHA256

```
#define WLAN_KEY_MGMT_IEEE8021X_SHA256 MBIT(7)
```

5.10.4.53 WLAN_KEY_MGMT_PSK_SHA256

```
#define WLAN_KEY_MGMT_PSK_SHA256 MBIT(8)
```

5.10.4.54 WLAN_KEY_MGMT_WPS

```
#define WLAN_KEY_MGMT_WPS MBIT(9)
```

5.10.4.55 WLAN_KEY_MGMT_SAE

```
#define WLAN_KEY_MGMT_SAE MBIT(10)
```

5.10.4.56 WLAN_KEY_MGMT_FT_SAE

```
#define WLAN_KEY_MGMT_FT_SAE MBIT(11)
```

5.10.4.57 WLAN_KEY_MGMT_WAPI_PSK

```
#define WLAN_KEY_MGMT_WAPI_PSK MBIT(12)
```

5.10.4.58 WLAN_KEY_MGMT_WAPI_CERT

```
#define WLAN_KEY_MGMT_WAPI_CERT MBIT(13)
```


5.10.4.59 WLAN_KEY_MGMT_CCKM

```
#define WLAN_KEY_MGMT_CCKM MBIT(14)
```

5.10.4.60 WLAN_KEY_MGMT_OSEN

```
#define WLAN_KEY_MGMT_OSEN MBIT(15)
```

5.10.4.61 WLAN_KEY_MGMT_IEEE8021X_SUITE_B

```
#define WLAN_KEY_MGMT_IEEE8021X_SUITE_B MBIT(16)
```

5.10.4.62 WLAN_KEY_MGMT_IEEE8021X_SUITE_B_192

```
#define WLAN_KEY_MGMT_IEEE8021X_SUITE_B_192 MBIT(17)
```

5.10.4.63 WLAN_KEY_MGMT_FILS_SHA256

```
#define WLAN_KEY_MGMT_FILS_SHA256 MBIT(18)
```

5.10.4.64 WLAN_KEY_MGMT_FILS_SHA384

```
#define WLAN_KEY_MGMT_FILS_SHA384 MBIT(19)
```

5.10.4.65 WLAN_KEY_MGMT_FT_FILS_SHA256

```
#define WLAN_KEY_MGMT_FT_FILS_SHA256 MBIT(20)
```

5.10.4.66 WLAN_KEY_MGMT_FT_FILS_SHA384

```
#define WLAN_KEY_MGMT_FT_FILS_SHA384 MBIT(21)
```

5.10.4.67 WLAN_KEY_MGMT_OWE

```
#define WLAN_KEY_MGMT_OWE MBIT(22)
```

5.10.4.68 WLAN_KEY_MGMT_DPP

```
#define WLAN_KEY_MGMT_DPP MBIT(23)
```

5.10.4.69 WLAN_KEY_MGMT_FT_IEEE8021X_SHA384

```
#define WLAN_KEY_MGMT_FT_IEEE8021X_SHA384 MBIT(24)
```

5.10.4.70 WLAN_KEY_MGMT_PASN

```
#define WLAN_KEY_MGMT_PASN MBIT(25)
```

5.10.4.71 WLAN_KEY_MGMT_SAE_EXT_KEY

```
#define WLAN_KEY_MGMT_SAE_EXT_KEY MBIT(26)
```

5.10.4.72 WLAN_KEY_MGMT_FT

```
#define WLAN_KEY_MGMT_FT
```

Value:

```
(WLAN_KEY_MGMT_FT_PSK | WLAN_KEY_MGMT_FT_IEEE8021X |  
 WLAN_KEY_MGMT_FT_IEEE8021X_SHA384 |  
 WLAN_KEY_MGMT_FT_SAE | \  
 WLAN_KEY_MGMT_FT_FILS_SHA256 | WLAN_KEY_MGMT_FT_FILS_SHA384)
```

5.10.4.73 WLAN_CIPHER_NONE

```
#define WLAN_CIPHER_NONE MBIT(0)
```

5.10.4.74 WLAN_CIPHER_WEP40

```
#define WLAN_CIPHER_WEP40 MBIT(1)
```

5.10.4.75 WLAN_CIPHER_WEP104

```
#define WLAN_CIPHER_WEP104 MBIT(2)
```

5.10.4.76 WLAN_CIPHER_TKIP

```
#define WLAN_CIPHER_TKIP MBIT(3)
```

5.10.4.77 WLAN_CIPHER_CCMP

```
#define WLAN_CIPHER_CCMP MBIT(4)
```

5.10.4.78 WLAN_CIPHER_AES_128_CMAC

```
#define WLAN_CIPHER_AES_128_CMAC MBIT(5)
```

5.10.4.79 WLAN_CIPHER_GCMP

```
#define WLAN_CIPHER_GCMP MBIT(6)
```

5.10.4.80 WLAN_CIPHER_SMS4

```
#define WLAN_CIPHER_SMS4 MBIT(7)
```

5.10.4.81 WLAN_CIPHER_GCMP_256

```
#define WLAN_CIPHER_GCMP_256 MBIT(8)
```

5.10.4.82 WLAN_CIPHER_CCMP_256

```
#define WLAN_CIPHER_CCMP_256 MBIT (9)
```

5.10.4.83 WLAN_CIPHER_BIP_GMAC_128

```
#define WLAN_CIPHER_BIP_GMAC_128 MBIT (11)
```

5.10.4.84 WLAN_CIPHER_BIP_GMAC_256

```
#define WLAN_CIPHER_BIP_GMAC_256 MBIT (12)
```

5.10.4.85 WLAN_CIPHER_BIP_CMAC_256

```
#define WLAN_CIPHER_BIP_CMAC_256 MBIT (13)
```

5.10.4.86 WLAN_CIPHER_GTK_NOT_USED

```
#define WLAN_CIPHER_GTK_NOT_USED MBIT (14)
```

5.10.4.87 NUM_CHAN_BAND_ENUMS

```
#define NUM_CHAN_BAND_ENUMS 3
```

5.10.4.88 DFS_REC_HDR_LEN

```
#define DFS_REC_HDR_LEN (8)
```

5.10.4.89 DFS_REC_HDR_NUM

```
#define DFS_REC_HDR_NUM (10)
```

5.10.4.90 BIN_COUNTER_LEN

```
#define BIN_COUNTER_LEN (7)
```

5.10.4.91 MAX_CHANNEL_LIST

```
#define MAX_CHANNEL_LIST 6
```

5.10.4.92 TX_AMPDU_RTS_CTS

```
#define TX_AMPDU_RTS_CTS 0
```

5.10.4.93 TX_AMPDU_CTS_2_SELF

```
#define TX_AMPDU_CTS_2_SELF 1
```

5.10.4.94 TX_AMPDU_DISABLE_PROTECTION

```
#define TX_AMPDU_DISABLE_PROTECTION 2
```

5.10.4.95 TX_AMPDU_DYNAMIC_RTS_CTS

```
#define TX_AMPDU_DYNAMIC_RTS_CTS 3
```

5.10.4.96 EU_CRYPT_DATA_MAX_LENGTH

```
#define EU_CRYPT_DATA_MAX_LENGTH 1300U
```

5.10.4.97 EU_CRYPT_KEY_MAX_LENGTH

```
#define EU_CRYPT_KEY_MAX_LENGTH 32U
```

5.10.4.98 EU_CRYPT0_KEYIV_MAX_LENGTH

```
#define EU_CRYPT0_KEYIV_MAX_LENGTH 32U
```

5.10.4.99 EU_CRYPT0_NONCE_MAX_LENGTH

```
#define EU_CRYPT0_NONCE_MAX_LENGTH 14U
```

5.10.4.100 EU_CRYPT0_AAD_MAX_LENGTH

```
#define EU_CRYPT0_AAD_MAX_LENGTH 32U
```

5.10.4.101 FILE_TYPE_NONE

```
#define FILE_TYPE_NONE 0
```

5.10.4.102 FILE_TYPE_ENTP_CA_CERT

```
#define FILE_TYPE_ENTP_CA_CERT 1
```

5.10.4.103 FILE_TYPE_ENTP_CLIENT_CERT

```
#define FILE_TYPE_ENTP_CLIENT_CERT 2
```

5.10.4.104 FILE_TYPE_ENTP_CLIENT_KEY

```
#define FILE_TYPE_ENTP_CLIENT_KEY 3
```

5.10.4.105 FILE_TYPE_ENTP_CA_CERT2

```
#define FILE_TYPE_ENTP_CA_CERT2 4
```

5.10.4.106 FILE_TYPE_ENTP_CLIENT_CERT2

```
#define FILE_TYPE_ENTP_CLIENT_CERT2 5
```

5.10.4.107 FILE_TYPE_ENTP_CLIENT_KEY2

```
#define FILE_TYPE_ENTP_CLIENT_KEY2 6
```

5.10.4.108 FILE_TYPE_ENTP_SERVER_CERT

```
#define FILE_TYPE_ENTP_SERVER_CERT 8
```

5.10.4.109 FILE_TYPE_ENTP_SERVER_KEY

```
#define FILE_TYPE_ENTP_SERVER_KEY 9
```

5.10.4.110 FILE_TYPE_ENTP_DH_PARAMS

```
#define FILE_TYPE_ENTP_DH_PARAMS 10
```

5.10.5 Typedef Documentation

5.10.5.1 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](#)

5.10.5.2 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](#)

5.10.5.3 wlan_cal_data_t

```
typedef wifi_cal_data_t wlan_cal_data_t
```

Configuration for Wireless Calibration data from [wifi_cal_data_t](#)

5.10.5.4 wlan_auto_reconnect_config_t

```
typedef wifi_auto_reconnect_config_t wlan_auto_reconnect_config_t
```

Configuration for Auto reconnect configuration from [wifi_auto_reconnect_config_t](#)

5.10.5.5 wlanflt_cfg_t

```
typedef wififlt_cfg_t wlanflt_cfg_t
```

Configuration for Memory Efficient Filters in Wi-Fi firmware from [wififlt_cfg_t](#)

5.10.5.6 wlanwowlan_ptn_cfg_t

```
typedef wifiwowlan_ptn_cfg_t wlanwowlan_ptn_cfg_t
```

Configuration for wowlan pattern parameters from [wifiwowlan_ptn_cfg_t](#)

5.10.5.7 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](#)

5.10.5.8 wlan_cloud_keep_alive_t

```
typedef wifi_cloud_keep_alive_t wlan_cloud_keep_alive_t
```

Configuration for Cloud Keep alive parameters from [wifi_cloud_keep_alive_t](#)

5.10.5.9 wlan_ds_rate

```
typedef wifi_ds_rate wlan_ds_rate
```

Configuration for TX Rate and Get data rate from [wifi_ds_rate](#)

5.10.5.10 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](#)

5.10.5.11 wlan_bandcfg_t

```
typedef wifi_bandcfg_t wlan_bandcfg_t
```

Configuration for Band from [wifi_bandcfg_t](#)

5.10.5.12 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](#)

5.10.5.13 wlan_chanlist_t

```
typedef wifi_chanlist_t wlan_chanlist_t
```

Configuration for Channel list from [wifi_chanlist_t](#)

5.10.5.14 wlan_txpwrlimit_t

```
typedef wifi_txpwrlimit_t wlan_txpwrlimit_t
```

Configuration for TX Pwr Limit from [wifi_txpwrlimit_t](#)

5.10.5.15 wlan_ext_coex_stats_t

```
typedef wifi_ext_coex_stats_t wlan_ext_coex_stats_t
```

Statistic of External Coex from [wifi_ext_coex_config_t](#)

5.10.5.16 wlan_ext_coex_config_t

```
typedef wifi_ext_coex_config_t wlan_ext_coex_config_t
```

Configuration for External Coex from [wifi_ext_coex_config_t](#)

5.10.5.17 wlan_rutxpwrlimit_t

```
typedef wifi_rutxpwrlimit_t wlan_rutxpwrlimit_t
```

Configuration for RU TX Pwr Limit from [wifi_rutxpwrlimit_t](#)

5.10.5.18 wlan_11ax_config_t

```
typedef wifi_11ax_config_t wlan_11ax_config_t
```

Configuration for 11AX capabilities [wifi_11ax_config_t](#)

5.10.5.19 wlan_twt_setup_config_t

```
typedef wifi_twt_setup_config_t wlan_twt_setup_config_t
```

Configuration for TWT Setup [wifi_twt_setup_config_t](#)

5.10.5.20 wlan_twt_teardown_config_t

```
typedef wifi_twt_teardown_config_t wlan_twt_teardown_config_t
```

Configuration for TWT Teardown [wifi_twt_teardown_config_t](#)

5.10.5.21 wlan_btwt_config_t

```
typedef wifi_btwt_config_t wlan_btwt_config_t
```

Configuration for Broadcast TWT Setup [wifi_btwt_config_t](#)

5.10.5.22 wlan_twt_report_t

```
typedef wifi_twt_report_t wlan_twt_report_t
```

Configuration for TWT Report [wifi_twt_report_t](#)

5.10.5.23 wlan_twt_information_t

```
typedef wifi_twt_information_t wlan_twt_information_t
```

Configuration for TWT Information [wifi_twt_information_t](#)

5.10.5.24 wlan_clock_sync_gpio_tsf_t

```
typedef wifi_clock_sync_gpio_tsf_t wlan_clock_sync_gpio_tsf_t
```

Configuration for Clock Sync GPIO TSF latch [wifi_clock_sync_gpio_tsf_t](#)

5.10.5.25 wlan_tsf_info_t

```
typedef wifi_tsf_info_t wlan_tsf_info_t
```

Configuration for TSF info [wifi_tsf_info_t](#)

5.10.5.26 wlan_mgmt_frame_t

```
typedef wifi_mgmt_frame_t wlan_mgmt_frame_t
```

5.10.5.27 wlan_csi_config_params_t

```
typedef wifi_csi_config_params_t wlan_csi_config_params_t
```

Configuration for Csi Config Params from [wifi_csi_config_params_t](#)

5.10.5.28 wlan_indrst_cfg_t

```
typedef wifi_indrst_cfg_t wlan_indrst_cfg_t
```

Configuration for GPIO independent reset [wifi_indrst_cfg_t](#)

5.10.5.29 wlan_txrate_setting

```
typedef txrate_setting wlan_txrate_setting
```

Configuration for TX Rate Setting from [txrate_setting](#)

5.10.5.30 wlan_rssi_info_t

```
typedef wifi_rssi_info_t wlan_rssi_info_t
```

Configuration for RSSI information [wifi_rssi_info_t](#)

5.10.5.31 wlan_uap_client_disassoc_t

```
typedef wifi_uap_client_disassoc_t wlan_uap_client_disassoc_t
```

5.10.6 Enumeration Type Documentation

5.10.6.1 IEEEtypes_Bss_t

```
enum IEEEtypes_Bss_t
```

Enumerator

BSS_INFRASTRUCTURE	
BSS_INDEPENDENT	
BSS_ANY	

5.10.6.2 wm_wlan_errno

enum `wm_wlan_errno`

Enum for wlan errors

Enumerator

WM_E_WLAN_ERRNO_BASE	
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

5.10.6.3 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 WLAN_CONNECTED state.
WLAN_REASON_AUTH_SUCCESS	The WLAN Connection Manager has successfully authenticated to a network and is now in the WLAN_ASSOCIATED 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 WLAN_DISCONNECTED 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 WLAN_DISCONNECTED state.
WLAN_REASON_BGSCAN_NETWORK_NOT_FOUND	The WLAN Connection Manager could not find the network in bg scan during roam attempt that it was connecting to and it is now in the WLAN_CONNECTED state with previous AP.
WLAN_REASON_NETWORK_AUTH_FAILED	The WLAN Connection Manager failed to authenticate with the network and is now in the WLAN_DISCONNECTED 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 WLAN_DISCONNECTED state.

Enumerator

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 WLAN_DISCONNECTED 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 WLAN_DISCONNECTED 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.
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 wlan_stop() and wlan_start() respectively.
WLAN_REASON_FW_HANG	The WLAN Connection Manager has entered in hang mode.
WLAN_REASON_FW_RESET	The WLAN Connection Manager has reset fw successfully.
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_CONN	A wireless client has authenticated and connected to 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
WLAN_REASON_RSSI_LOW	The WLAN Connection Manager has received subscribed RSSI low event on station interface as per configured threshold and frequency. If CONFIG_11K, CONFIG_11V, CONFIG_11R or CONFIG_ROAMING enabled then RSSI low event is processed internally.

5.10.6.4 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

5.10.6.5 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.
WLAN_ASSOCIATED	The WLAN Connection Manager is not connected but associated.
WLAN_AUTHENTICATED	The WLAN Connection Manager is not connected but authenticated.
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.

5.10.6.6 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

Enumerator

WLAN_IEEE_DEEP_SLEEP	IEEE and Deep sleep power save mode
WLAN_WNM	WNM power save mode
WLAN_WNM_DEEP_SLEEP	WNM and Deep sleep power save mode

5.10.6.7 wlan_ps_state

```
enum wlan_ps_state
```

Enumerator

PS_STATE_AWAKE	
PS_STATE_PRE_SLEEP	
PS_STATE_SLEEP_CFM	
PS_STATE_SLEEP	

5.10.6.8 ENH_PS_MODES

```
enum ENH_PS_MODES
```

Enumerator

GET_PS	
SLEEP_CONFIRM	
EXT_PS_PARAM	
DIS_WNM_PS	
EN_WNM_PS	
DIS_AUTO_PS	
EN_AUTO_PS	

5.10.6.9 Host_Sleep_Action

```
enum Host_Sleep_Action
```

Enumerator

HS_CONFIGURE	
HS_ACTIVATE	

5.10.6.10 wlan_csi_opt

enum `wlan_csi_opt`

Enumerator

CSI_FILTER_OPT_ADD	
CSI_FILTER_OPT_DELETE	
CSI_FILTER_OPT_CLEAR	
CSI_FILTER_OPT_DUMP	

5.10.6.11 wlan_monitor_opt

enum `wlan_monitor_opt`

Enumerator

MONITOR_FILTER_OPT_ADD_MAC	
MONITOR_FILTER_OPT_DELETE_MAC	
MONITOR_FILTER_OPT_CLEAR_MAC	
MONITOR_FILTER_OPT_DUMP	

5.10.6.12 ChanBand_e

enum `ChanBand_e`

Enumerator

Band_2_4_GHz	
Band_5_GHz	
Band_4_GHz	

5.10.6.13 ChanWidth_e

enum `ChanWidth_e`

Enumerator

ChanWidth_20_MHz	
ChanWidth_10_MHz	
ChanWidth_40_MHz	
ChanWidth_80_MHz	

5.10.6.14 Chan2Offset_e

enum `Chan2Offset_e`

Enumerator

SECONDARY_CHAN_NONE	
SECONDARY_CHAN_ABOVE	
SECONDARY_CHAN_BELOW	

5.10.6.15 ScanMode_e

enum `ScanMode_e`

Enumerator

MANUAL_MODE	
ACS_MODE	

5.10.6.16 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_WPA2_FT	The network uses WPA2 security with PSK FT.

Enumerator

WLAN_SECURITY_WPA3_SAE	The network uses WPA3 security with SAE.
WLAN_SECURITY_WPA3_FT_SAE	The network uses WPA3 security with SAE FT.
WLAN_SECURITY_WPA3_SAE_EXT_KEY	The network uses WPA3 security with SAE EXT KEY.
WLAN_SECURITY_WPA2_WPA3_SAE_MIXED	The network uses WPA2/WPA3 SAE mixed security with PSK.
WLAN_SECURITY_EAP_TLS	The network uses WPA2 Enterprise EAP-TLS security The identity field in wlan_network structure is used
WLAN_SECURITY_EAP_TLS_SHA256	The network uses WPA2 Enterprise EAP-TLS SHA256 security The identity field in wlan_network structure is used
WLAN_SECURITY_EAP_TLS_FT	The network uses WPA2 Enterprise EAP-TLS FT security The identity field in wlan_network structure is used
WLAN_SECURITY_EAP_TLS_FT_SHA384	The network uses WPA2 Enterprise EAP-TLS FT SHA384 security The identity field in wlan_network structure is used
WLAN_SECURITY_EAP_TTLS	The network uses WPA2 Enterprise EAP-TTLS security The identity field in wlan_network structure is used
WLAN_SECURITY_EAP_TTLS_MSCHAPV2	The network uses WPA2 Enterprise EAP-TTLS-MSCHAPV2 security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_PEAP_MSCHAPV2	The network uses WPA2 Enterprise EAP-PEAP-MSCHAPV2 security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_PEAP_TLS	The network uses WPA2 Enterprise EAP-PEAP-TLS security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_PEAP_GTC	The network uses WPA2 Enterprise EAP-PEAP-GTC security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_FAST_MSCHAPV2	The network uses WPA2 Enterprise EAP-FAST-MSCHAPV2 security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_FAST_GTC	The network uses WPA2 Enterprise EAP-FAST-GTC security The anonymous identity, identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_SIM	The network uses WPA2 Enterprise EAP-SIM security The identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_AKA	The network uses WPA2 Enterprise EAP-AKA security The identity and password fields in wlan_network structure are used
WLAN_SECURITY_EAP_AKA_PRIME	The network uses WPA2 Enterprise EAP-AKA-PRIME security The identity and password fields in wlan_network structure are used
WLAN_SECURITY_DPP	The network uses DPP security with NAK(Net Access Key)
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.

5.10.6.17 eap_tls_cipher_type

enum `eap_tls_cipher_type`

EAP TLS Cipher types

Enumerator

EAP_TLS_NONE	
EAP_TLS_ECC_P384	EAP TLS with ECDH & ECDSA with p384
EAP_TLS_RSA_3K	EAP TLS with ECDH & RSA with > 3K

5.10.6.18 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

5.10.6.19 wlan_mef_type

enum `wlan_mef_type`

Enumerator

MEF_TYPE_DELETE	
MEF_TYPE_PING	
MEF_TYPE_ARP	
MEF_TYPE_MULTICAST	
MEF_TYPE_IPV6_NS	
MEF_TYPE_END	

5.11 wlan_11d.h File Reference

WLAN module 11d API.

5.11.1 Function Documentation

5.11.1.1 wlan_enable_11d()

```
static int wlan_enable_11d (  
    int state ) [inline], [static]
```

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.

Parameters

in	state	1: enable, 0: disable
----	-------	-----------------------

Returns

-WM_FAIL if operation was failed.
WM_SUCCESS if operation was successful.

5.11.1.2 wlan_enable_uap_11d()

```
static int wlan_enable_uap_11d (  
    int state ) [inline], [static]
```

Enable 11D support in WLAN Driver for uap interface.

Note

This API should be called after WLAN is initialized but before starting uAP or making any connection attempts on station interface.

Parameters

in	state	1: enable, 0: disable
----	-------	-----------------------

Returns

-WM_FAIL if operation was failed.
WM_SUCCESS if operation was successful.

5.12 wlan_tests.h File Reference

WLAN Connection Manager Tests.

5.12.1 Function Documentation

5.12.1.1 test_wlan_cfg_process()

```
void test_wlan_cfg_process (
    uint32_t index,
    int argc,
    char ** argv )
```

5.12.1.2 print_txpwrlimit()

```
void print_txpwrlimit (
    wlan_txpwrlimit_t * txpwrlimit )
```

Print the TX PWR Limit table received from Wi-Fi firmware

Parameters

in	txpwrlimit	A wlan_txpwrlimit_t struct holding the the TX PWR Limit table received from Wi-Fi firmware.
----	------------	---

5.12.2 Enumeration Type Documentation

5.12.2.1 anonymous enum

anonymous enum

Enumerator

TEST_WLAN_11AX_CFG	
TEST_WLAN_BCAST_TWT	
TEST_WLAN_TWT_SETUP	
TEST_WLAN_TWT_TEARDOWN	

5.13 wm_net.h File Reference

Network Abstraction Layer.

5.13.1 Detailed Description

This provides the calls related to the network layer.

5.13.2 Function Documentation

5.13.2.1 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

5.13.2.2 net_stop_dhcp_timer()

```
void net_stop_dhcp_timer (
    void )
```

Deactivate the dhcp timer

5.13.2.3 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.

5.13.2.4 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.

5.13.2.5 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

5.13.2.6 net_wlan_set_mac_address()

```
void net_wlan_set_mac_address (
    unsigned char * stamac,
    unsigned char * uapmac )
```

set MAC hardware address to lwip network interface

Parameters

in	<i>stamac</i>	sta MAC address.
in	<i>uapmac</i>	uap MAC address.

5.13.2.7 net_stack_buffer_skip()

```
static uint8_t* net_stack_buffer_skip (
    void * buf,
    uint16_t in_offset ) [inline], [static]
```

Skip a number of bytes at the start of a stack buffer

Parameters

in	<i>buf</i>	input stack buffer.
in	<i>in_offset</i>	offset to skip.

Returns

the payload pointer after skip a number of bytes

5.13.2.8 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.

5.13.2.9 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	sock	socket descriptor
----	------	-------------------

Returns

[out] interface handle

5.13.2.10 net_wlan_init()

```
int net_wlan_init (
    void )
```

Initialize TCP/IP networking stack

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

5.13.2.11 net_wlan_deinit()

```
int net_wlan_deinit (
    void )
```

Deinitialize TCP/IP networking stack

Returns

WM_SUCCESS on success
-WM_FAIL otherwise

5.13.2.12 net_get_sta_interface()

```
struct netif* net_get_sta_interface (
    void )
```

Get STA interface netif structure pointer

A pointer to STA interface netif structure

5.13.2.13 net_get_uap_interface()

```
struct netif* net_get_uap_interface (
    void )
```

Get uAP interface netif structure pointer

A pointer to uAP interface netif structure

5.13.2.14 net_alloc_client_data_id()

```
int net_alloc_client_data_id ( )
```

Get client data index for storing private data in * netif.

Returns

allocated client data index, -1 if error or not supported.

5.13.2.15 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

5.13.2.16 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

5.13.2.17 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

5.13.2.18 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

5.13.2.19 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

5.13.2.20 net_interface_dhcp_cleanup()

```
void net_interface_dhcp_cleanup (
    void * intrfc_handle )
```

Cleanup DHCP client on given interface

Cleanup 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
----	----------------------	------------------

5.13.2.21 net_configure_address()

```
int net_configure_address (
    struct net_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.

5.13.2.22 net_configure_dns()

```
void net_configure_dns (
    struct net_ip_config * ip,
    unsigned int 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

5.13.2.23 net_get_if_addr()

```
int net_get_if_addr (
    struct net_ip_config * addr,
    void * intrfc_handle )
```

Get interface IP Address in [net_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>	net_ip_config
in	<i>intrfc_handle</i>	interface handle

Returns

WM_SUCCESS on success or error code.

5.13.2.24 net_get_if_ipv6_addr()

```
int net_get_if_ipv6_addr (
    struct net_ip_config * addr,
    void * intrfc_handle )
```

Get interface IPv6 Addresses & their states in [net_ip_config](#)

This function will get the IPv6 addresses & address states of a given interface. Use [net_get_sta_handle\(\)](#) to get interface handle.

Parameters

out	<i>addr</i>	net_ip_config
in	<i>intrfc_handle</i>	interface handle

Returns

WM_SUCCESS on success or error code.

5.13.2.25 net_get_if_ipv6_pref_addr()

```
int net_get_if_ipv6_pref_addr (
    struct net_ip_config * addr,
    void * intrfc_handle )
```

Get list of preferred IPv6 Addresses of a given interface in [net_ip_config](#)

This function will get the list of IPv6 addresses whose address state is Preferred. Use [net_get_sta_handle\(\)](#) to get interface handle.

Parameters

out	<i>addr</i>	net_ip_config
in	<i>intrfc_handle</i>	interface handle

Returns

Number of IPv6 addresses whose address state is Preferred

5.13.2.26 ipv6_addr_state_to_desc()

```
char* ipv6_addr_state_to_desc (
    unsigned char addr_state )
```

Get the description of IPv6 address state

This function will get the IPv6 address state description like - Invalid, Preferred, Deprecated

Parameters

in	<i>addr_state</i>	Address state
----	-------------------	---------------

Returns

IPv6 address state description

5.13.2.27 `ipv6_addr_addr_to_desc()`

```
char* ipv6_addr_addr_to_desc (  
    struct net\_ipv6\_config * ipv6_conf )
```

Get the description of IPv6 address

This function will get the IPv6 address type description like - Linklocal, Global, Sitelocal, Uniqueglobal

Parameters

in	<i>ipv6_conf</i>	Pointer to IPv6 configuration of type net_ipv6_config
----	------------------	---

Returns

IPv6 address description

5.13.2.28 `ipv6_addr_type_to_desc()`

```
char* ipv6_addr_type_to_desc (  
    struct net\_ipv6\_config * ipv6_conf )
```

Get the description of IPv6 address type

This function will get the IPv6 address type description like - Linklocal, Global, Sitelocal, Uniqueglobal

Parameters

in	<i>ipv6_conf</i>	Pointer to IPv6 configuration of type net_ipv6_config
----	------------------	---

Returns

IPv6 address type description

5.13.2.29 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.

5.13.2.30 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.

5.13.2.31 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.

Confidential

Parameters

in	<i>mask</i>	Subnet Mask pointer
in	<i>intrfc_handle</i>	interface

Returns

WM_SUCCESS on success or error code.

5.13.2.32 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.

5.13.2.33 net_stat()

```
void net_stat (  
    void )
```

Display network statistics

5.13.3 Macro Documentation**5.13.3.1 NET_SUCCESS**

```
#define NET_SUCCESS WM_SUCCESS
```

5.13.3.2 NET_ERROR

```
#define NET_ERROR (-WM_FAIL)
```

5.13.3.3 NET_ENOBUFS

```
#define NET_ENOBUFS ENOBUFS
```

5.13.3.4 NET_BLOCKING_OFF

```
#define NET_BLOCKING_OFF 1
```

5.13.3.5 NET_BLOCKING_ON

```
#define NET_BLOCKING_ON 0
```

5.13.3.6 net_socket

```
#define net_socket(  
    domain,  
    type,  
    protocol ) socket(domain, type, protocol)
```

5.13.3.7 net_select

```
#define net_select(  
    nfd,  
    read,  
    write,  
    except,  
    timeout ) select(nfd, read, write, except, timeout)
```

5.13.3.8 net_bind

```
#define net_bind(  
    sock,  
    addr,  
    len ) bind(sock, addr, len)
```

5.13.3.9 net_listen

```
#define net_listen(  
    sock,  
    backlog ) listen(sock, backlog)
```

5.13.3.10 net_close

```
#define net_close(  
    c ) close((c))
```

5.13.3.11 net_accept

```
#define net_accept(  
    sock,  
    addr,  
    len ) accept(sock, addr, len)
```

5.13.3.12 net_shutdown

```
#define net_shutdown(  
    c,  
    b ) shutdown(c, b)
```

5.13.3.13 net_connect

```
#define net_connect(  
    sock,  
    addr,  
    len ) connect(sock, addr, len)
```

5.13.3.14 net_read

```
#define net_read(  
    sock,  
    data,  
    len ) read(sock, data, len)
```

5.13.3.15 net_write

```
#define net_write(  
    sock,  
    data,  
    len ) write(sock, data, len)
```

5.13.3.16 net_get_mlan_handle

```
#define net_get_mlan_handle( ) net_get_sta_handle()
```

5.13.4 Enumeration Type Documentation

5.13.4.1 net_address_types

```
enum net_address_types
```

Enumerator

NET_ADDR_TYPE_STATIC	static IP address
NET_ADDR_TYPE_DHCP	Dynamic IP address
NET_ADDR_TYPE_LLA	Link level address

5.14 wm_utils.h File Reference

Utility functions.

5.14.1 Detailed Description

Collection of some common helper functions

5.14.2 Function Documentation

5.14.2.1 `wmpanic()`

```
NORETURN void wmpanic (
    void )
```

5.14.2.2 `wm_hex2bin()`

```
static unsigned int wm_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

5.14.2.3 `wm_bin2hex()`

```
void wm_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

5.14.2.4 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	<i>func</i>	Function pointer of type random_hdlr_t
----	-------------	--

Returns

WM_SUCCESS if successful
-WM_E_NOSPC if there is no space available for additional handlers

5.14.2.5 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	<i>func</i>	Function pointer of type random_hdlr_t used during registering
----	-------------	--

Returns

WM_SUCCESS if successful
-WM_E_INVALID if the passed pointer is invalid

5.14.2.6 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 random_hdlr_t
----	------	--

Returns

WM_SUCCESS if successful
-WM_E_NOSPC if there is no space available for additional handlers

5.14.2.7 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 random_hdlr_t used during registering
----	------	--

Returns

WM_SUCCESS if successful
-WM_E_INVALID if the passed pointer is invalid

5.14.2.8 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()`.

5.14.2.9 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

5.14.2.10 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

5.14.2.11 wm_frac_part_of()

```
static int wm_frac_part_of (
    float x,
    short precision ) [inline], [static]
```

5.14.2.12 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	<i>s</i>	Pointer to string to be duplicated
----	----------	------------------------------------

Returns

Pointer to newly allocated string which is duplicate of input string
NULL on error

5.14.2.13 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

5.14.2.14 wm_strtof()

```
float wm_strtof (
    const char * str,
    char ** endptr )
```

5.14.2.15 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.

5.14.2.16 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.

5.14.3 Macro Documentation

5.14.3.1 ffs

```
#define ffs __builtin_ffs
```

5.14.3.2 WARN_UNUSED_RET

```
#define WARN_UNUSED_RET
```

5.14.3.3 PACK_START

```
#define PACK_START __packed
```

5.14.3.4 PACK_END

```
#define PACK_END
```

5.14.3.5 NORETURN

```
#define NORETURN
```

5.14.3.6 __WM_ALIGN__

```
#define __WM_ALIGN__(  
    num,  
    num_type,  
    align ) WM_MASK(num, (num_type)align - 1)
```

5.14.3.7 WM_MASK

```
#define WM_MASK(  
    num,  
    mask ) ((num + mask) & ~(mask))
```

5.14.3.8 dump_hex

```
#define dump_hex(  
    ... )
```

Value:

```
do  
{  
    while (0)
```

5.14.3.9 dump_hex_ascii

```
#define dump_hex_ascii(  
    ... )
```

Value:

```
do  
{  
    while (0)
```

5.14.3.10 dump_ascii

```
#define dump_ascii(  
    ... )
```

Value:

```
do  
{  
    while (0)
```

5.14.3.11 print_ascii

```
#define print_ascii(  
    ... )
```

Value:

```
do  
{  
    } while (0)
```

5.14.3.12 dump_json

```
#define dump_json(  
    ... )
```

Value:

```
do  
{  
    } while (0)
```

5.14.3.13 wm_int_part_of

```
#define wm_int_part_of(  
    x ) ((int)(x))
```

5.14.4 Typedef Documentation

5.14.4.1 random_hdlr_t

```
typedef uint32_t(* random_hdlr_t) (void)
```

Function prototype for a random entropy/seed generator

Returns

a 32bit random number

Index

`_Cipher_t`, 9

- `aes_128_cmac`, 10
- `bip_cmac_256`, 11
- `bip_gmac_128`, 11
- `bip_gmac_256`, 11
- `ccmp`, 10
- `ccmp_256`, 10
- `gcmp`, 10
- `gcmp_256`, 10
- `gtk_not_used`, 11
- `none`, 9
- `rsvd`, 11
- `rsvd2`, 11
- `sms4`, 10
- `tkip`, 10
- `wep104`, 10
- `wep40`, 9

`_SecurityMode_t`, 12

- `ft_1x`, 13
- `ft_1x_sha384`, 14
- `ft_psk`, 14
- `ft_sae`, 14
- `noRsn`, 12
- `owe`, 13
- `rsvd`, 14
- `wepDynamic`, 12
- `wepStatic`, 12
- `wpa`, 12
- `wpa2`, 13
- `wpa2_entp`, 13
- `wpa2_entp_sha256`, 13
- `wpa2_sha256`, 13
- `wpa3_1x_sha256`, 14
- `wpa3_1x_sha384`, 14
- `wpa3_sae`, 13
- `wpaNone`, 13

`__WM_ALIGN__`

- `wm_utils.h`, 418

`_rw_lock`

- `OSA_MUTEX_HANDLE_DEFINE`, 25, 26
- `OSA_SEMAPHORE_HANDLE_DEFINE`, 26
- `reader_cb`, 26
- `reader_count`, 26

`_wifi_set_mac_addr`

- `wifi.h`, 196

`A_ID_MAX_LENGTH`

- `wlan.h`, 370

`a_id`

- `wlan_network_security`, 137

`ACTION_GET`

- `wlan.h`, 368

`ACTION_SET`

- `wlan.h`, 368

`ARG_UNUSED`

- `wlan.h`, 367

`acs_band`

- `wlan_network`, 124

`act_sub_ch`

- `wifi_mfg_cmd_tx_cont_t`, 73
- `wifi_mfg_cmd_tx_frame_t`, 76

`action`

- `wifi_btwt_config_t`, 35
- `wifi_mef_entry_t`, 60
- `wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t`, 67
- `wifi_mfg_cmd_generic_cfg_t`, 64
- `wifi_mfg_cmd_he_tb_tx_t`, 65
- `wifi_mfg_cmd_otp_cal_data_rd_wr_t`, 69
- `wifi_mfg_cmd_otp_mac_addr_rd_wr_t`, 71
- `wifi_mfg_cmd_tx_cont_t`, 72
- `wifi_mfg_cmd_tx_frame_t`, 74
- `wnm_sleep_result_t`, 150

`addr1`

- `wifi_mgmt_frame_t`, 79

`addr2`

- `wifi_mgmt_frame_t`, 79

`addr3`

- `wifi_mgmt_frame_t`, 79

`addr4`

- `wifi_mgmt_frame_t`, 79

`addr_state`

- `ipv6_config`, 22
- `net_ipv6_config`, 25

`addr_type`

- `ipv4_config`, 20
- `ipv6_config`, 22
- `net_ipv4_config`, 23
- `net_ipv6_config`, 25

`address`

- `ipv4_config`, 20
- `ipv6_config`, 21
- `net_ipv4_config`, 23
- `net_ipv6_config`, 25

`address_types`

- `wlan.h`, 393

`adhoc_awake_period`

- `wlan_ieeeeps_config`, 120

`adjust_burst_sifs`

- `wifi_mfg_cmd_tx_frame_t`, 75

- adv_coding
 - txrate_setting, 30
 - wifi_mfg_cmd_tx_frame_t, 76
- aes_128_cmac
 - _Cipher_t, 10
 - wlan_cipher, 117
- aid
 - wifi_mfg_cmd_he_tb_tx_t, 66
- alloc_cnt
 - wifi_os_mem_info, 81
- announced
 - wifi_twt_setup_config_t, 108
- anonymous_identity
 - wlan_network_security, 133
- ant_mode
 - wifi_antcfg_t, 32
- ap_mfpc
 - wifi_scan_result2, 97
 - wlan_scan_result, 149
- ap_mfpr
 - wifi_scan_result2, 97
 - wlan_scan_result, 149
- ap_pwe
 - wifi_scan_result2, 97
 - wlan_scan_result, 149
- avg_tbt_offset
 - wifi_tbt_offset_t, 103
- axq_mu_timer
 - wifi_mfg_cmd_he_tb_tx_t, 66
- BAND_SPECIFIED
 - wifi-decl.h, 179
- BANDWIDTH_20MHZ
 - wifi.h, 233
- BANDWIDTH_40MHZ
 - wifi.h, 233
- BANDWIDTH_80MHZ
 - wifi.h, 233
- BEACON_REPORT_BUF_SIZE
 - wifi.h, 234
- BIN_COUNTER_LEN
 - wlan.h, 378
- BIT
 - wifi-decl.h, 174
- BSS_TYPE_STA
 - wifi-decl.h, 173
- BSS_TYPE_UAP
 - wifi-decl.h, 173
- band
 - wifi_11ax_config_t, 31
 - wifi_scan_result2, 98
- band_config
 - wifi_csi_config_params_t, 46
 - wifi_ecsa_info, 52
- bandConfig
 - ChanBandInfo_t, 16
- BandConfig_t, 15
 - chan2Offset, 15
 - chanBand, 15
 - chanWidth, 15
 - scanMode, 15
- bandcfg
 - wifi_remain_on_channel_t, 84
- bandwidth
 - txrate_setting, 29
- basic_trig_user_info
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 69
- bcn_miss_timeout
 - wlan_ieeeeps_config, 120
- bcn_nf_avg
 - wifi_rssi_info_t, 87
- bcn_nf_last
 - wifi_rssi_info_t, 87
- bcn_rssi_avg
 - wifi_rssi_info_t, 87
- bcn_rssi_last
 - wifi_rssi_info_t, 87
- bcn_snr_avg
 - wifi_rssi_info_t, 87
- bcn_snr_last
 - wifi_rssi_info_t, 86
- beacon_period
 - wifi_scan_result2, 96
 - wlan_network, 127
 - wlan_scan_result, 149
- BeamChange
 - wifi_mfg_cmd_tx_frame_t, 77
- binCounter
 - Event_Radar_Detected_Info, 19
- bip_cmac_256
 - _Cipher_t, 11
 - wlan_cipher, 118
- bip_gmac_128
 - _Cipher_t, 11
 - wlan_cipher, 118
- bip_gmac_256
 - _Cipher_t, 11
 - wlan_cipher, 118
- bits_field
 - wlan_rrm_beacon_report_data, 141
- bss_transition_supported
 - wifi_scan_result2, 99
 - wlan_network, 127
 - wlan_scan_result, 149
- bss_type
 - wifi_csi_config_params_t, 45
 - wifi_ecsa_info, 52
 - wifi_scan_params_t, 92
- bssid
 - wifi_mfg_cmd_tx_frame_t, 75
 - wifi_scan_params_t, 91
 - wifi_scan_params_v2_t, 93
 - wifi_scan_result2, 95
 - wlan_network, 123
 - wlan_nlist_report_param, 139
 - wlan_rrm_beacon_report_data, 140
 - wlan_rrm_neighbor_ap_t, 141

- wlan_scan_result, 145
- bssid_specific
 - wlan_network, 125
- bssidInfo
 - wlan_rrm_neighbor_ap_t, 141
- btm_mode
 - wlan_network, 127
 - wlan_nlist_report_param, 139
- burst_sifs_in_us
 - wifi_mfg_cmd_tx_frame_t, 76
- byte_seq
 - wifi_mef_filter_t, 62
- CARD_WAKEUP_GPIO_PIN
 - wlan.h, 372
- CONFIG_APP_FRM_CLI_HISTORY
 - cli.h, 155
- CONFIG_FW_VDLL
 - wifi.h, 232
- CONFIG_GTK_REKEY_OFFLOAD
 - wifi.h, 232
- CONFIG_STA_AUTO_DHCPV4
 - wifi.h, 231
- CONFIG_TCP_ACK_ENH
 - wifi.h, 232
- CONFIG_WIFI_AUTO_POWER_SAVE
 - wifi.h, 232
- CONFIG_WIFI_STA_RECONNECT
 - wifi.h, 231
- CONFIG_WLAN_KNOWN_NETWORKS
 - wlan.h, 367
- CRITERIA_BROADCAST
 - wifi-decl.h, 175
- CRITERIA_MULTICAST
 - wifi-decl.h, 176
- CRITERIA_UNICAST
 - wifi-decl.h, 176
- CSI_FILTER_MAX
 - wifi-decl.h, 180
- ca_cert2_data
 - wlan_network_security, 135
- ca_cert2_len
 - wlan_network_security, 135
- ca_cert_data
 - wlan_network_security, 133
- ca_cert_hash
 - wlan_network_security, 134
- ca_cert_len
 - wlan_network_security, 134
- cached
 - wifi_cloud_keep_alive_t, 43
- cal_data
 - wifi_mfg_cmd_otp_cal_data_rd_wr_t, 70
- cal_data_len
 - wifi_mfg_cmd_otp_cal_data_rd_wr_t, 70
- cal_data_status
 - wifi_mfg_cmd_otp_cal_data_rd_wr_t, 70
- cb
 - wifi_scan_params_v2_t, 94
- cb_fn
 - osa.h, 171
- ccmp
 - _Cipher_t, 10
 - wlan_cipher, 117
- ccmp_256
 - _Cipher_t, 10
 - wlan_cipher, 118
- chan2Offset
 - BandConfig_t, 15
- Chan2Offset_e
 - wlan.h, 391
- chan_desc
 - wifi_txpwrlimit_config_t, 112
- chan_freq
 - wifi_chan_info_t, 37
- chan_info
 - wifi_chanlist_t, 40
- chan_list
 - wifi_scan_params_v2_t, 93
- chan_num
 - wifi_chan_info_t, 37
 - wifi_channel_desc_t, 41
 - wifi_rupwrlimit_config_t, 88
- chan_number
 - wifi_chan_scan_param_set_t, 39
 - wifi_scan_chan_list_t, 90
 - wifi_scan_channel_list_t, 91
- chan_scan_param
 - wifi_chan_list_param_set_t, 38
- chan_width
 - wifi_channel_desc_t, 41
- chanBand
 - BandConfig_t, 15
- ChanBand_e
 - wlan.h, 390
- ChanBandInfo_t, 15
 - bandConfig, 16
 - chanNum, 16
- chanInfo
 - wifi_cw_mode_ctrl_t, 48
- chanNum
 - ChanBandInfo_t, 16
- chanWidth
 - BandConfig_t, 15
- ChanWidth_e
 - wlan.h, 390
- Channel
 - wifi_scan_result2, 96
- channel
 - wifi_csi_config_params_t, 46
 - wifi_cw_mode_ctrl_t, 48
 - wifi_ecsa_info, 52
 - wifi_remain_on_channel_t, 84
 - wifi_scan_params_t, 92
 - wlan_network, 124
 - wlan_rrm_beacon_report_data, 140
 - wlan_rrm_neighbor_ap_t, 142

- wlan_scan_result, 145
- channel_num
 - wlan_rrm_beacon_report_data, 140
- channel_specific
 - wlan_network, 126
- channels
 - wlan_nlist_report_param, 138
- chip_id
 - wifi_csi_config_params_t, 46
- clear_event_chanswann
 - wifi.h, 205
- cli.h, 151
 - CONFIG_APP_FRM_CLI_HISTORY, 155
 - cli_add_history_hook, 155
 - cli_deinit, 152
 - cli_get_cmd_buffer, 154
 - cli_init, 152
 - cli_name_val_get, 155
 - cli_name_val_set, 155
 - cli_register_command, 151
 - cli_register_commands, 153
 - cli_stop, 153
 - cli_submit_cmd_buffer, 154
 - cli_unregister_command, 152
 - cli_unregister_commands, 153
 - help_command, 155
 - lookup_command, 151
- cli_add_history_hook
 - cli.h, 155
- cli_command, 16
 - function, 16
 - help, 16
 - name, 16
- cli_deinit
 - cli.h, 152
- cli_get_cmd_buffer
 - cli.h, 154
- cli_init
 - cli.h, 152
- cli_name_val_get
 - cli.h, 155
- cli_name_val_set
 - cli.h, 155
- cli_register_command
 - cli.h, 151
- cli_register_commands
 - cli.h, 153
- cli_stop
 - cli.h, 153
- cli_submit_cmd_buffer
 - cli.h, 154
- cli_unregister_command
 - cli.h, 152
- cli_unregister_commands
 - cli.h, 153
- client_cert2_data
 - wlan_network_security, 135
- client_cert2_len
 - wlan_network_security, 135
- client_cert_data
 - wlan_network_security, 134
- client_cert_len
 - wlan_network_security, 134
- client_key2_data
 - wlan_network_security, 135
- client_key2_len
 - wlan_network_security, 135
- client_key2_passwd
 - wlan_network_security, 135
- client_key_data
 - wlan_network_security, 134
- client_key_len
 - wlan_network_security, 134
- client_key_passwd
 - wlan_network_security, 134
- clock_sync_Role
 - wifi_clock_sync_gpio_tsf_t, 42
- clock_sync_gpio_level_toggle
 - wifi_clock_sync_gpio_tsf_t, 42
- clock_sync_gpio_pin_number
 - wifi_clock_sync_gpio_tsf_t, 42
- clock_sync_gpio_pulse_width
 - wifi_clock_sync_gpio_tsf_t, 42
- clock_sync_mode
 - wifi_clock_sync_gpio_tsf_t, 41
- config_bands
 - wifi_bandcfg_t, 34
- count
 - wifi_sta_list_t, 101
- criteria
 - wififlt_cfg_t, 56
- cs_mode
 - wifi_mfg_cmd_tx_cont_t, 73
- csi_deliver_data_to_user
 - wifi.h, 228
- csi_enable
 - wifi_csi_config_params_t, 45
- csi_event_cnt
 - wifi.h, 238
- csi_event_data_len
 - wifi.h, 238
- csi_filter
 - wifi_csi_config_params_t, 46
- csi_filter_cnt
 - wifi_csi_config_params_t, 46
- csi_local_buff_init
 - wifi.h, 228
- csi_local_buff_statu, 17
 - OSA_SEMAPHORE_HANDLE_DEFINE, 17
 - read_index, 17
 - valid_data_cnt, 18
 - write_index, 17
- csi_monitor_enable
 - wifi_csi_config_params_t, 46
- csi_save_data_to_local_buff
 - wifi.h, 228

- current_antenna
 - wifi_antcfg_t, 33
- current_channel
 - wifi_rf_channel_t, 85
- current_level
 - wifi_tx_power_t, 111
- cw_mode
 - wifi_mfg_cmd_tx_cont_t, 72
- DFS_REC_HDR_LEN
 - wlan.h, 378
- DFS_REC_HDR_NUM
 - wlan.h, 378
- DOMAIN_MATCH_MAX_LENGTH
 - wlan.h, 370
- data
 - test_cfg_table_t, 28
 - wifi_cal_data_t, 37
 - wifi_message, 63
 - wifi_twt_report_t, 107
 - wlan_message, 121
- data1
 - wifi_mfg_cmd_generic_cfg_t, 64
- data2
 - wifi_mfg_cmd_generic_cfg_t, 65
- data3
 - wifi_mfg_cmd_generic_cfg_t, 65
- data_len
 - wifi_cal_data_t, 36
- data_nf_avg
 - wifi_rssi_info_t, 86
- data_nf_last
 - wifi_rssi_info_t, 86
- data_rate
 - wifi_ds_rate, 51
 - wifi_mfg_cmd_tx_frame_t, 75
- data_rssi_avg
 - wifi_rssi_info_t, 86
- data_rssi_last
 - wifi_rssi_info_t, 86
- data_snr_avg
 - wifi_rssi_info_t, 87
- data_snr_last
 - wifi_rssi_info_t, 87
- Dcm
 - wifi_mfg_cmd_tx_frame_t, 77
- dcm
 - txrate_setting, 30
- DefaultPriority
 - wifi_ext_coex_config_t, 54
- delay_to_ps
 - wlan_ieeeeps_config, 120
- dest_addr
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
- detect_count
 - Event_Radar_Detected_Info, 18
- device_id
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 67
 - wifi_mfg_cmd_generic_cfg_t, 64
- wifi_mfg_cmd_he_tb_tx_t, 66
- wifi_mfg_cmd_otp_cal_data_rd_wr_t, 70
- wifi_mfg_cmd_otp_mac_addr_rd_wr_t, 71
- wifi_mfg_cmd_tx_cont_t, 72
- wifi_mfg_cmd_tx_frame_t, 75
- dfsRecordHdrs
 - Event_Radar_Detected_Info, 19
- dh_data
 - wlan_network_security, 136
- dh_len
 - wlan_network_security, 136
- dhcp-server.h, 156
 - dhcp_enable_dns_server, 157
 - dhcp_get_ip_from_mac, 158
 - dhcp_server_lease_timeout, 158
 - dhcp_server_start, 156
 - dhcp_server_stop, 157
 - dhcp_stat, 158
 - dhcpcd_cli_deinit, 156
 - dhcpcd_cli_init, 156
 - MAX_QNAME_SIZE, 159
 - wm_dhcpcd_errno, 159
- dhcp_enable_dns_server
 - dhcp-server.h, 157
- dhcp_get_ip_from_mac
 - dhcp-server.h, 158
- dhcp_server_lease_timeout
 - dhcp-server.h, 158
- dhcp_server_start
 - dhcp-server.h, 156
- dhcp_server_stop
 - dhcp-server.h, 157
- dhcp_stat
 - dhcp-server.h, 158
- dhcpcd_cli_deinit
 - dhcp-server.h, 156
- dhcpcd_cli_init
 - dhcp-server.h, 156
- dialog_tok
 - wlan_rrm_scan_cb_param, 143
- dialog_token
 - wlan_nlist_report_param, 139
- dns1
 - ipv4_config, 21
 - net_ipv4_config, 24
- dns2
 - ipv4_config, 21
 - net_ipv4_config, 24
- domain_match
 - wlan_network_security, 134
- domain_suffix_match
 - wlan_network_security, 135
- Doppler
 - wifi_mfg_cmd_tx_frame_t, 77
- doppler
 - txrate_setting, 30
- dot11ac
 - wlan_network, 126

- wlan_scan_result, 146
- dot11ax
 - wlan_network, 126
 - wlan_scan_result, 146
- dot11n
 - wlan_network, 126
 - wlan_scan_result, 145
- dpp_c_sign_key
 - wlan_network_security, 137
- dpp_connector
 - wlan_network_security, 137
- dpp_net_access_key
 - wlan_network_security, 138
- dst_addr
 - wlan_nlist_report_param, 139
 - wlan_rrm_scan_cb_param, 143
- dst_ip
 - wifi_cloud_keep_alive_t, 44
 - wifi_nat_keep_alive_t, 80
 - wifi_tcp_keep_alive_t, 104
- dst_mac
 - wifi_cloud_keep_alive_t, 44
 - wifi_nat_keep_alive_t, 80
 - wifi_tcp_keep_alive_t, 104
- dst_port
 - wifi_cloud_keep_alive_t, 44
 - wifi_nat_keep_alive_t, 80
- dst_tcp_port
 - wifi_tcp_keep_alive_t, 104
- dtim_period
 - wifi_scan_result2, 96
 - wlan_network, 127
 - wlan_scan_result, 149
- dump_ascii
 - wm_utils.h, 419
- dump_hex
 - wm_utils.h, 419
- dump_hex_ascii
 - wm_utils.h, 419
- dump_json
 - wm_utils.h, 420
- duration
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
 - wlan_rrm_beacon_report_data, 140
- duration_id
 - wifi_mgmt_frame_t, 79
- ENH_PS_MODES
 - wlan.h, 389
- EU_CRYPT0_AAD_MAX_LENGTH
 - wlan.h, 380
- EU_CRYPT0_DATA_MAX_LENGTH
 - wlan.h, 379
- EU_CRYPT0_KEY_MAX_LENGTH
 - wlan.h, 379
- EU_CRYPT0_KEYIV_MAX_LENGTH
 - wlan.h, 379
- EU_CRYPT0_NONCE_MAX_LENGTH
 - wlan.h, 380
- EXT_RADIO_PRI_ip_gpio_num
 - wifi_ext_coex_config_t, 54
- EXT_RADIO_PRI_ip_gpio_polarity
 - wifi_ext_coex_config_t, 54
- EXT_RADIO_REQ_ip_gpio_num
 - wifi_ext_coex_config_t, 54
- EXT_RADIO_REQ_ip_gpio_polarity
 - wifi_ext_coex_config_t, 54
- eap_crypto_binding
 - wlan_network_security, 133
- eap_password
 - wlan_network_security, 133
- eap_result_ind
 - wlan_network_security, 133
- eap_tls_cipher_type
 - wlan.h, 392
- eap_ver
 - wlan_network_security, 132
- ed_ctrl_2g
 - wifi_ed_mac_ctrl_t, 52
- ed_ctrl_5g
 - wifi_ed_mac_ctrl_t, 53
- ed_offset_2g
 - wifi_ed_mac_ctrl_t, 53
- ed_offset_5g
 - wifi_ed_mac_ctrl_t, 53
- enable
 - wifi_cloud_keep_alive_t, 43
 - wifi_mfg_cmd_he_tb_tx_t, 66
 - wifi_mfg_cmd_tx_frame_t, 75
 - wifi_tcp_keep_alive_t, 103
 - wifi_wowlan_ptn_cfg_t, 116
- enable_tx
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
 - wifi_mfg_cmd_tx_cont_t, 72
- Enabled
 - wifi_ext_coex_config_t, 54
- error
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
 - wifi_mfg_cmd_generic_cfg_t, 64
 - wifi_mfg_cmd_he_tb_tx_t, 66
 - wifi_mfg_cmd_otp_cal_data_rd_wr_t, 70
 - wifi_mfg_cmd_otp_mac_addr_rd_wr_t, 71
 - wifi_mfg_cmd_tx_cont_t, 72
 - wifi_mfg_cmd_tx_frame_t, 75
- evaluate_time
 - wifi_antcfg_t, 33
- event
 - wifi_message, 63
- Event_Radar_Detected_Info, 18
 - binCounter, 19
 - detect_count, 18
 - dfsRecordHdrs, 19
 - main_det_type, 18
 - numDfsRecords, 19
 - pri_binCnt, 19
 - pri_radar_type, 19
 - pw_chirp_idx, 19

- pw_chirp_type, 19
- pw_value, 19
- reallyPassed, 20
- reg_domain, 18
- ext_id
 - wifi_11ax_config_t, 31
- ext_radio_pri_count
 - wifi_ext_coex_stats_t, 56
- ext_radio_req_count
 - wifi_ext_coex_stats_t, 55
- FILE_TYPE_ENTP_CA_CERT2
 - wlan.h, 380
- FILE_TYPE_ENTP_CA_CERT
 - wlan.h, 380
- FILE_TYPE_ENTP_CLIENT_CERT2
 - wlan.h, 380
- FILE_TYPE_ENTP_CLIENT_CERT
 - wlan.h, 380
- FILE_TYPE_ENTP_CLIENT_KEY2
 - wlan.h, 381
- FILE_TYPE_ENTP_CLIENT_KEY
 - wlan.h, 380
- FILE_TYPE_ENTP_DH_PARAMS
 - wlan.h, 381
- FILE_TYPE_ENTP_SERVER_CERT
 - wlan.h, 381
- FILE_TYPE_ENTP_SERVER_KEY
 - wlan.h, 381
- FILE_TYPE_NONE
 - wlan.h, 380
- FILLING_BYTE_SEQ
 - wifi-decl.h, 179
- FILLING_MASK_SEQ
 - wifi-decl.h, 179
- FILLING_NUM_BYTES
 - wifi-decl.h, 178
- FILLING_OFFSET
 - wifi-decl.h, 178
- FILLING_PATTERN
 - wifi-decl.h, 178
- FILLING_REPEAT
 - wifi-decl.h, 178
- FILLING_TYPE
 - wifi-decl.h, 178
- fast_prov
 - wlan_network_security, 137
- ffs
 - wm_utils.h, 418
- fill_flag
 - wifi_mef_filter_t, 61
- fill_sequential_pattern
 - wm_utils.h, 417
- filter_item
 - wifi_mef_entry_t, 61
- filter_num
 - wifi_mef_entry_t, 60
- first_chan
 - wifi_sub_band_set_t, 101
- flags
 - wifi_auto_reconnect_config_t, 34
 - wifi_csi_filter_t, 47
- flow_identifier
 - wifi_twt_information_t, 106
 - wifi_twt_setup_config_t, 109
 - wifi_twt_teardown_config_t, 110
- frame_ctrl_flags
 - wifi_mgmt_frame_t, 79
- frame_length
 - wifi_mfg_cmd_tx_frame_t, 75
- frame_pattern
 - wifi_mfg_cmd_tx_frame_t, 75
- frame_type
 - wifi_frame_t, 57
 - wifi_mgmt_frame_t, 78
- free_cnt
 - wifi_os_mem_info, 81
- freq
 - wlan_rrm_neighbor_ap_t, 142
- frm_len
 - wifi_mgmt_frame_t, 78
- frmCtl
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
- ft_1x
 - _SecurityMode_t, 13
 - wlan_network, 127
 - wlan_scan_result, 147
- ft_1x_sha384
 - _SecurityMode_t, 14
 - wlan_scan_result, 148
- ft_psk
 - _SecurityMode_t, 14
 - wlan_network, 127
 - wlan_scan_result, 148
- ft_sae
 - _SecurityMode_t, 14
 - wlan_network, 127
 - wlan_scan_result, 148
- function
 - cli_command, 16
- fw_bands
 - wifi_bandcfg_t, 34
- g_bcn_nf_last
 - wifi.h, 237
- g_data_nf_last
 - wifi.h, 237
- g_data_snr_last
 - wifi.h, 237
- g_osa_idle_hooks
 - osa.h, 171
- g_osa_tick_hooks
 - osa.h, 171
- g_rssi
 - wifi.h, 237
- gcmp
 - _Cipher_t, 10
 - wlan_cipher, 117

- gcmp_256
 - _Cipher_t, 10
 - wlan_cipher, 118
- get_random_sequence
 - wm_utils.h, 415
- get_scan_params
 - wlan.h, 245
- get_sub_band_from_region_code
 - wifi.h, 203
- get_sub_band_from_region_code_5ghz
 - wifi.h, 203
- gf_mode
 - wifi_mfg_cmd_tx_frame_t, 76
- gpio_pin
 - wifi_inrst_cfg_t, 59
- group_cipher
 - wlan_network_security, 130
- group_mgmt_cipher
 - wlan_network_security, 130
- gtk_not_used
 - _Cipher_t, 11
 - wlan_cipher, 118
- gw
 - ipv4_config, 20
 - net_ipv4_config, 24
- HASH_MAX_LENGTH
 - wlan.h, 370
- HOST_WAKEUP_GPIO_PIN
 - wlan.h, 372
- hard_constraint
 - wifi_twt_setup_config_t, 109
- he_mac_cap
 - wifi_11ax_config_t, 31
- he_oper_chwidth
 - wlan_network, 125
- he_phy_cap
 - wifi_11ax_config_t, 32
- he_txrx_mcs_support
 - wifi_11ax_config_t, 32
- head_id
 - wifi_csi_config_params_t, 45
- help
 - cli_command, 16
- help_command
 - cli.h, 155
- Host_Sleep_Action
 - wlan.h, 389
- hostapd_connected_sta_list
 - wifi.h, 231
- ht_capab
 - wlan_network, 124
- ICMP_OF_IP_PROTOCOL
 - wifi-decl.h, 177
- IDENTITY_MAX_LENGTH
 - wlan.h, 370
- IEEEtypes_ADDRESS_SIZE
 - wlan.h, 368
- IEEEtypes_Bss_t
 - wlan.h, 385
- IEEEtypes_ElementId_t
 - wifi.h, 235
- IEEEtypes_SSID_SIZE
 - wlan.h, 368
- IP_PROTOCOL_OFFSET
 - wifi-decl.h, 178
- IPV4_PKT_OFFSET
 - wifi-decl.h, 178
- id
 - wifi_11ax_config_t, 31
 - wlan_message, 121
 - wlan_network, 123
- identities
 - wlan_network_security, 137
- identity
 - wlan_network_security, 133
- IgnorePriority
 - wifi_ext_coex_config_t, 54
- implicit
 - wifi_twt_setup_config_t, 108
- information_state
 - wifi_twt_information_t, 106
- interval
 - wifi_nat_keep_alive_t, 80
 - wifi_tcp_keep_alive_t, 104
- ip
 - wlan_network, 125
- iperf.h, 160
 - iperf_cli_deinit, 160
 - iperf_cli_init, 160
 - iperf_e, 160
 - iperf_w, 160
- iperf_cli_deinit
 - iperf.h, 160
- iperf_cli_init
 - iperf.h, 160
- iperf_e
 - iperf.h, 160
- iperf_w
 - iperf.h, 160
- ipv4
 - net_ip_config, 23
 - wlan_ip_config, 121
- ipv4_config, 20
 - addr_type, 20
 - address, 20
 - dns1, 21
 - dns2, 21
 - gw, 20
 - netmask, 21
- ipv6
 - net_ip_config, 22
 - wlan_ip_config, 121
- ipv6_addr_addr_to_desc
 - wm_net.h, 405
- ipv6_addr_state_to_desc

- wm_net.h, 404
- ipv6_addr_type_to_desc
 - wm_net.h, 405
- ipv6_config, 21
 - addr_state, 22
 - addr_type, 22
 - address, 21
- ipv6_count
 - net_ip_config, 22
 - wlan_ip_config, 121
- ir_mode
 - wifi_inrst_cfg_t, 59
- is_bssid
 - wifi_scan_params_v2_t, 93
- is_ep_valid_security
 - wlan.h, 244
- is_ibss_bit_set
 - wifi_scan_result2, 95
- is_pmf_required
 - wifi_scan_result2, 97
 - wlan_network_security, 130
- is_ssid
 - wifi_scan_params_v2_t, 93
- is_sta_connected
 - wlan.h, 259
- is_sta_ipv4_connected
 - wlan.h, 259
- is_sta_ipv6_connected
 - wlan.h, 259
- is_uap_started
 - wlan.h, 258
- is_valid_security
 - wlan.h, 244
- key_mgmt
 - wlan_network_security, 129
- last_ind
 - wlan_rrm_beacon_report_data, 140
- len
 - test_cfg_param_t, 27
 - test_cfg_table_t, 28
 - wifi_11ax_config_t, 31
- length
 - wifi_twt_report_t, 107
- line_num
 - wifi_os_mem_info, 81
- listen_interval
 - wlan_ieeeeps_config, 119
- lookup_command
 - cli.h, 151
- MAX_CHANNEL_LIST
 - wifi-decl.h, 179
 - wlan.h, 379
- MAX_CUSTOM_HOOKS
 - osa.h, 170
- MAX_FUNC_SYMBOL_LEN
 - wifi-decl.h, 180
- MAX_NEIGHBOR_AP_LIMIT
 - wifi.h, 235
- MAX_NUM_BYTE_SEQ
 - wifi-decl.h, 176
- MAX_NUM_CHANS_IN_NBOR_RPT
 - wifi.h, 234
- MAX_NUM_ENTRIES
 - wifi-decl.h, 176
- MAX_NUM_FILTERS
 - wifi-decl.h, 174
- MAX_NUM_MASK_SEQ
 - wifi-decl.h, 176
- MAX_NUM_SSID
 - wifi-decl.h, 179
- MAX_OPERAND
 - wifi-decl.h, 176
- MAX_QNAME_SIZE
 - dhcp-server.h, 159
- MAX_USERS
 - wlan.h, 370
- MBIT
 - wifi.h, 234
- MEF_ACTION_ALLOW_AND_WAKEUP_HOST
 - wifi-decl.h, 175
- MEF_ACTION_ALLOW
 - wifi-decl.h, 175
- MEF_ACTION_WAKE
 - wifi-decl.h, 175
- MEF_AUTO_ARP
 - wifi-decl.h, 175
- MEF_AUTO_PING
 - wifi-decl.h, 175
- MEF_MAGIC_PKT
 - wifi-decl.h, 175
- MEF_MODE_HOST_SLEEP
 - wifi-decl.h, 174
- MEF_MODE_NON_HOST_SLEEP
 - wifi-decl.h, 174
- MEF_NS_RESP
 - wifi-decl.h, 175
- MKEEP_ALIVE_IP_PKT_MAX
 - wifi-decl.h, 179
- MLAN_MAC_ADDR_LENGTH
 - wifi-decl.h, 171
- MLAN_MAX_PASS_LENGTH
 - wifi-decl.h, 174
- MLAN_MAX_SSID_LENGTH
 - wifi-decl.h, 173
- MLAN_MAX_VER_STR_LEN
 - wifi-decl.h, 172
- MOD_GROUPS
 - wifi-decl.h, 172
- mac
 - wifi_mac_addr_t, 60
 - wifi_sta_info_t, 100
- mac_addr
 - wifi_csi_filter_t, 47
 - wifi_mfg_cmd_otp_mac_addr_rd_wr_t, 71

- main_det_type
 - Event_Radar_Detected_Info, 18
- mask
 - wifi_wowlan_pattern_t, 115
- mask_seq
 - wifi_mef_filter_t, 63
- max_keep_alives
 - wifi_tcp_keep_alive_t, 104
- max_pkttext
 - txrate_setting, 30
- max_power
 - wifi_tx_power_t, 111
- max_scan_time
 - wifi_chan_scan_param_set_t, 39
- max_sta_support
 - wifi_btwt_config_t, 35
- max_tbtt_offset
 - wifi_tbtt_offset_t, 102
- max_tx_pwr
 - wifi_sub_band_set_t, 102
- MaxPE
 - wifi_mfg_cmd_tx_frame_t, 77
- mbo_assoc_disallowed
 - wifi_scan_result2, 99
- mcstCipher
 - wlan_network_security, 130
- mdid
 - wifi_scan_result2, 99
 - wlan_network, 126
- mef_entry
 - wififlt_cfg_t, 57
- mfg_cmd
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 67
 - wifi_mfg_cmd_generic_cfg_t, 64
 - wifi_mfg_cmd_he_tb_tx_t, 65
 - wifi_mfg_cmd_otp_cal_data_rd_wr_t, 69
 - wifi_mfg_cmd_otp_mac_addr_rd_wr_t, 71
 - wifi_mfg_cmd_tx_cont_t, 72
 - wifi_mfg_cmd_tx_frame_t, 74
- mfp
 - wifi_pmf_params_t, 82
 - wlan_network_security, 132
- mfp
 - wifi_pmf_params_t, 82
 - wlan_network_security, 132
- MidP
 - wifi_mfg_cmd_tx_frame_t, 77
- min_power
 - wifi_tx_power_t, 111
- min_scan_time
 - wifi_chan_scan_param_set_t, 39
- min_tbtt_offset
 - wifi_tbtt_offset_t, 102
- mkeep_alive_id
 - wifi_cloud_keep_alive_t, 43
- mod_group
 - wifi_txpwrlimit_entry_t, 113
- mode
 - tx_ampdu_prot_mode_para, 29
 - wifi_cw_mode_ctrl_t, 48
 - wifi_mef_entry_t, 60
- multiple_dtim_interval
 - wlan_jeeps_config, 119
- n_patterns
 - wifi_wowlan_ptn_cfg_t, 116
- NET_BLOCKING_OFF
 - wm_net.h, 409
- NET_BLOCKING_ON
 - wm_net.h, 409
- NET_ENOBUFS
 - wm_net.h, 409
- NET_ERROR
 - wm_net.h, 408
- NET_SUCCESS
 - wm_net.h, 408
- NORETURN
 - wm_utils.h, 418
- NUM_CHAN_BAND_ENUMS
 - wlan.h, 378
- name
 - cli_command, 16
 - test_cfg_param_t, 27
 - test_cfg_table_t, 27
 - wifi_os_mem_info, 81
 - wlan_network, 123
- negotiation_type
 - wifi_twt_setup_config_t, 108
 - wifi_twt_tearardown_config_t, 110
- neighbor_ap
 - wlan_rrm_neighbor_report_t, 142
- neighbor_cnt
 - wlan_rrm_neighbor_report_t, 143
- neighbor_report_supported
 - wifi_scan_result2, 99
 - wlan_network, 128
 - wlan_scan_result, 149
- nentries
 - wififlt_cfg_t, 56
- net_accept
 - wm_net.h, 410
- net_address_types
 - wm_net.h, 411
- net_alloc_client_data_id
 - wm_net.h, 400
- net_bind
 - wm_net.h, 409
- net_close
 - wm_net.h, 410
- net_configure_address
 - wm_net.h, 402
- net_configure_dns
 - wm_net.h, 403
- net_connect
 - wm_net.h, 410
- net_dhcp_hostname_set
 - wm_net.h, 396

net_get_if_addr
 wm_net.h, 403
 net_get_if_ip_addr
 wm_net.h, 406
 net_get_if_ip_mask
 wm_net.h, 406
 net_get_if_ipv6_addr
 wm_net.h, 403
 net_get_if_ipv6_pref_addr
 wm_net.h, 404
 net_get_if_name
 wm_net.h, 405
 net_get_mlan_handle
 wm_net.h, 411
 net_get_sock_error
 wm_net.h, 397
 net_get_sta_handle
 wm_net.h, 400
 net_get_sta_interface
 wm_net.h, 399
 net_get_uap_handle
 wm_net.h, 400
 net_get_uap_interface
 wm_net.h, 400
 net_inet_aton
 wm_net.h, 397
 net_inet_ntoa
 wm_net.h, 398
 net_interface_dhcp_cleanup
 wm_net.h, 402
 net_interface_dhcp_stop
 wm_net.h, 402
 net_interface_down
 wm_net.h, 401
 net_interface_up
 wm_net.h, 401
 net_ip_config, 22
 ipv4, 23
 ipv6, 22
 ipv6_count, 22
 net_ipv4_config, 23
 addr_type, 23
 address, 23
 dns1, 24
 dns2, 24
 gw, 24
 netmask, 24
 net_ipv4stack_init
 wm_net.h, 408
 net_ipv6_config, 24
 addr_state, 25
 addr_type, 25
 address, 25
 net_listen
 wm_net.h, 410
 net_read
 wm_net.h, 410
 net_select
 wm_net.h, 409
 net_shutdown
 wm_net.h, 410
 net_sock_to_interface
 wm_net.h, 398
 net_socket
 wm_net.h, 409
 net_socket_blocking
 wm_net.h, 396
 net_stack_buffer_skip
 wm_net.h, 398
 net_stat
 wm_net.h, 408
 net_stop_dhcp_timer
 wm_net.h, 396
 net_wlan_deinit
 wm_net.h, 399
 net_wlan_init
 wm_net.h, 399
 net_wlan_set_mac_address
 wm_net.h, 397
 net_write
 wm_net.h, 411
 netmask
 ipv4_config, 21
 net_ipv4_config, 24
 nlist_mode
 wlan_nlist_report_param, 138
 no_of_chan
 wifi_sub_band_set_t, 102
 no_of_channels
 wifi_chan_list_param_set_t, 38
 noRsn
 _SecurityMode_t, 12
 nominal_wake
 wifi_btwt_config_t, 35
 none
 _Cipher_t, 9
 wlan_cipher, 117
 notes
 test_cfg_param_t, 27
 nss
 wifi_rate_cfg_t, 83
 num_byte_seq
 wifi_mef_filter_t, 62
 num_bytes
 wifi_mef_filter_t, 62
 num_channels
 wifi_scan_params_v2_t, 93
 wlan_nlist_report_param, 138
 num_chans
 wifi_chanlist_t, 40
 wifi_rtxpwrlimit_t, 89
 wifi_txpwrlimit_t, 113
 num_mask_seq
 wifi_mef_filter_t, 62
 num_mod_grps
 wifi_txpwrlimit_config_t, 112

- num_of_chan
 - wifi_scan_chan_list_t, 90
- num_probes
 - wifi_scan_params_v2_t, 94
- numDfsRecords
 - Event_Radar_Detected_Info, 19
- NumPkt
 - wifi_mfg_cmd_tx_frame_t, 77
- nusers
 - wlan_network_security, 136
- OPERAND_BYTE_SEQ
 - wifi-decl.h, 176
- OPERAND_DNUM
 - wifi-decl.h, 176
- OS_MEM_STAT_TABLE_SIZE
 - wifi-decl.h, 180
- OSA_DumpThreadInfo
 - osa.h, 169
- OSA_MUTEX_HANDLE_DEFINE
 - _rw_lock, 25, 26
- OSA_MsgQWaiting
 - osa.h, 170
- OSA_RWLockCreate
 - osa.h, 164
- OSA_RWLockCreateWithCB
 - osa.h, 164
- OSA_RWLockDestroy
 - osa.h, 165
- OSA_RWLockReadLock
 - osa.h, 166
- OSA_RWLockReadUnlock
 - osa.h, 167
- OSA_RWLockWriteLock
 - osa.h, 165
- OSA_RWLockWriteUnlock
 - osa.h, 166
- OSA_Rand
 - osa.h, 169
- OSA_RandRange
 - osa.h, 169
- OSA_RemoveIdleFunction
 - osa.h, 168
- OSA_RemoveTickFunction
 - osa.h, 168
- OSA_SEMAPHORE_HANDLE_DEFINE
 - _rw_lock, 26
 - csi_local_buff_statu, 17
- OSA_SetupIdleFunction
 - osa.h, 167
- OSA_SetupTickFunction
 - osa.h, 167
- OSA_Srand
 - osa.h, 169
- OSA_ThreadSelfComplete
 - osa.h, 170
- OSA_TimerActivate
 - osa.h, 161
- OSA_TimerChange
 - osa.h, 162
- osa.h, 162
- OSA_TimerCreate
 - osa.h, 161
- OSA_TimerDeactivate
 - osa.h, 163
- OSA_TimerDestroy
 - osa.h, 164
- OSA_TimerGetContext
 - osa.h, 163
- OSA_TimerIsRunning
 - osa.h, 162
- OSA_TimerReset
 - osa.h, 163
- offset
 - test_cfg_param_t, 27
 - wifi_mef_filter_t, 62
- op_class
 - wlan_rrm_neighbor_ap_t, 142
- osa.h, 161
 - cb_fn, 171
 - g_osa_idle_hooks, 171
 - g_osa_tick_hooks, 171
 - MAX_CUSTOM_HOOKS, 170
 - OSA_DumpThreadInfo, 169
 - OSA_MsgQWaiting, 170
 - OSA_RWLockCreate, 164
 - OSA_RWLockCreateWithCB, 164
 - OSA_RWLockDestroy, 165
 - OSA_RWLockReadLock, 166
 - OSA_RWLockReadUnlock, 167
 - OSA_RWLockWriteLock, 165
 - OSA_RWLockWriteUnlock, 166
 - OSA_Rand, 169
 - OSA_RandRange, 169
 - OSA_RemoveIdleFunction, 168
 - OSA_RemoveTickFunction, 168
 - OSA_SetupIdleFunction, 167
 - OSA_SetupTickFunction, 167
 - OSA_Srand, 169
 - OSA_ThreadSelfComplete, 170
 - OSA_TimerActivate, 161
 - OSA_TimerChange, 162
 - OSA_TimerCreate, 161
 - OSA_TimerDeactivate, 163
 - OSA_TimerDestroy, 164
 - OSA_TimerGetContext, 163
 - OSA_TimerIsRunning, 162
 - OSA_TimerReset, 163
 - wm_rand_seed, 171
- osa_rw_lock_t, 25
- owe
 - _SecurityMode_t, 13
- PAC_OPAQUE_ENCR_KEY_MAX_LENGTH
 - wlan.h, 370
- PACK_END
 - wm_utils.h, 418
- PACK_START
 - wm_utils.h, 418

PASSWORD_MAX_LENGTH
wlan.h, 370

PING_DEFAULT_COUNT
wifi_ping.h, 243

PING_DEFAULT_SIZE
wifi_ping.h, 243

PING_DEFAULT_TIMEOUT_SEC
wifi_ping.h, 243

PING_INTERVAL
wifi_ping.h, 242

PING_ID
wifi_ping.h, 242

PING_MAX_COUNT
wifi_ping.h, 243

PING_MAX_SIZE
wifi_ping.h, 243

PMK_BIN_LEN
wifi-decl.h, 172

PMK_HEX_LEN
wifi-decl.h, 172

PORT_PROTOCOL_OFFSET
wifi-decl.h, 178

pac_opaque_encr_key
wlan_network_security, 137

packet
wifi_cloud_keep_alive_t, 44

pairwise_cipher
wlan_network_security, 130

param
wifi_ds_rate, 51

param_list
test_cfg_table_t, 28

param_num
test_cfg_table_t, 28

passive_scan_or_radar_detect
wifi_chan_info_t, 37

password
wlan_network_security, 131

password_len
wlan_network_security, 131

passwords
wlan_network_security, 137

pattern
wifi_mef_filter_t, 62
wifi_wowlan_pattern_t, 115

pattern_len
wifi_wowlan_pattern_t, 115

patterns
wifi_wowlan_ptn_cfg_t, 116

payload
wifi_mgmt_frame_t, 79

payload_pattern
wifi_mfg_cmd_tx_cont_t, 72

peap_label
wlan_network_security, 132

phecap_ie_present
wifi_scan_result2, 98

phtcap_ie_present
wifi_scan_result2, 97

phtinfo_ie_present
wifi_scan_result2, 97

phy_type
wlan_rrm_neighbor_ap_t, 142

ping_cli_deinit
wifi_ping.h, 242

ping_cli_init
wifi_ping.h, 241

ping_e
wifi_ping.h, 242

ping_stats
wifi_ping.h, 241

ping_w
wifi_ping.h, 242

pkc
wlan_network_security, 130

pkt_len
wifi_cloud_keep_alive_t, 44

pkt_offset
wifi_wowlan_pattern_t, 115

pkt_type
wifi_csi_filter_t, 47

pktLength
wifi_cw_mode_ctrl_t, 49

pmk
wlan_network_security, 131

pmk_valid
wlan_network_security, 132

power_mgmt_status
wifi_sta_info_t, 100

preamble
txrate_setting, 29

pri_binCnt
Event_Radar_Detected_Info, 19

pri_radar_type
Event_Radar_Detected_Info, 19

print_ascii
wm_utils.h, 419

print_mac
wlan.h, 312

print_txpwrlimit
wlan_tests.h, 395

protect
wlan_nlist_report_param, 139
wlan_rrm_scan_cb_param, 143

ps_mode
wlan_ieeeeps_config, 120

ps_null_interval
wlan_ieeeeps_config, 119

psk
wlan_network_security, 130

psk_len
wlan_network_security, 131

pvhtcap_ie_present
wifi_scan_result2, 98

pw_chirp_idx
Event_Radar_Detected_Info, 19

- pw_chirp_type
 - Event_Radar_Detected_Info, 19
- pw_value
 - Event_Radar_Detected_Info, 19
- pwe_derivation
 - wlan_network_security, 131
- QNum
 - wifi_mfg_cmd_tx_frame_t, 78
- qnum
 - wifi_mfg_cmd_he_tb_tx_t, 66
- README.txt, 171
- RPN_TYPE_AND
 - wifi-decl.h, 177
- RPN_TYPE_OR
 - wifi-decl.h, 177
- RSSI
 - wifi_scan_result2, 96
- ra4us
 - wifi_csi_config_params_t, 46
- radio_type
 - wifi_scan_channel_list_t, 91
- random_hdlr_t
 - wm_utils.h, 420
- random_initialize_seed
 - wm_utils.h, 414
- random_register_handler
 - wm_utils.h, 413
- random_register_seed_handler
 - wm_utils.h, 413
- random_unregister_handler
 - wm_utils.h, 413
- random_unregister_seed_handler
 - wm_utils.h, 414
- rate
 - wifi_rate_cfg_t, 83
- rate_cfg
 - wifi_ds_rate, 51
- rate_format
 - wifi_rate_cfg_t, 83
- rate_index
 - wifi_rate_cfg_t, 83
- rate_setting
 - wifi_rate_cfg_t, 83
- rateInfo
 - wifi_cw_mode_ctrl_t, 49
- read_index
 - csi_local_buff_statu, 17
- reader_cb
 - _rw_lock, 26
- reader_count
 - _rw_lock, 26
- reallyPassed
 - Event_Radar_Detected_Info, 20
- reason
 - wifi_message, 63
- reason_code
 - wifi_uap_client_disassoc_t, 114
- reconnect_counter
 - wifi_auto_reconnect_config_t, 33
- reconnect_interval
 - wifi_auto_reconnect_config_t, 33
- reg_domain
 - Event_Radar_Detected_Info, 18
- region_string_2_region_code
 - wifi.h, 230
- register_csi_user_callback
 - wifi.h, 228
- remain_period
 - wifi_remain_on_channel_t, 84
- remove
 - wifi_remain_on_channel_t, 84
- rep_data
 - wlan_rrm_scan_cb_param, 143
- repeat
 - wifi_mef_filter_t, 62
- report_detail
 - wlan_rrm_beacon_report_data, 141
- reserve
 - wifi_twt_report_t, 107
- reserved_1
 - wifi_ext_coex_config_t, 55
- reserved_2
 - wifi_ext_coex_config_t, 55
- reserverd
 - txrate_setting, 30
- reset
 - wifi_cloud_keep_alive_t, 43
 - wifi_tcp_keep_alive_t, 103
- reset_ie_index
 - wifi.h, 184
- result
 - wnm_sleep_result_t, 150
- retry_count
 - wifi_cloud_keep_alive_t, 43
- retry_interval
 - wifi_cloud_keep_alive_t, 43
- rf_type
 - wifi_rf_channel_t, 85
- role
 - wlan_network, 125
 - wlan_scan_result, 145
- rpn
 - wifi_mef_entry_t, 61
- rsn_mcstCipher
 - wifi_scan_result2, 96
- rsn_ucstCipher
 - wifi_scan_result2, 97
- rsi
 - wifi_sta_info_t, 100
 - wlan_network, 124
 - wlan_scan_result, 148
- rsvd
 - _Cipher_t, 11
 - _SecurityMode_t, 14
 - wifi_mfg_cmd_tx_cont_t, 73

- wifi_mfg_cmd_tx_frame_t, 77
- wlan_cipher, 118
- rsvd2
 - _Cipher_t, 11
 - wlan_cipher, 119
- ruPower
 - wifi_rupwrlimit_config_t, 88
- rupwrlimit_config
 - wifi_rutxpwrlimit_t, 89
- rx_bw
 - wifi_data_rate_t, 50
- rx_data_rate
 - wifi_data_rate_t, 49
- rx_gi
 - wifi_data_rate_t, 50
- sae_groups
 - wlan_network_security, 131
- sample_initialise_random_seed
 - wm_utils.h, 415
- scan_chan_gap
 - wifi_scan_params_v2_t, 94
- scan_duration
 - wifi_scan_params_t, 92
- scan_only
 - wifi_scan_params_v2_t, 93
- scan_time
 - wifi_scan_channel_list_t, 91
- scan_type
 - wifi_scan_channel_list_t, 91
- scanMode
 - BandConfig_t, 15
- ScanMode_e
 - wlan.h, 391
- sec_channel_offset
 - wlan_network, 124
- security
 - wlan_network, 125
- security_specific
 - wlan_network, 126
- send_interval
 - wifi_cloud_keep_alive_t, 43
- send_sleep_confirm_command
 - wifi.h, 207
- seq_ctl
 - wifi_mgmt_frame_t, 79
- seq_no
 - wifi_tcp_keep_alive_t, 104
- server_cert_data
 - wlan_network_security, 136
- server_cert_len
 - wlan_network_security, 136
- server_key_data
 - wlan_network_security, 136
- server_key_len
 - wlan_network_security, 136
- server_key_passwd
 - wlan_network_security, 136
- set_event_chanswann
 - wifi.h, 205
- set_scan_params
 - wlan.h, 245
- short_gi
 - wifi_mfg_cmd_tx_frame_t, 76
- short_preamble
 - wifi_mfg_cmd_tx_frame_t, 76
- shortGI
 - txrate_setting, 29
- signal_bw
 - wifi_mfg_cmd_tx_frame_t, 77
- size
 - wifi_os_mem_info, 81
- sms4
 - _Cipher_t, 10
 - wlan_cipher, 118
- soft_crc32
 - wm_utils.h, 416
- sp_gap
 - wifi_btwt_config_t, 36
- split_scan_delay
 - wifi_scan_params_t, 92
- src_addr
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
- src_ip
 - wifi_cloud_keep_alive_t, 44
- src_mac
 - wifi_cloud_keep_alive_t, 44
- src_port
 - wifi_cloud_keep_alive_t, 44
- src_tcp_port
 - wifi_tcp_keep_alive_t, 104
- ssid
 - wifi_scan_params_t, 92
 - wifi_scan_params_v2_t, 93
 - wifi_scan_result2, 95
 - wlan_network, 123
 - wlan_rrm_beacon_report_data, 140
 - wlan_rrm_neighbor_ap_t, 141
 - wlan_scan_result, 145
- ssid_len
 - wifi_scan_result2, 95
 - wlan_scan_result, 145
- ssid_length
 - wlan_rrm_beacon_report_data, 140
- ssid_specific
 - wlan_network, 125
- sta_addr
 - wifi_uap_client_disassoc_t, 114
- standalone_hetb
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
- start_freq
 - wifi_channel_desc_t, 41
 - wifi_rupwrlimit_config_t, 88
- status
 - wifi_remain_on_channel_t, 84
- stbc
 - txrate_setting, 30

- wifi_mfg_cmd_tx_frame_t, 76
- strdup
 - wm_utils.h, 415
- sub_command
 - wifi_ds_rate, 51
- sub_id
 - wifi_btwt_config_t, 35
- subband
 - wifi_txpwrlimit_t, 113
- subtype
 - wifi_csi_filter_t, 47
- suspend_duration
 - wifi_twt_information_t, 106
- TCP_OF_IP_PROTOCOL
 - wifi-decl.h, 177
- TX_AMPDU_CTS_2_SELF
 - wlan.h, 379
- TX_AMPDU_DISABLE_PROTECTION
 - wlan.h, 379
- TX_AMPDU_DYNAMIC_RTS_CTS
 - wlan.h, 379
- TX_AMPDU_RTS_CTS
 - wlan.h, 379
- TYPE_BIT_EQ
 - wifi-decl.h, 177
- TYPE_BYTE_EQ
 - wifi-decl.h, 177
- TYPE_DNUM_EQ
 - wifi-decl.h, 177
- tail_id
 - wifi_csi_config_params_t, 46
- teardown_all_twt
 - wifi_twt_teardown_config_t, 110
- test_cfg_param_t, 26
 - len, 27
 - name, 27
 - notes, 27
 - offset, 27
- test_cfg_table_t, 27
 - data, 28
 - len, 28
 - name, 27
 - param_list, 28
 - param_num, 28
- test_wlan_cfg_process
 - wlan_tests.h, 395
- timeout
 - wifi_tcp_keep_alive_t, 104
- tkip
 - _Cipher_t, 10
 - wlan_cipher, 117
- tls_cipher
 - wlan_network_security, 133
- token
 - wlan_rrm_beacon_report_data, 140
- trans_bssid
 - wifi_scan_result2, 99
 - wlan_scan_result, 148
- trans_mode
 - wifi_scan_result2, 98
- trans_ssid
 - wifi_scan_result2, 99
 - wlan_scan_result, 148
- trans_ssid_len
 - wifi_scan_result2, 99
 - wlan_scan_result, 148
- transition_disable
 - wlan_network_security, 131
- trig_common_field
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 68
- trig_user_info_field
 - wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 69
- trigger_enabled
 - wifi_twt_setup_config_t, 108
- tsf
 - wifi_tsf_info_t, 105
- tsf_format
 - wifi_tsf_info_t, 105
- tsf_info
 - wifi_tsf_info_t, 105
- tsf_offset
 - wifi_tsf_info_t, 105
- twt_exponent
 - wifi_btwt_config_t, 36
 - wifi_twt_setup_config_t, 109
- twt_info_disabled
 - wifi_twt_setup_config_t, 108
- twt_mantissa
 - wifi_btwt_config_t, 35
 - wifi_twt_setup_config_t, 109
- twt_offset
 - wifi_btwt_config_t, 36
- twt_request
 - wifi_twt_setup_config_t, 109
- twt_wakeup_duration
 - wifi_twt_setup_config_t, 109
- tx_ampdu_prot_mode_para, 28
 - mode, 29
- tx_bf
 - wifi_mfg_cmd_tx_frame_t, 76
- tx_bw
 - wifi_data_rate_t, 50
- tx_data_rate
 - wifi_data_rate_t, 49
- tx_gi
 - wifi_data_rate_t, 50
- tx_power
 - wifi_mfg_cmd_he_tb_tx_t, 66
 - wifi_txpwrlimit_entry_t, 113
- tx_rate
 - wifi_mfg_cmd_tx_cont_t, 73
- txPower
 - wifi_cw_mode_ctrl_t, 48
- txpwrlimit_config
 - wifi_txpwrlimit_t, 114
- txpwrlimit_entry

- wifi_txpwrlimit_config_t, 112
- txrate_setting, 29
 - adv_coding, 30
 - bandwidth, 29
 - dcm, 30
 - doppler, 30
 - max_pkttext, 30
 - preamble, 29
 - reserverd, 30
 - shortGI, 29
 - stbc, 30
- type
 - wifi_mef_filter_t, 62
 - wifi_twt_report_t, 107
 - wlan_network, 125
 - wlan_network_security, 129
 - wlan_scan_result, 145
- UAP_DEFAULT_BANDWIDTH
 - wifi-decl.h, 173
- UAP_DEFAULT_BEACON_PERIOD
 - wifi-decl.h, 173
- UAP_DEFAULT_CHANNEL
 - wifi-decl.h, 173
- UAP_DEFAULT_HIDDEN_SSID
 - wifi-decl.h, 173
- UDP_OF_IP_PROTOCOL
 - wifi-decl.h, 177
- ucstCipher
 - wlan_network_security, 130
- unregister_csi_user_callback
 - wifi.h, 228
- val
 - wifi_11ax_config_t, 32
- valid_data_cnt
 - csi_local_buff_statu, 18
- verify_scan_channel_value
 - wlan.h, 244
- verify_scan_duration_value
 - wlan.h, 244
- verify_sequential_pattern
 - wm_utils.h, 417
- verify_split_scan_delay
 - wlan.h, 244
- version_str
 - wifi_fw_version_ext_t, 58
 - wifi_fw_version_t, 58
- version_str_sel
 - wifi_fw_version_ext_t, 58
- vht_capab
 - wlan_network, 124
- vht_oper_chwidth
 - wlan_network, 124
- WARN_UNUSED_RET
 - wm_utils.h, 418
- WIFI_COMMAND_RESPONSE_WAIT_MS
 - wifi.h, 233
- WIFI_MAX_CHANNEL_NUM
 - wifi-decl.h, 172
- WIFI_MGMT_ACTION
 - wifi.h, 234
- WIFI_MGMT_AUTH
 - wifi.h, 234
- WIFI_MGMT_DEAUTH
 - wifi.h, 234
- WIFI_MGMT_DIASOC
 - wifi.h, 234
- WIFI_REG16
 - wifi.h, 232
- WIFI_REG32
 - wifi.h, 232
- WIFI_REG8
 - wifi.h, 232
- WIFI_SUPPORT_11AC
 - wifi-decl.h, 172
- WIFI_SUPPORT_11AX
 - wifi-decl.h, 172
- WIFI_SUPPORT_11N
 - wifi-decl.h, 172
- WIFI_SUPPORT_LEGACY
 - wifi-decl.h, 173
- WIFI_WRITE_REG16
 - wifi.h, 233
- WIFI_WRITE_REG32
 - wifi.h, 233
- WIFI_WRITE_REG8
 - wifi.h, 233
- WLAN_11D_SCAN_LIMIT
 - wlan.h, 368
- WLAN_BTWT_REPORT_LEN
 - wifi-decl.h, 179
- WLAN_BTWT_REPORT_MAX_NUM
 - wifi-decl.h, 179
- WLAN_CIPHER_AES_128_CMAC
 - wlan.h, 377
- WLAN_CIPHER_BIP_CMAC_256
 - wlan.h, 378
- WLAN_CIPHER_BIP_GMAC_128
 - wlan.h, 378
- WLAN_CIPHER_BIP_GMAC_256
 - wlan.h, 378
- WLAN_CIPHER_CCMP_256
 - wlan.h, 377
- WLAN_CIPHER_CCMP
 - wlan.h, 377
- WLAN_CIPHER_GCMP_256
 - wlan.h, 377
- WLAN_CIPHER_GCMP
 - wlan.h, 377
- WLAN_CIPHER_GTK_NOT_USED
 - wlan.h, 378
- WLAN_CIPHER_NONE
 - wlan.h, 376
- WLAN_CIPHER_SMS4
 - wlan.h, 377

- WLAN_CIPHER_TKIP
wlan.h, [377](#)
- WLAN_CIPHER_WEP104
wlan.h, [377](#)
- WLAN_CIPHER_WEP40
wlan.h, [376](#)
- WLAN_DRV_VERSION
wlan.h, [367](#)
- WLAN_ERROR_ACTION
wlan.h, [372](#)
- WLAN_ERROR_NOMEM
wlan.h, [371](#)
- WLAN_ERROR_NONE
wlan.h, [371](#)
- WLAN_ERROR_NOT_SUPPORTED
wlan.h, [372](#)
- WLAN_ERROR_PARAM
wlan.h, [371](#)
- WLAN_ERROR_PS_ACTION
wlan.h, [372](#)
- WLAN_ERROR_STATE
wlan.h, [371](#)
- WLAN_GRANT_op_gpio_num
wifi_ext_coex_config_t, [54](#)
- WLAN_GRANT_op_gpio_polarity
wifi_ext_coex_config_t, [55](#)
- WLAN_KEY_MGMT_CCKM
wlan.h, [374](#)
- WLAN_KEY_MGMT_DPP
wlan.h, [376](#)
- WLAN_KEY_MGMT_FILS_SHA256
wlan.h, [375](#)
- WLAN_KEY_MGMT_FILS_SHA384
wlan.h, [375](#)
- WLAN_KEY_MGMT_FT_FILS_SHA256
wlan.h, [375](#)
- WLAN_KEY_MGMT_FT_FILS_SHA384
wlan.h, [375](#)
- WLAN_KEY_MGMT_FT_IEEE8021X_SHA384
wlan.h, [376](#)
- WLAN_KEY_MGMT_FT_IEEE8021X
wlan.h, [373](#)
- WLAN_KEY_MGMT_FT_PSK
wlan.h, [373](#)
- WLAN_KEY_MGMT_FT_SAE
wlan.h, [374](#)
- WLAN_KEY_MGMT_FT
wlan.h, [376](#)
- WLAN_KEY_MGMT_IEEE8021X_NO_WPA
wlan.h, [373](#)
- WLAN_KEY_MGMT_IEEE8021X_SHA256
wlan.h, [374](#)
- WLAN_KEY_MGMT_IEEE8021X_SUITE_B_192
wlan.h, [375](#)
- WLAN_KEY_MGMT_IEEE8021X_SUITE_B
wlan.h, [375](#)
- WLAN_KEY_MGMT_IEEE8021X
wlan.h, [373](#)
- WLAN_KEY_MGMT_NONE
wlan.h, [373](#)
- WLAN_KEY_MGMT_OSEN
wlan.h, [375](#)
- WLAN_KEY_MGMT_OWE
wlan.h, [375](#)
- WLAN_KEY_MGMT_PASN
wlan.h, [376](#)
- WLAN_KEY_MGMT_PSK_SHA256
wlan.h, [374](#)
- WLAN_KEY_MGMT_PSK
wlan.h, [373](#)
- WLAN_KEY_MGMT_SAE_EXT_KEY
wlan.h, [376](#)
- WLAN_KEY_MGMT_SAE
wlan.h, [374](#)
- WLAN_KEY_MGMT_WAPI_CERT
wlan.h, [374](#)
- WLAN_KEY_MGMT_WAPI_PSK
wlan.h, [374](#)
- WLAN_KEY_MGMT_WPA_NONE
wlan.h, [373](#)
- WLAN_KEY_MGMT_WPS
wlan.h, [374](#)
- WLAN_MAC_ADDR_LENGTH
wlan.h, [371](#)
- WLAN_MAX_KNOWN_NETWORKS
wlan.h, [371](#)
- WLAN_MAX_STA_FILTER_NUM
wlan.h, [371](#)
- WLAN_MGMT_ACTION
wlan.h, [373](#)
- WLAN_MGMT_AUTH
wlan.h, [372](#)
- WLAN_MGMT_DEAUTH
wlan.h, [372](#)
- WLAN_MGMT_DIASOC
wlan.h, [372](#)
- WLAN_NETWORK_NAME_MAX_LENGTH
wlan.h, [369](#)
- WLAN_NETWORK_NAME_MIN_LENGTH
wlan.h, [369](#)
- WLAN_PASSWORD_MAX_LENGTH
wlan.h, [369](#)
- WLAN_PASSWORD_MIN_LENGTH
wlan.h, [369](#)
- WLAN_PMK_LENGTH
wlan.h, [371](#)
- WLAN_PSK_MAX_LENGTH
wlan.h, [369](#)
- WLAN_PSK_MIN_LENGTH
wlan.h, [369](#)
- WLAN_REASON_CODE_PREV_AUTH_NOT_VALID
wlan.h, [368](#)
- WLAN_RECONNECT_LIMIT
wlan.h, [369](#)
- WLAN_RESCAN_LIMIT
wlan.h, [368](#)

- WM_MASK
 - wm_utils.h, 418
- WOWLAN_MAX_OFFSET_LEN
 - wifi-decl.h, 174
- WOWLAN_MAX_PATTERN_LEN
 - wifi-decl.h, 174
- WPA_WPA2_WEP
 - wifi_scan_result2, 96
- wep
 - wlan_scan_result, 146
- wep104
 - _Cipher_t, 10
 - wlan_cipher, 117
- wep40
 - _Cipher_t, 9
 - wlan_cipher, 117
- wepDynamic
 - _SecurityMode_t, 12
- wepStatic
 - _SecurityMode_t, 12
- width
 - wifi_rupwrlimit_config_t, 88
- wifi-decl.h, 171
 - BAND_SPECIFIED, 179
 - BIT, 174
 - BSS_TYPE_STA, 173
 - BSS_TYPE_UAP, 173
 - CRITERIA_BROADCAST, 175
 - CRITERIA_MULTICAST, 176
 - CRITERIA_UNICAST, 176
 - CSI_FILTER_MAX, 180
 - FILLING_BYTE_SEQ, 179
 - FILLING_MASK_SEQ, 179
 - FILLING_NUM_BYTES, 178
 - FILLING_OFFSET, 178
 - FILLING_PATTERN, 178
 - FILLING_REPEAT, 178
 - FILLING_TYPE, 178
 - ICMP_OF_IP_PROTOCOL, 177
 - IP_PROTOCOL_OFFSET, 178
 - IPV4_PKT_OFFSET, 178
 - MAX_CHANNEL_LIST, 179
 - MAX_FUNC_SYMBOL_LEN, 180
 - MAX_NUM_BYTE_SEQ, 176
 - MAX_NUM_ENTRIES, 176
 - MAX_NUM_FILTERS, 174
 - MAX_NUM_MASK_SEQ, 176
 - MAX_NUM_SSID, 179
 - MAX_OPERAND, 176
 - MEF_ACTION_ALLOW_AND_WAKEUP_HOST, 175
 - MEF_ACTION_ALLOW, 175
 - MEF_ACTION_WAKE, 175
 - MEF_AUTO_ARP, 175
 - MEF_AUTO_PING, 175
 - MEF_MAGIC_PKT, 175
 - MEF_MODE_HOST_SLEEP, 174
 - MEF_MODE_NON_HOST_SLEEP, 174
 - MEF_NS_RESP, 175
 - MKEEP_ALIVE_IP_PKT_MAX, 179
 - MLAN_MAC_ADDR_LENGTH, 171
 - MLAN_MAX_PASS_LENGTH, 174
 - MLAN_MAX_SSID_LENGTH, 173
 - MLAN_MAX_VER_STR_LEN, 172
 - MOD_GROUPS, 172
 - OPERAND_BYTE_SEQ, 176
 - OPERAND_DNUM, 176
 - OS_MEM_STAT_TABLE_SIZE, 180
 - PMK_BIN_LEN, 172
 - PMK_HEX_LEN, 172
 - PORT_PROTOCOL_OFFSET, 178
 - RPN_TYPE_AND, 177
 - RPN_TYPE_OR, 177
 - TCP_OF_IP_PROTOCOL, 177
 - TYPE_BIT_EQ, 177
 - TYPE_BYTE_EQ, 177
 - TYPE_DNUM_EQ, 177
 - UAP_DEFAULT_BANDWIDTH, 173
 - UAP_DEFAULT_BEACON_PERIOD, 173
 - UAP_DEFAULT_CHANNEL, 173
 - UAP_DEFAULT_HIDDEN_SSID, 173
 - UDP_OF_IP_PROTOCOL, 177
 - WIFI_MAX_CHANNEL_NUM, 172
 - WIFI_SUPPORT_11AC, 172
 - WIFI_SUPPORT_11AX, 172
 - WIFI_SUPPORT_11N, 172
 - WIFI_SUPPORT_LEGACY, 173
 - WLAN_BTWT_REPORT_LEN, 179
 - WLAN_BTWT_REPORT_MAX_NUM, 179
 - WOWLAN_MAX_OFFSET_LEN, 174
 - WOWLAN_MAX_PATTERN_LEN, 174
 - wifi_SubBand_t, 181
 - wifi_bss_features, 180
 - wifi_bss_security, 180
 - wifi_ds_command_type, 181
 - wifi_frame_type_t, 182
 - wlan_type, 181
- wifi.h, 182
 - _wifi_set_mac_addr, 196
 - BANDWIDTH_20MHZ, 233
 - BANDWIDTH_40MHZ, 233
 - BANDWIDTH_80MHZ, 233
 - BEACON_REPORT_BUF_SIZE, 234
 - CONFIG_FW_VDLL, 232
 - CONFIG_GTK_REKEY_OFFLOAD, 232
 - CONFIG_STA_AUTO_DHCPV4, 231
 - CONFIG_TCP_ACK_ENH, 232
 - CONFIG_WIFI_AUTO_POWER_SAVE, 232
 - CONFIG_WIFI_STA_RECONNECT, 231
 - clear_event_chanswann, 205
 - csi_deliver_data_to_user, 228
 - csi_event_cnt, 238
 - csi_event_data_len, 238
 - csi_local_buff_init, 228
 - csi_save_data_to_local_buff, 228
 - g_bcn_nf_last, 237

g_data_nf_last, 237
 g_data_snr_last, 237
 g_rssi, 237
 get_sub_band_from_region_code, 203
 get_sub_band_from_region_code_5ghz, 203
 hostapd_connected_sta_list, 231
 IEEEtypes_ElementId_t, 235
 MAX_NEIGHBOR_AP_LIMIT, 235
 MAX_NUM_CHANS_IN_NBOR_RPT, 234
 MBIT, 234
 region_string_2_region_code, 230
 register_csi_user_callback, 228
 reset_ie_index, 184
 send_sleep_confirm_command, 207
 set_event_chanswann, 205
 unregister_csi_user_callback, 228
 WIFI_COMMAND_RESPONSE_WAIT_MS, 233
 WIFI_MGMT_ACTION, 234
 WIFI_MGMT_AUTH, 234
 WIFI_MGMT_DEAUTH, 234
 WIFI_MGMT_DIASSOC, 234
 WIFI_REG16, 232
 WIFI_REG32, 232
 WIFI_REG8, 232
 WIFI_WRITE_REG16, 233
 WIFI_WRITE_REG32, 233
 WIFI_WRITE_REG8, 233
 wifi_11d_is_channel_allowed, 202
 wifi_11h_enable, 210
 wifi_add_mcast_filter, 196
 wifi_add_to_bypassq, 186
 wifi_cfg_rf_he_tb_tx, 222
 wifi_clear_mgmt_ie, 214
 wifi_cloud_keep_alive, 214
 wifi_config_bgscan_and_rssi, 192
 wifi_config_roaming, 192
 wifi_configure_delay_to_ps, 208
 wifi_configure_listen_interval, 208
 wifi_configure_null_pkt_interval, 208
 wifi_csi_cfg, 227
 wifi_deauthenticate, 229
 wifi_deinit, 183
 wifi_deregister_amsdu_data_input_callback, 185
 wifi_deregister_data_input_callback, 184
 wifi_deregister_deliver_packet_above_callback, 185
 wifi_deregister_wrapper_net_is_ip_or_ipv6_callback, 186
 wifi_disable_11d_support, 203
 wifi_disable_uap_11d_support, 203
 wifi_enable_11d_support, 203
 wifi_enable_low_pwr_mode, 195
 wifi_enable_uap_11d_support, 203
 wifi_enter_deepsleep_power_save, 207
 wifi_enter_ieee_power_save, 206
 wifi_enter_wnm_power_save, 207
 wifi_exit_deepsleep_power_save, 207
 wifi_exit_ieee_power_save, 206
 wifi_exit_wnm_power_save, 207
 wifi_get_11ax_rtxpowerlimit_legacy, 215
 wifi_get_antenna, 198
 wifi_get_country_code, 201
 wifi_get_data_rate, 212
 wifi_get_default_ht_capab, 229
 wifi_get_default_vht_capab, 230
 wifi_get_delay_to_ps, 208
 wifi_get_device_firmware_version_ext, 191
 wifi_get_device_mac_addr, 190
 wifi_get_device_uap_mac_addr, 191
 wifi_get_fw_info, 212
 wifi_get_indrst_cfg, 230
 wifi_get_ipv4_multicast_mac, 197
 wifi_get_ipv6_multicast_mac, 198
 wifi_get_last_cmd_sent_ms, 192
 wifi_get_listen_interval, 208
 wifi_get_outbuf, 192
 wifi_get_region_code, 200
 wifi_get_rf_band, 220
 wifi_get_rf_bandwidth, 220
 wifi_get_rf_channel, 220
 wifi_get_rf_otp_cal_data, 223
 wifi_get_rf_otp_mac_addr, 223
 wifi_get_rf_per, 220
 wifi_get_rf_radio_mode, 220
 wifi_get_rf_rx_antenna, 221
 wifi_get_rf_tx_antenna, 221
 wifi_get_scan_result, 193
 wifi_get_scan_result_count, 194
 wifi_get_sec_channel_offset, 226
 wifi_get_tsf_info, 219
 wifi_get_turbo_mode, 229
 wifi_get_twt_report, 217
 wifi_get_tx_power, 204
 wifi_get_txratecfg, 209
 wifi_get_uap_channel, 201
 wifi_get_uap_turbo_mode, 229
 wifi_get_value1, 192
 wifi_get_wakeup_reason, 207
 wifi_get_wpa_ie_in_assoc, 196
 wifi_get_xfer_pending, 209
 wifi_handle_event_data_pause, 224
 wifi_host_11k_cfg, 213
 wifi_host_11k_neighbor_req, 213
 wifi_host_11v_bss_trans_query, 213
 wifi_host_mbo_cfg, 227
 wifi_init, 182
 wifi_init_fcc, 183
 wifi_inject_frame, 225
 wifi_is_remain_on_channel, 231
 wifi_is_wpa_supplicant_input, 225
 wifi_low_level_output, 186
 wifi_mbo_preferch_cfg, 227
 wifi_mbo_send_preferch_wnm, 227
 wifi_mem_access, 199
 wifi_nxp_reset_scan_flag, 226
 wifi_nxp_scan_res_get, 226

wifi_nxp_set_default_scan_ies, 226
wifi_nxp_survey_res_get, 226
wifi_process_hs_cfg_resp, 198
wifi_process_ps_enh_response, 199
wifi_raw_packet_rcv, 215
wifi_raw_packet_send, 214
wifi_reg_t, 236
wifi_register_amsdu_data_input_callback, 185
wifi_register_data_input_callback, 184
wifi_register_deliver_packet_above_callback, 185
wifi_register_event_queue, 193
wifi_register_fw_dump_cb, 223
wifi_register_wrapper_net_is_ip_or_ipv6_callback, 185
wifi_remove_mcast_filter, 197
wifi_rf_trigger_frame_cfg, 222
wifi_rx_block_cnt, 237
wifi_rx_status, 237
wifi_same_ess_ft, 213
wifi_scan_process_results, 199
wifi_send_hs_cfg_cmd, 205
wifi_send_mgmt_auth_request, 228
wifi_send_scan_cmd, 228
wifi_set_11ax_cfg, 216
wifi_set_11ax_rtxpowerlimit, 215
wifi_set_11ax_rtxpowerlimit_legacy, 215
wifi_set_11ax_tol_time, 215
wifi_set_11ax_tx_omi, 215
wifi_set_antenna, 198
wifi_set_auto_arp, 214
wifi_set_btwt_cfg, 216
wifi_set_cal_data, 195
wifi_set_clocksync_cfg, 219
wifi_set_country_code, 201
wifi_set_country_ie_ignore, 201
wifi_set_frag, 213
wifi_set_htcapinfo, 204
wifi_set_httxcf, 204
wifi_set_indrst_cfg, 230
wifi_set_mac_addr, 196
wifi_set_packet_filters, 211
wifi_set_packet_retry_count, 186
wifi_set_power_save_mode, 207
wifi_set_ps_cfg, 205
wifi_set_region_code, 200
wifi_set_region_power_cfg, 203
wifi_set_rf_band, 220
wifi_set_rf_bandwidth, 220
wifi_set_rf_channel, 219
wifi_set_rf_otp_cal_data, 223
wifi_set_rf_otp_mac_addr, 223
wifi_set_rf_radio_mode, 219
wifi_set_rf_rx_antenna, 221
wifi_set_rf_test_mode, 219
wifi_set_rf_tx_antenna, 221
wifi_set_rf_tx_cont_mode, 221
wifi_set_rf_tx_frame, 222
wifi_set_rf_tx_power, 221
wifi_set_rssi_low_threshold, 224
wifi_set_rts, 213
wifi_set_rx_status, 184
wifi_set_sta_mac_filter, 214
wifi_set_turbo_mode, 229
wifi_set_twt_setup_cfg, 216
wifi_set_twt_teardown_cfg, 217
wifi_set_tx_power, 204
wifi_set_tx_status, 183
wifi_set_txbfcap, 204
wifi_set_txratecfg, 209
wifi_set_uap_turbo_mode, 229
wifi_set_xfer_pending, 209
wifi_show_os_mem_stat, 225
wifi_shutdown_enable, 238
wifi_sta_ampdu_rx_disable, 190
wifi_sta_ampdu_rx_enable, 188
wifi_sta_ampdu_rx_enable_per_tid, 188
wifi_sta_ampdu_rx_enable_per_tid_is_allowed, 188
wifi_sta_ampdu_tx_disable, 187
wifi_sta_ampdu_tx_enable, 187
wifi_sta_ampdu_tx_enable_per_tid, 187
wifi_sta_ampdu_tx_enable_per_tid_is_allowed, 187
wifi_sta_deauth, 195
wifi_stop_bgscan, 192
wifi_supp_inject_frame, 225
wifi_tcp_keep_alive, 214
wifi_test_independent_reset, 231
wifi_trigger_oob_inrst, 231
wifi_twt_information, 217
wifi_tx_block_cnt, 237
wifi_tx_card_awake_lock, 209
wifi_tx_card_awake_unlock, 210
wifi_tx_status, 237
wifi_uap_ampdu_rx_disable, 189
wifi_uap_ampdu_rx_enable, 189
wifi_uap_ampdu_rx_enable_per_tid, 189
wifi_uap_ampdu_rx_enable_per_tid_is_allowed, 189
wifi_uap_ampdu_tx_disable, 190
wifi_uap_ampdu_tx_enable, 189
wifi_uap_ampdu_tx_enable_per_tid, 190
wifi_uap_ampdu_tx_enable_per_tid_is_allowed, 190
wifi_uap_bss_sta_list, 194
wifi_uap_client_assoc, 230
wifi_uap_client_deauth, 230
wifi_uap_config_wifi_capa, 212
wifi_uap_do_acs, 212
wifi_uap_enable_11d_support, 202
wifi_uap_enable_sticky_bit, 208
wifi_uap_get_bandwidth, 212
wifi_uap_get_pmfcfg, 213
wifi_uap_pmf_getset, 202
wifi_uap_ps_inactivity_sleep_enter, 206
wifi_uap_ps_inactivity_sleep_exit, 206

- wifi_uap_ps_sta_ageout_timer_getset, 199
- wifi_uap_rates_getset, 199
- wifi_uap_set_bandwidth, 212
- wifi_uap_set_httxcf, 204
- wifi_uap_set_httxcf_int, 204
- wifi_uap_sta_ageout_timer_getset, 199
- wifi_uap_start, 211
- wifi_uap_stop, 211
- wifi_unregister_event_queue, 193
- wifi_unset_rf_test_mode, 219
- wifi_update_last_cmd_sent_ms, 192
- wifi_wake_up_card, 209
- wifi_wmm_get_packet_cnt, 224
- wifi_wmm_get_pkt_prio, 224
- wifi_wmm_init, 224
- wifi_wmm_tx_stats_dump, 224
- wifi_wpa_supplicant_eapol_input, 226
- wlan_nlist_mode, 236
- wlan_rrm_beacon_reporting_detail, 236
- wrapper_clear_media_connected_event, 206
- wrapper_wifi_assoc, 208
- wrapper_wlan_11d_clear_parsedtable, 206
- wrapper_wlan_11d_enable, 210
- wrapper_wlan_11d_support_is_enabled, 205
- wrapper_wlan_cmd_11n_addba_rspgen, 210
- wrapper_wlan_cmd_11n_ba_stream_timeout, 209
- wrapper_wlan_cmd_11n_delba_rspgen, 210
- wrapper_wlan_cmd_get_hw_spec, 205
- wrapper_wlan_ecsa_enable, 210
- wrapper_wlan_sta_ampdu_enable, 211
- wrapper_wlan_uap_11d_enable, 210
- wrapper_wlan_uap_ampdu_enable, 211
- wifi_11ax_config_t, 31
 - band, 31
 - ext_id, 31
 - he_mac_cap, 31
 - he_phy_cap, 32
 - he_txrx_mcs_support, 32
 - id, 31
 - len, 31
 - val, 32
- wifi_11d_is_channel_allowed
 - wifi.h, 202
- wifi_11h_enable
 - wifi.h, 210
- wifi_SubBand_t
 - wifi-decl.h, 181
- wifi_add_mcast_filter
 - wifi.h, 196
- wifi_add_to_bypassq
 - wifi.h, 186
- wifi_antcfg_t, 32
 - ant_mode, 32
 - current_antenna, 33
 - evaluate_time, 33
- wifi_auto_reconnect_config_t, 33
 - flags, 34
 - reconnect_counter, 33
 - reconnect_interval, 33
- wifi_bandcfg_t, 34
 - config_bands, 34
 - fw_bands, 34
- wifi_bss_features
 - wifi-decl.h, 180
- wifi_bss_security
 - wifi-decl.h, 180
- wifi_btwt_config_t, 35
 - action, 35
 - max_sta_support, 35
 - nominal_wake, 35
 - sp_gap, 36
 - sub_id, 35
 - tw_t_exponent, 36
 - tw_t_mantissa, 35
 - tw_t_offset, 36
- wifi_cal_data_t, 36
 - data, 37
 - data_len, 36
- wifi_cfg_rf_he_tb_tx
 - wifi.h, 222
- wifi_chan_info_t, 37
 - chan_freq, 37
 - chan_num, 37
 - passive_scan_or_radar_detect, 37
- wifi_chan_list_param_set_t, 38
 - chan_scan_param, 38
 - no_of_channels, 38
- wifi_chan_scan_param_set_t, 39
 - chan_number, 39
 - max_scan_time, 39
 - min_scan_time, 39
- wifi_chanlist_t, 39
 - chan_info, 40
 - num_chans, 40
- wifi_channel_desc_t, 40
 - chan_num, 41
 - chan_width, 41
 - start_freq, 41
- wifi_clear_mgmt_ie
 - wifi.h, 214
- wifi_clock_sync_gpio_tsf_t, 41
 - clock_sync_Role, 42
 - clock_sync_gpio_level_toggle, 42
 - clock_sync_gpio_pin_number, 42
 - clock_sync_gpio_pulse_width, 42
 - clock_sync_mode, 41
- wifi_cloud_keep_alive
 - wifi.h, 214
- wifi_cloud_keep_alive_t, 42
 - cached, 43
 - dst_ip, 44
 - dst_mac, 44
 - dst_port, 44
 - enable, 43
 - mkeep_alive_id, 43
 - packet, 44

- pkt_len, 44
 - reset, 43
 - retry_count, 43
 - retry_interval, 43
 - send_interval, 43
 - src_ip, 44
 - src_mac, 44
 - src_port, 44
- wifi_config_bgscan_and_rssi
 - wifi.h, 192
- wifi_config_roaming
 - wifi.h, 192
- wifi_configure_delay_to_ps
 - wifi.h, 208
- wifi_configure_listen_interval
 - wifi.h, 208
- wifi_configure_null_pkt_interval
 - wifi.h, 208
- wifi_csi_cfg
 - wifi.h, 227
- wifi_csi_config_params_t, 45
 - band_config, 46
 - bss_type, 45
 - channel, 46
 - chip_id, 46
 - csi_enable, 45
 - csi_filter, 46
 - csi_filter_cnt, 46
 - csi_monitor_enable, 46
 - head_id, 45
 - ra4us, 46
 - tail_id, 46
- wifi_csi_filter_t, 47
 - flags, 47
 - mac_addr, 47
 - pkt_type, 47
 - subtype, 47
- wifi_cw_mode_ctrl_t, 48
 - chanInfo, 48
 - channel, 48
 - mode, 48
 - pktLength, 49
 - rateInfo, 49
 - txPower, 48
- wifi_data_rate_t, 49
 - rx_bw, 50
 - rx_data_rate, 49
 - rx_gi, 50
 - tx_bw, 50
 - tx_data_rate, 49
 - tx_gi, 50
- wifi_deauthenticate
 - wifi.h, 229
- wifi_deinit
 - wifi.h, 183
- wifi_deregister_amsdu_data_input_callback
 - wifi.h, 185
- wifi_deregister_data_input_callback
 - wifi.h, 184
- wifi_deregister_deliver_packet_above_callback
 - wifi.h, 185
- wifi_deregister_wrapper_net_is_ip_or_ipv6_callback
 - wifi.h, 186
- wifi_disable_11d_support
 - wifi.h, 203
- wifi_disable_uap_11d_support
 - wifi.h, 203
- wifi_ds_command_type
 - wifi-decl.h, 181
- wifi_ds_rate, 50
 - data_rate, 51
 - param, 51
 - rate_cfg, 51
 - sub_command, 51
- wifi_ecsa_info, 51
 - band_config, 52
 - bss_type, 52
 - channel, 52
- wifi_ed_mac_ctrl_t, 52
 - ed_ctrl_2g, 52
 - ed_ctrl_5g, 53
 - ed_offset_2g, 53
 - ed_offset_5g, 53
- wifi_enable_11d_support
 - wifi.h, 203
- wifi_enable_low_pwr_mode
 - wifi.h, 195
- wifi_enable_uap_11d_support
 - wifi.h, 203
- wifi_enter_deepsleep_power_save
 - wifi.h, 207
- wifi_enter_ieee_power_save
 - wifi.h, 206
- wifi_enter_wnm_power_save
 - wifi.h, 207
- wifi_event
 - wifi_events.h, 238
- wifi_event_reason
 - wifi_events.h, 240
- wifi_events.h, 238
 - wifi_event, 238
 - wifi_event_reason, 240
 - wifi_wakeup_event_t, 240
 - wlan_bss_role, 240
 - wlan_bss_type, 240
- wifi_exit_deepsleep_power_save
 - wifi.h, 207
- wifi_exit_ieee_power_save
 - wifi.h, 206
- wifi_exit_wnm_power_save
 - wifi.h, 207
- wifi_ext_coex_config_t, 53
 - DefaultPriority, 54
 - EXT_RADIO_PRI_ip_gpio_num, 54
 - EXT_RADIO_PRI_ip_gpio_polarity, 54
 - EXT_RADIO_REQ_ip_gpio_num, 54

- EXT_RADIO_REQ_ip_gpio_polarity, 54
- Enabled, 54
- IgnorePriority, 54
- reserved_1, 55
- reserved_2, 55
- WLAN_GRANT_op_gpio_num, 54
- WLAN_GRANT_op_gpio_polarity, 55
- wifi_ext_coex_stats_t, 55
 - ext_radio_pri_count, 56
 - ext_radio_req_count, 55
 - wlan_grant_count, 56
- wififlt_cfg_t, 56
 - criteria, 56
 - mef_entry, 57
 - nentries, 56
- wifi_frame_t, 57
 - frame_type, 57
- wifi_frame_type_t
 - wifi-decl.h, 182
- wifi_fw_version_ext_t, 57
 - version_str, 58
 - version_str_sel, 58
- wifi_fw_version_t, 58
 - version_str, 58
- wifi_get_11ax_rutxpowerlimit_legacy
 - wifi.h, 215
- wifi_get_antenna
 - wifi.h, 198
- wifi_get_country_code
 - wifi.h, 201
- wifi_get_data_rate
 - wifi.h, 212
- wifi_get_default_ht_capab
 - wifi.h, 229
- wifi_get_default_vht_capab
 - wifi.h, 230
- wifi_get_delay_to_ps
 - wifi.h, 208
- wifi_get_device_firmware_version_ext
 - wifi.h, 191
- wifi_get_device_mac_addr
 - wifi.h, 190
- wifi_get_device_uap_mac_addr
 - wifi.h, 191
- wifi_get_fw_info
 - wifi.h, 212
- wifi_get_indrst_cfg
 - wifi.h, 230
- wifi_get_ipv4_multicast_mac
 - wifi.h, 197
- wifi_get_ipv6_multicast_mac
 - wifi.h, 198
- wifi_get_last_cmd_sent_ms
 - wifi.h, 192
- wifi_get_listen_interval
 - wifi.h, 208
- wifi_get_outbuf
 - wifi.h, 192
- wifi_get_region_code
 - wifi.h, 200
- wifi_get_rf_band
 - wifi.h, 220
- wifi_get_rf_bandwidth
 - wifi.h, 220
- wifi_get_rf_channel
 - wifi.h, 220
- wifi_get_rf_otp_cal_data
 - wifi.h, 223
- wifi_get_rf_otp_mac_addr
 - wifi.h, 223
- wifi_get_rf_per
 - wifi.h, 220
- wifi_get_rf_radio_mode
 - wifi.h, 220
- wifi_get_rf_rx_antenna
 - wifi.h, 221
- wifi_get_rf_tx_antenna
 - wifi.h, 221
- wifi_get_scan_result
 - wifi.h, 193
- wifi_get_scan_result_count
 - wifi.h, 194
- wifi_get_sec_channel_offset
 - wifi.h, 226
- wifi_get_tsf_info
 - wifi.h, 219
- wifi_get_turbo_mode
 - wifi.h, 229
- wifi_get_twt_report
 - wifi.h, 217
- wifi_get_tx_power
 - wifi.h, 204
- wifi_get_txratecfg
 - wifi.h, 209
- wifi_get_uap_channel
 - wifi.h, 201
- wifi_get_uap_turbo_mode
 - wifi.h, 229
- wifi_get_value1
 - wifi.h, 192
- wifi_get_wakeup_reason
 - wifi.h, 207
- wifi_get_wpa_ie_in_assoc
 - wifi.h, 196
- wifi_get_xfer_pending
 - wifi.h, 209
- wifi_handle_event_data_pause
 - wifi.h, 224
- wifi_host_11k_cfg
 - wifi.h, 213
- wifi_host_11k_neighbor_req
 - wifi.h, 213
- wifi_host_11v_bss_trans_query
 - wifi.h, 213
- wifi_host_mbo_cfg
 - wifi.h, 227

- wifi_indrst_cfg_t, 59
 - gpio_pin, 59
 - ir_mode, 59
- wifi_init
 - wifi.h, 182
- wifi_init_fcc
 - wifi.h, 183
- wifi_inject_frame
 - wifi.h, 225
- wifi_is_remain_on_channel
 - wifi.h, 231
- wifi_is_wpa_supplicant_input
 - wifi.h, 225
- wifi_low_level_output
 - wifi.h, 186
- wifi_mac_addr_t, 59
 - mac, 60
- wifi_mbo_preferch_cfg
 - wifi.h, 227
- wifi_mbo_send_preferch_wnm
 - wifi.h, 227
- wifi_mef_entry_t, 60
 - action, 60
 - filter_item, 61
 - filter_num, 60
 - mode, 60
 - rpn, 61
- wifi_mef_filter_t, 61
 - byte_seq, 62
 - fill_flag, 61
 - mask_seq, 63
 - num_byte_seq, 62
 - num_bytes, 62
 - num_mask_seq, 62
 - offset, 62
 - pattern, 62
 - repeat, 62
 - type, 62
- wifi_mem_access
 - wifi.h, 199
- wifi_message, 63
 - data, 63
 - event, 63
 - reason, 63
- wifi_mfg_cmd_IEEEtypes_CtlBasicTrigHdr_t, 67
 - action, 67
 - basic_trig_user_info, 69
 - dest_addr, 68
 - device_id, 67
 - duration, 68
 - enable_tx, 68
 - error, 68
 - frmCtl, 68
 - mfg_cmd, 67
 - src_addr, 68
 - standalone_hetb, 68
 - trig_common_field, 68
 - trig_user_info_field, 69
- wifi_mfg_cmd_generic_cfg_t, 64
 - action, 64
 - data1, 64
 - data2, 65
 - data3, 65
 - device_id, 64
 - error, 64
 - mfg_cmd, 64
- wifi_mfg_cmd_he_tb_tx_t, 65
 - action, 65
 - aid, 66
 - axq_mu_timer, 66
 - device_id, 66
 - enable, 66
 - error, 66
 - mfg_cmd, 65
 - qnum, 66
 - tx_power, 66
- wifi_mfg_cmd_otp_cal_data_rd_wr_t, 69
 - action, 69
 - cal_data, 70
 - cal_data_len, 70
 - cal_data_status, 70
 - device_id, 70
 - error, 70
 - mfg_cmd, 69
- wifi_mfg_cmd_otp_mac_addr_rd_wr_t, 70
 - action, 71
 - device_id, 71
 - error, 71
 - mac_addr, 71
 - mfg_cmd, 71
- wifi_mfg_cmd_tx_cont_t, 71
 - act_sub_ch, 73
 - action, 72
 - cs_mode, 73
 - cw_mode, 72
 - device_id, 72
 - enable_tx, 72
 - error, 72
 - mfg_cmd, 72
 - payload_pattern, 72
 - rsvd, 73
 - tx_rate, 73
- wifi_mfg_cmd_tx_frame_t, 74
 - act_sub_ch, 76
 - action, 74
 - adjust_burst_sifs, 75
 - adv_coding, 76
 - BeamChange, 77
 - bssid, 75
 - burst_sifs_in_us, 76
 - data_rate, 75
 - Dcm, 77
 - device_id, 75
 - Doppler, 77
 - enable, 75
 - error, 75

- frame_length, 75
- frame_pattern, 75
- gf_mode, 76
- MaxPE, 77
- mfg_cmd, 74
- MidP, 77
- NumPkt, 77
- QNum, 78
- rsvd, 77
- short_gi, 76
- short_preamble, 76
- signal_bw, 77
- stbc, 76
- tx_bf, 76
- wifi_mgmt_frame_t, 78
 - addr1, 79
 - addr2, 79
 - addr3, 79
 - addr4, 79
 - duration_id, 79
 - frame_ctrl_flags, 79
 - frame_type, 78
 - frm_len, 78
 - payload, 79
 - seq_ctl, 79
- wifi_nat_keep_alive_t, 80
 - dst_ip, 80
 - dst_mac, 80
 - dst_port, 80
 - interval, 80
- wifi_nxp_reset_scan_flag
 - wifi.h, 226
- wifi_nxp_scan_res_get
 - wifi.h, 226
- wifi_nxp_set_default_scan_ies
 - wifi.h, 226
- wifi_nxp_survey_res_get
 - wifi.h, 226
- wifi_os_mem_info, 81
 - alloc_cnt, 81
 - free_cnt, 81
 - line_num, 81
 - name, 81
 - size, 81
- wifi_ping.h, 241
 - PING_DEFAULT_COUNT, 243
 - PING_DEFAULT_SIZE, 243
 - PING_DEFAULT_TIMEOUT_SEC, 243
 - PING_INTERVAL, 242
 - PING_ID, 242
 - PING_MAX_COUNT, 243
 - PING_MAX_SIZE, 243
 - ping_cli_deinit, 242
 - ping_cli_init, 241
 - ping_e, 242
 - ping_stats, 241
 - ping_w, 242
- wifi_pmf_params_t, 82
 - mfpc, 82
 - mfpr, 82
- wifi_process_hs_cfg_resp
 - wifi.h, 198
- wifi_process_ps_enh_response
 - wifi.h, 199
- wifi_rate_cfg_t, 82
 - nss, 83
 - rate, 83
 - rate_format, 83
 - rate_index, 83
 - rate_setting, 83
- wifi_raw_packet_recv
 - wifi.h, 215
- wifi_raw_packet_send
 - wifi.h, 214
- wifi_reg_t
 - wifi.h, 236
- wifi_register_amsdu_data_input_callback
 - wifi.h, 185
- wifi_register_data_input_callback
 - wifi.h, 184
- wifi_register_deliver_packet_above_callback
 - wifi.h, 185
- wifi_register_event_queue
 - wifi.h, 193
- wifi_register_fw_dump_cb
 - wifi.h, 223
- wifi_register_wrapper_net_is_ip_or_ipv6_callback
 - wifi.h, 185
- wifi_remain_on_channel_t, 84
 - bandcfg, 84
 - channel, 84
 - remain_period, 84
 - remove, 84
 - status, 84
- wifi_remove_mcast_filter
 - wifi.h, 197
- wifi_rf_channel_t, 85
 - current_channel, 85
 - rf_type, 85
- wifi_rf_trigger_frame_cfg
 - wifi.h, 222
- wifi_rssi_info_t, 86
 - bcn_nf_avg, 87
 - bcn_nf_last, 87
 - bcn_rssi_avg, 87
 - bcn_rssi_last, 87
 - bcn_snr_avg, 87
 - bcn_snr_last, 86
 - data_nf_avg, 86
 - data_nf_last, 86
 - data_rssi_avg, 86
 - data_rssi_last, 86
 - data_snr_avg, 87
 - data_snr_last, 87
- wifi_rupwrlimit_config_t, 88
 - chan_num, 88

- ruPower, 88
- start_freq, 88
- width, 88
- wifi_rutxpwrlimit_t, 89
 - num_chans, 89
 - rupwrlimit_config, 89
- wifi_rx_block_cnt
 - wifi.h, 237
- wifi_rx_status
 - wifi.h, 237
- wifi_same_ess_ft
 - wifi.h, 213
- wifi_scan_chan_list_t, 90
 - chan_number, 90
 - num_of_chan, 90
- wifi_scan_channel_list_t, 90
 - chan_number, 91
 - radio_type, 91
 - scan_time, 91
 - scan_type, 91
- wifi_scan_params_t, 91
 - bss_type, 92
 - bssid, 91
 - channel, 92
 - scan_duration, 92
 - split_scan_delay, 92
 - ssid, 92
- wifi_scan_params_v2_t, 92
 - bssid, 93
 - cb, 94
 - chan_list, 93
 - is_bssid, 93
 - is_ssid, 93
 - num_channels, 93
 - num_probes, 94
 - scan_chan_gap, 94
 - scan_only, 93
 - ssid, 93
- wifi_scan_process_results
 - wifi.h, 199
- wifi_scan_result2, 94
 - ap_mfpc, 97
 - ap_mfpr, 97
 - ap_pwe, 97
 - band, 98
 - beacon_period, 96
 - bss_transition_supported, 99
 - bssid, 95
 - Channel, 96
 - dtim_period, 96
 - is_ibss_bit_set, 95
 - is_pmf_required, 97
 - mbo_assoc_disallowed, 99
 - mdid, 99
 - neighbor_report_supported, 99
 - phecap_ie_present, 98
 - phtcap_ie_present, 97
 - phtinfo_ie_present, 97
 - pvhtcap_ie_present, 98
 - RSSI, 96
 - rsn_mcstCipher, 96
 - rsn_ucstCipher, 97
 - ssid, 95
 - ssid_len, 95
 - trans_bssid, 99
 - trans_mode, 98
 - trans_ssid, 99
 - trans_ssid_len, 99
 - WPA_WPA2_WEP, 96
 - wmm_ie_present, 98
 - wpa2_entp_IE_exist, 98
 - wpa_mcstCipher, 96
 - wpa_ucstCipher, 96
 - wps_IE_exist, 98
 - wps_session, 98
- wifi_send_hs_cfg_cmd
 - wifi.h, 205
- wifi_send_mgmt_auth_request
 - wifi.h, 228
- wifi_send_scan_cmd
 - wifi.h, 228
- wifi_set_11ax_cfg
 - wifi.h, 216
- wifi_set_11ax_rutxpowerlimit
 - wifi.h, 215
- wifi_set_11ax_rutxpowerlimit_legacy
 - wifi.h, 215
- wifi_set_11ax_tol_time
 - wifi.h, 215
- wifi_set_11ax_tx_omi
 - wifi.h, 215
- wifi_set_antenna
 - wifi.h, 198
- wifi_set_auto_arp
 - wifi.h, 214
- wifi_set_btwt_cfg
 - wifi.h, 216
- wifi_set_cal_data
 - wifi.h, 195
- wifi_set_clocksync_cfg
 - wifi.h, 219
- wifi_set_country_code
 - wifi.h, 201
- wifi_set_country_ie_ignore
 - wifi.h, 201
- wifi_set_frag
 - wifi.h, 213
- wifi_set_htcapinfo
 - wifi.h, 204
- wifi_set_httxcfg
 - wifi.h, 204
- wifi_set_indrst_cfg
 - wifi.h, 230
- wifi_set_mac_addr
 - wifi.h, 196
- wifi_set_packet_filters

[wifi.h](#), [211](#)
[wifi_set_packet_retry_count](#)
[wifi.h](#), [186](#)
[wifi_set_power_save_mode](#)
[wifi.h](#), [207](#)
[wifi_set_ps_cfg](#)
[wifi.h](#), [205](#)
[wifi_set_region_code](#)
[wifi.h](#), [200](#)
[wifi_set_region_power_cfg](#)
[wifi.h](#), [203](#)
[wifi_set_rf_band](#)
[wifi.h](#), [220](#)
[wifi_set_rf_bandwidth](#)
[wifi.h](#), [220](#)
[wifi_set_rf_channel](#)
[wifi.h](#), [219](#)
[wifi_set_rf_otp_cal_data](#)
[wifi.h](#), [223](#)
[wifi_set_rf_otp_mac_addr](#)
[wifi.h](#), [223](#)
[wifi_set_rf_radio_mode](#)
[wifi.h](#), [219](#)
[wifi_set_rf_rx_antenna](#)
[wifi.h](#), [221](#)
[wifi_set_rf_test_mode](#)
[wifi.h](#), [219](#)
[wifi_set_rf_tx_antenna](#)
[wifi.h](#), [221](#)
[wifi_set_rf_tx_cont_mode](#)
[wifi.h](#), [221](#)
[wifi_set_rf_tx_frame](#)
[wifi.h](#), [222](#)
[wifi_set_rf_tx_power](#)
[wifi.h](#), [221](#)
[wifi_set_rssi_low_threshold](#)
[wifi.h](#), [224](#)
[wifi_set_rts](#)
[wifi.h](#), [213](#)
[wifi_set_rx_status](#)
[wifi.h](#), [184](#)
[wifi_set_sta_mac_filter](#)
[wifi.h](#), [214](#)
[wifi_set_turbo_mode](#)
[wifi.h](#), [229](#)
[wifi_set_twt_setup_cfg](#)
[wifi.h](#), [216](#)
[wifi_set_twt_teardown_cfg](#)
[wifi.h](#), [217](#)
[wifi_set_tx_power](#)
[wifi.h](#), [204](#)
[wifi_set_tx_status](#)
[wifi.h](#), [183](#)
[wifi_set_txbfcap](#)
[wifi.h](#), [204](#)
[wifi_set_txratecfg](#)
[wifi.h](#), [209](#)
[wifi_set_uap_turbo_mode](#)
[wifi.h](#), [229](#)
[wifi_set_xfer_pending](#)
[wifi.h](#), [209](#)
[wifi_show_os_mem_stat](#)
[wifi.h](#), [225](#)
[wifi_shutdown_enable](#)
[wifi.h](#), [238](#)
[wifi_sta_ampdu_rx_disable](#)
[wifi.h](#), [190](#)
[wifi_sta_ampdu_rx_enable](#)
[wifi.h](#), [188](#)
[wifi_sta_ampdu_rx_enable_per_tid](#)
[wifi.h](#), [188](#)
[wifi_sta_ampdu_rx_enable_per_tid_is_allowed](#)
[wifi.h](#), [188](#)
[wifi_sta_ampdu_tx_disable](#)
[wifi.h](#), [187](#)
[wifi_sta_ampdu_tx_enable](#)
[wifi.h](#), [187](#)
[wifi_sta_ampdu_tx_enable_per_tid](#)
[wifi.h](#), [187](#)
[wifi_sta_ampdu_tx_enable_per_tid_is_allowed](#)
[wifi.h](#), [187](#)
[wifi_sta_deauth](#)
[wifi.h](#), [195](#)
[wifi_sta_info_t](#), [100](#)
[mac](#), [100](#)
[power_mgmt_status](#), [100](#)
[rssi](#), [100](#)
[wifi_sta_list_t](#), [101](#)
[count](#), [101](#)
[wifi_stop_bgscan](#)
[wifi.h](#), [192](#)
[wifi_sub_band_set_t](#), [101](#)
[first_chan](#), [101](#)
[max_tx_pwr](#), [102](#)
[no_of_chan](#), [102](#)
[wifi_supp_inject_frame](#)
[wifi.h](#), [225](#)
[wifi_tbt_offset_t](#), [102](#)
[avg_tbt_offset](#), [103](#)
[max_tbt_offset](#), [102](#)
[min_tbt_offset](#), [102](#)
[wifi_tcp_keep_alive](#)
[wifi.h](#), [214](#)
[wifi_tcp_keep_alive_t](#), [103](#)
[dst_ip](#), [104](#)
[dst_mac](#), [104](#)
[dst_tcp_port](#), [104](#)
[enable](#), [103](#)
[interval](#), [104](#)
[max_keep_alives](#), [104](#)
[reset](#), [103](#)
[seq_no](#), [104](#)
[src_tcp_port](#), [104](#)
[timeout](#), [104](#)
[wifi_test_independent_reset](#)
[wifi.h](#), [231](#)

wifi_trigger_oob_indrst
 wifi.h, 231

wifi_tsf_info_t, 105
 tsf, 105
 tsf_format, 105
 tsf_info, 105
 tsf_offset, 105

wifi_twt_information
 wifi.h, 217

wifi_twt_information_t, 106
 flow_identifier, 106
 information_state, 106
 suspend_duration, 106

wifi_twt_report_t, 107
 data, 107
 length, 107
 reserve, 107
 type, 107

wifi_twt_setup_config_t, 108
 announced, 108
 flow_identifier, 109
 hard_constraint, 109
 implicit, 108
 negotiation_type, 108
 trigger_enabled, 108
 tw_t_exponent, 109
 tw_t_info_disabled, 108
 tw_t_mantissa, 109
 tw_t_request, 109
 tw_t_wakeup_duration, 109

wifi_twt_tearardown_config_t, 110
 flow_identifier, 110
 negotiation_type, 110
 teardown_all_twt, 110

wifi_tx_block_cnt
 wifi.h, 237

wifi_tx_card_awake_lock
 wifi.h, 209

wifi_tx_card_awake_unlock
 wifi.h, 210

wifi_tx_power_t, 110
 current_level, 111
 max_power, 111
 min_power, 111

wifi_tx_status
 wifi.h, 237

wifi_txpwrlimit_config_t, 111
 chan_desc, 112
 num_mod_grps, 112
 txpwrlimit_entry, 112

wifi_txpwrlimit_entry_t, 112
 mod_group, 113
 tx_power, 113

wifi_txpwrlimit_t, 113
 num_chans, 113
 subband, 113
 txpwrlimit_config, 114

wifi_uap_ampdu_rx_disable
 wifi.h, 189

wifi_uap_ampdu_rx_enable
 wifi.h, 189

wifi_uap_ampdu_rx_enable_per_tid
 wifi.h, 189

wifi_uap_ampdu_rx_enable_per_tid_is_allowed
 wifi.h, 189

wifi_uap_ampdu_tx_disable
 wifi.h, 190

wifi_uap_ampdu_tx_enable
 wifi.h, 189

wifi_uap_ampdu_tx_enable_per_tid
 wifi.h, 190

wifi_uap_ampdu_tx_enable_per_tid_is_allowed
 wifi.h, 190

wifi_uap_bss_sta_list
 wifi.h, 194

wifi_uap_client_assoc
 wifi.h, 230

wifi_uap_client_deauth
 wifi.h, 230

wifi_uap_client_disassoc_t, 114
 reason_code, 114
 sta_addr, 114

wifi_uap_config_wifi_capa
 wifi.h, 212

wifi_uap_do_acs
 wifi.h, 212

wifi_uap_enable_11d_support
 wifi.h, 202

wifi_uap_enable_sticky_bit
 wifi.h, 208

wifi_uap_get_bandwidth
 wifi.h, 212

wifi_uap_get_pmfcfg
 wifi.h, 213

wifi_uap_pmf_getset
 wifi.h, 202

wifi_uap_ps_inactivity_sleep_enter
 wifi.h, 206

wifi_uap_ps_inactivity_sleep_exit
 wifi.h, 206

wifi_uap_ps_sta_ageout_timer_getset
 wifi.h, 199

wifi_uap_rates_getset
 wifi.h, 199

wifi_uap_set_bandwidth
 wifi.h, 212

wifi_uap_set_httxcf
 wifi.h, 204

wifi_uap_set_httxcf_int
 wifi.h, 204

wifi_uap_sta_ageout_timer_getset
 wifi.h, 199

wifi_uap_start
 wifi.h, 211

wifi_uap_stop
 wifi.h, 211

- wifi_unregister_event_queue
 - wifi.h, 193
- wifi_unset_rf_test_mode
 - wifi.h, 219
- wifi_update_last_cmd_sent_ms
 - wifi.h, 192
- wifi_wake_up_card
 - wifi.h, 209
- wifi_wakeup_event_t
 - wifi_events.h, 240
- wifi_wmm_get_packet_cnt
 - wifi.h, 224
- wifi_wmm_get_pkt_prio
 - wifi.h, 224
- wifi_wmm_init
 - wifi.h, 224
- wifi_wmm_tx_stats_dump
 - wifi.h, 224
- wifi_wowlan_pattern_t, 115
 - mask, 115
 - pattern, 115
 - pattern_len, 115
 - pkt_offset, 115
- wifi_wowlan_ptn_cfg_t, 115
 - enable, 116
 - n_patterns, 116
 - patterns, 116
- wifi_wpa_supplicant_eapol_input
 - wifi.h, 226
- wlan.h, 243
 - A_ID_MAX_LENGTH, 370
 - ACTION_GET, 368
 - ACTION_SET, 368
 - ARG_UNUSED, 367
 - address_types, 393
 - BIN_COUNTER_LEN, 378
 - CARD_WAKEUP_GPIO_PIN, 372
 - CONFIG_WLAN_KNOWN_NETWORKS, 367
 - Chan2Offset_e, 391
 - ChanBand_e, 390
 - ChanWidth_e, 390
 - DFS_REC_HDR_LEN, 378
 - DFS_REC_HDR_NUM, 378
 - DOMAIN_MATCH_MAX_LENGTH, 370
 - ENH_PS_MODES, 389
 - EU_CRYPT0_AAD_MAX_LENGTH, 380
 - EU_CRYPT0_DATA_MAX_LENGTH, 379
 - EU_CRYPT0_KEY_MAX_LENGTH, 379
 - EU_CRYPT0_KEYIV_MAX_LENGTH, 379
 - EU_CRYPT0_NONCE_MAX_LENGTH, 380
 - eap_tls_cipher_type, 392
 - FILE_TYPE_ENTP_CA_CERT2, 380
 - FILE_TYPE_ENTP_CA_CERT, 380
 - FILE_TYPE_ENTP_CLIENT_CERT2, 380
 - FILE_TYPE_ENTP_CLIENT_CERT, 380
 - FILE_TYPE_ENTP_CLIENT_KEY2, 381
 - FILE_TYPE_ENTP_CLIENT_KEY, 380
 - FILE_TYPE_ENTP_DH_PARAMS, 381
 - FILE_TYPE_ENTP_SERVER_CERT, 381
 - FILE_TYPE_ENTP_SERVER_KEY, 381
 - FILE_TYPE_NONE, 380
 - get_scan_params, 245
 - HASH_MAX_LENGTH, 370
 - HOST_WAKEUP_GPIO_PIN, 372
 - Host_Sleep_Action, 389
 - IDENTITY_MAX_LENGTH, 370
 - IEEEtypes_ADDRESS_SIZE, 368
 - IEEEtypes_Bss_t, 385
 - IEEEtypes_SSID_SIZE, 368
 - is_ep_valid_security, 244
 - is_sta_connected, 259
 - is_sta_ipv4_connected, 259
 - is_sta_ipv6_connected, 259
 - is_uap_started, 258
 - is_valid_security, 244
 - MAX_CHANNEL_LIST, 379
 - MAX_USERS, 370
 - NUM_CHAN_BAND_ENUMS, 378
 - PAC_OPAQUE_ENCR_KEY_MAX_LENGTH, 370
 - PASSWORD_MAX_LENGTH, 370
 - print_mac, 312
 - ScanMode_e, 391
 - set_scan_params, 245
 - TX_AMPDU_CTS_2_SELF, 379
 - TX_AMPDU_DISABLE_PROTECTION, 379
 - TX_AMPDU_DYNAMIC_RTS_CTS, 379
 - TX_AMPDU_RTS_CTS, 379
 - verify_scan_channel_value, 244
 - verify_scan_duration_value, 244
 - verify_split_scan_delay, 244
 - WLAN_11D_SCAN_LIMIT, 368
 - WLAN_CIPHER_AES_128_CM4C, 377
 - WLAN_CIPHER_BIP_CM4C_256, 378
 - WLAN_CIPHER_BIP_GMAC_128, 378
 - WLAN_CIPHER_BIP_GMAC_256, 378
 - WLAN_CIPHER_CCMP_256, 377
 - WLAN_CIPHER_CCMP, 377
 - WLAN_CIPHER_GCMP_256, 377
 - WLAN_CIPHER_GCMP, 377
 - WLAN_CIPHER_GTK_NOT_USED, 378
 - WLAN_CIPHER_NONE, 376
 - WLAN_CIPHER_SMS4, 377
 - WLAN_CIPHER_TKIP, 377
 - WLAN_CIPHER_WEP104, 377
 - WLAN_CIPHER_WEP40, 376
 - WLAN_DRV_VERSION, 367
 - WLAN_ERROR_ACTION, 372
 - WLAN_ERROR_NOMEM, 371
 - WLAN_ERROR_NONE, 371
 - WLAN_ERROR_NOT_SUPPORTED, 372
 - WLAN_ERROR_PARAM, 371
 - WLAN_ERROR_PS_ACTION, 372
 - WLAN_ERROR_STATE, 371
 - WLAN_KEY_MGMT_CCKM, 374
 - WLAN_KEY_MGMT_DPP, 376
 - WLAN_KEY_MGMT_FILS_SHA256, 375

- WLAN_KEY_MGMT_FILS_SHA384, [375](#)
- WLAN_KEY_MGMT_FT_FILS_SHA256, [375](#)
- WLAN_KEY_MGMT_FT_FILS_SHA384, [375](#)
- WLAN_KEY_MGMT_FT_IEEE8021X_SHA384, [376](#)
- WLAN_KEY_MGMT_FT_IEEE8021X, [373](#)
- WLAN_KEY_MGMT_FT_PSK, [373](#)
- WLAN_KEY_MGMT_FT_SAE, [374](#)
- WLAN_KEY_MGMT_FT, [376](#)
- WLAN_KEY_MGMT_IEEE8021X_NO_WPA, [373](#)
- WLAN_KEY_MGMT_IEEE8021X_SHA256, [374](#)
- WLAN_KEY_MGMT_IEEE8021X_SUITE_B_192, [375](#)
- WLAN_KEY_MGMT_IEEE8021X_SUITE_B, [375](#)
- WLAN_KEY_MGMT_IEEE8021X, [373](#)
- WLAN_KEY_MGMT_NONE, [373](#)
- WLAN_KEY_MGMT_OSEN, [375](#)
- WLAN_KEY_MGMT_OWE, [375](#)
- WLAN_KEY_MGMT_PASN, [376](#)
- WLAN_KEY_MGMT_PSK_SHA256, [374](#)
- WLAN_KEY_MGMT_PSK, [373](#)
- WLAN_KEY_MGMT_SAE_EXT_KEY, [376](#)
- WLAN_KEY_MGMT_SAE, [374](#)
- WLAN_KEY_MGMT_WAPI_CERT, [374](#)
- WLAN_KEY_MGMT_WAPI_PSK, [374](#)
- WLAN_KEY_MGMT_WPA_NONE, [373](#)
- WLAN_KEY_MGMT_WPS, [374](#)
- WLAN_MAC_ADDR_LENGTH, [371](#)
- WLAN_MAX_KNOWN_NETWORKS, [371](#)
- WLAN_MAX_STA_FILTER_NUM, [371](#)
- WLAN_MGMT_ACTION, [373](#)
- WLAN_MGMT_AUTH, [372](#)
- WLAN_MGMT_DEAUTH, [372](#)
- WLAN_MGMT_DIASOC, [372](#)
- WLAN_NETWORK_NAME_MAX_LENGTH, [369](#)
- WLAN_NETWORK_NAME_MIN_LENGTH, [369](#)
- WLAN_PASSWORD_MAX_LENGTH, [369](#)
- WLAN_PASSWORD_MIN_LENGTH, [369](#)
- WLAN_PMK_LENGTH, [371](#)
- WLAN_PSK_MAX_LENGTH, [369](#)
- WLAN_PSK_MIN_LENGTH, [369](#)
- WLAN_REASON_CODE_PREV_AUTH_NOT_VALID, [368](#)
- WLAN_RECONNECT_LIMIT, [369](#)
- WLAN_RESCAN_LIMIT, [368](#)
- wlan_11ax_config_t, [383](#)
- wlan_add_network, [248](#)
- wlan_auto_reconnect_config_t, [382](#)
- wlan_auto_reconnect_disable, [303](#)
- wlan_auto_reconnect_enable, [302](#)
- wlan_bandcfg_t, [382](#)
- wlan_basic_cli_deinit, [284](#)
- wlan_basic_cli_init, [284](#)
- wlan_btwt_config_t, [384](#)
- wlan_cal_data_t, [381](#)
- wlan_cfg_rf_he_tb_tx, [317](#)
- wlan_chanlist_t, [383](#)
- wlan_check_11ac_capa, [351](#)
- wlan_check_11ax_capa, [351](#)
- wlan_check_11n_capa, [350](#)
- wlan_clear_mgmt_ie, [296](#)
- wlan_cli_deinit, [285](#)
- wlan_cli_init, [285](#)
- wlan_clock_sync_gpio_tsf_t, [384](#)
- wlan_cloud_keep_alive_enabled, [356](#)
- wlan_cloud_keep_alive_t, [382](#)
- wlan_config_mef, [345](#)
- wlan_configure_delay_to_ps, [271](#)
- wlan_configure_listen_interval, [270](#)
- wlan_configure_null_pkt_interval, [272](#)
- wlan_connect, [249](#)
- wlan_connect_opt, [251](#)
- wlan_connection_state, [388](#)
- wlan_csi_cfg, [346](#)
- wlan_csi_config_params_t, [384](#)
- wlan_csi_opt, [390](#)
- wlan_cw_mode_ctrl_t, [383](#)
- wlan_deepsleeps_off, [275](#)
- wlan_deepsleeps_on, [275](#)
- wlan_deinit, [246](#)
- wlan_disconnect, [252](#)
- wlan_dpp_auth_init, [362](#)
- wlan_dpp_bootstrap_gen, [361](#)
- wlan_dpp_bootstrap_get_uri, [361](#)
- wlan_dpp_chirp, [364](#)
- wlan_dpp_configurator_add, [359](#)
- wlan_dpp_configurator_params, [360](#)
- wlan_dpp_configurator_sign, [364](#)
- wlan_dpp_listen, [362](#)
- wlan_dpp_mud_url, [360](#)
- wlan_dpp_pkex_add, [363](#)
- wlan_dpp_qr_code, [361](#)
- wlan_dpp_reconfig, [364](#)
- wlan_dpp_stop_listen, [363](#)
- wlan_ds_rate, [382](#)
- wlan_ed_mac_ctrl_t, [382](#)
- wlan_enable_disable_htc, [333](#)
- wlan_enable_low_pwr_mode, [264](#)
- wlan_enhanced_cli_deinit, [286](#)
- wlan_enhanced_cli_init, [286](#)
- wlan_event_reason, [386](#)
- wlan_ext_coex_config_t, [383](#)
- wlan_ext_coex_stats_t, [383](#)
- wlanflt_cfg_t, [382](#)
- wlan_free_entp_cert_files, [350](#)
- wlan_ft_roam, [340](#)
- wlan_get_11ax_cfg, [336](#)
- wlan_get_11ax_rtxpowerlimit_legacy, [335](#)
- wlan_get_11d_enable_status, [296](#)
- wlan_get_address, [254](#)
- wlan_get_antcfg, [273](#)
- wlan_get_auto_reconnect_config, [303](#)
- wlan_get_average_signal_strength, [297](#)
- wlan_get_beacon_period, [276](#)
- wlan_get_board_type, [366](#)
- wlan_get_btwt_cfg, [337](#)

wlan_get_cal_data, 299
wlan_get_chanlist, 300
wlan_get_connection_state, 261
wlan_get_current_bssid, 282
wlan_get_current_channel, 282
wlan_get_current_network, 256
wlan_get_current_network_bssid, 257
wlan_get_current_network_ssid, 257
wlan_get_current_nf, 245
wlan_get_current_rssi, 245
wlan_get_current_signal_strength, 297
wlan_get_current_uap_network, 257
wlan_get_current_uap_network_ssid, 258
wlan_get_data_rate, 277
wlan_get_delay_to_ps, 271
wlan_get_dtim_period, 276
wlan_get_ed_mac_mode, 266
wlan_get_entp_cert_files, 350
wlan_get_ext_coex_stats, 295
wlan_get_firmware_version_ext, 273
wlan_get_host_11k_status, 341
wlan_get_indrst_cfg, 365
wlan_get_listen_interval, 271
wlan_get_mac_address, 254
wlan_get_mac_address_uap, 254
wlan_get_mgmt_ie, 294
wlan_get_network, 259
wlan_get_network_byname, 260
wlan_get_network_count, 261
wlan_get_otp_user_data, 298
wlan_get_pmfcfg, 277
wlan_get_ps_mode, 282
wlan_get_region_code, 359
wlan_get_rf_band, 315
wlan_get_rf_bandwidth, 316
wlan_get_rf_channel, 314
wlan_get_rf_otp_cal_data, 324
wlan_get_rf_otp_mac_addr, 323
wlan_get_rf_per, 316
wlan_get_rf_radio_mode, 314
wlan_get_rf_rx_antenna, 321
wlan_get_rf_tx_antenna, 320
wlan_get_roaming_status, 270
wlan_get_scan_result, 263
wlan_get_signal_info, 351
wlan_get_sta_tx_power, 293
wlan_get_tsf, 274
wlan_get_tsf_info, 339
wlan_get_turbo_mode, 353
wlan_get_twt_report, 338
wlan_get_twt_setup_cfg, 337
wlan_get_twt_teartdown_cfg, 338
wlan_get_txpwrlimit, 301
wlan_get_txratecfg, 293
wlan_get_uap_address, 255
wlan_get_uap_channel, 256
wlan_get_uap_connection_state, 262
wlan_get_uap_ed_mac_mode, 266
wlan_get_uap_max_clients, 288
wlan_get_uap_supported_max_clients, 288
wlan_get_uap_turbo_mode, 353
wlan_get_wlan_region_code, 294
wlan_host_11k_cfg, 341
wlan_host_11k_neighbor_req, 342
wlan_host_11v_bss_trans_query, 342
wlan_host_set_sta_mac_filter, 365
wlan_ieeeaps_off, 275
wlan_ieeeaps_on, 274
wlan_independent_reset, 365
wlan_indrst_cfg_t, 385
wlan_init, 245
wlan_initialize_sta_network, 247
wlan_initialize_uap_network, 247
wlan_is_power_save_enabled, 272
wlan_mef_set_auto_arp, 344
wlan_mef_set_auto_ping, 344
wlan_mef_type, 393
wlan_mgmt_frame_t, 384
wlan_monitor_opt, 390
wlan_pmksa_flush, 343
wlan_pmksa_list, 343
wlan_ps_mode, 388
wlan_ps_state, 389
wlan_reassociate, 252
wlan_register_csi_user_callback, 346
wlan_register_fw_dump_cb, 324
wlan_remain_on_channel, 298
wlan_remove_all_network_profiles, 247
wlan_remove_network, 248
wlan_rf_trigger_frame_cfg, 318
wlan_rssi_info_t, 385
wlan_rutxpwrlimit_t, 383
wlan_rx_mgmt_indication, 340
wlan_save_cloud_keep_alive_params, 356
wlan_scan, 262
wlan_scan_channel_list_t, 381
wlan_scan_params_v2_t, 381
wlan_scan_with_opt, 263
wlan_security_type, 391
wlan_send_debug_htc, 333
wlan_send_hostcmd, 332
wlan_set_11ax_cfg, 336
wlan_set_11ax_rutxpowerlimit, 335
wlan_set_11ax_rutxpowerlimit_legacy, 335
wlan_set_11ax_tol_time, 334
wlan_set_11ax_tx_omi, 334
wlan_set_11d_state, 359
wlan_set_antcfg, 272
wlan_set_auto_arp, 281
wlan_set_btwt_cfg, 336
wlan_set_cal_data, 266
wlan_set_chanlist, 300
wlan_set_chanlist_and_txpwrlimit, 299
wlan_set_clocksync_cfg, 339
wlan_set_country_code, 357
wlan_set_country_ie_ignore, 358

wlan_set_crypto_AES_CCMP_decrypt, 329
wlan_set_crypto_AES_CCMP_encrypt, 329
wlan_set_crypto_AES_ECB_decrypt, 327
wlan_set_crypto_AES_ECB_encrypt, 326
wlan_set_crypto_AES_GCMP_decrypt, 331
wlan_set_crypto_AES_GCMP_encrypt, 330
wlan_set_crypto_AES_WRAP_decrypt, 328
wlan_set_crypto_AES_WRAP_encrypt, 327
wlan_set_crypto_RC4_decrypt, 325
wlan_set_crypto_RC4_encrypt, 325
wlan_set_ed_mac_mode, 264
wlan_set_entp_cert_files, 349
wlan_set_ext_coex_config, 296
wlan_set_frag, 311
wlan_set_htcapinfo, 289
wlan_set_httxcfg, 290
wlan_set_ieee80211_cfg, 270
wlan_set_indrst_cfg, 365
wlan_set_ipv6_ns_mef, 345
wlan_set_ipv6_ns_offload, 281
wlan_set_mac_addr, 268
wlan_set_mgmt_ie, 295
wlan_set_network_ip_byname, 366
wlan_set_okc, 342
wlan_set_packet_filters, 278
wlan_set_ps_cfg, 355
wlan_set_reassoc_control, 303
wlan_set_region_code, 358
wlan_set_region_power_cfg, 299
wlan_set_rf_band, 314
wlan_set_rf_bandwidth, 315
wlan_set_rf_channel, 313
wlan_set_rf_otp_cal_data, 323
wlan_set_rf_otp_mac_addr, 322
wlan_set_rf_radio_mode, 313
wlan_set_rf_rx_antenna, 320
wlan_set_rf_test_mode, 312
wlan_set_rf_tx_antenna, 319
wlan_set_rf_tx_cont_mode, 317
wlan_set_rf_tx_frame, 322
wlan_set_rf_tx_power, 321
wlan_set_rg_power_cfg, 353
wlan_set_roaming, 269
wlan_set_rssi_low_threshold, 347
wlan_set_rts, 310
wlan_set_scan_channel_gap, 341
wlan_set_scan_interval, 343
wlan_set_sta_mac_addr, 268
wlan_set_sta_mac_filter, 312
wlan_set_sta_tx_power, 293
wlan_set_turbo_mode, 355
wlan_set_twt_setup_cfg, 337
wlan_set_twt_tearardown_cfg, 337
wlan_set_txpwrlimit, 301
wlan_set_txratecfg, 291
wlan_set_uap_ed_mac_mode, 265
wlan_set_uap_frag, 311
wlan_set_uap_mac_addr, 268
wlan_set_uap_max_clients, 289
wlan_set_uap_rts, 311
wlan_set_uap_turbo_mode, 355
wlan_set_wwsm_txpwrlimit, 294
wlan_show_os_mem_stat, 340
wlan_sta_ampdu_rx_disable, 309
wlan_sta_ampdu_rx_enable, 309
wlan_sta_ampdu_tx_disable, 308
wlan_sta_ampdu_tx_enable, 308
wlan_start, 246
wlan_start_ap_wps_pbc, 349
wlan_start_ap_wps_pin, 348
wlan_start_cloud_keep_alive, 357
wlan_start_network, 252
wlan_start_wps_pbc, 348
wlan_start_wps_pin, 347
wlan_stop, 246
wlan_stop_cloud_keep_alive, 357
wlan_stop_network, 253
wlan_string_dup, 366
wlan_tcp_keep_alive, 275
wlan_tcp_keep_alive_t, 382
wlan_test_mode_cli_deinit, 287
wlan_test_mode_cli_init, 287
wlan_tsf_info_t, 384
wlan_twt_information, 338
wlan_twt_information_t, 384
wlan_twt_report_t, 384
wlan_twt_setup_config_t, 383
wlan_twt_tearardown_config_t, 384
wlan_tx_ampdu_prot_mode, 344
wlan_txpwrlimit_t, 383
wlan_txrate_setting, 385
wlan_uap_ampdu_rx_disable, 310
wlan_uap_ampdu_rx_enable, 309
wlan_uap_ampdu_tx_disable, 309
wlan_uap_ampdu_tx_enable, 309
wlan_uap_client_disassoc_t, 385
wlan_uap_ctrl_deauth, 306
wlan_uap_disconnect_sta, 366
wlan_uap_get_bandwidth, 304
wlan_uap_get_pmfcfg, 278
wlan_uap_set_bandwidth, 304
wlan_uap_set_beacon_period, 304
wlan_uap_set_ecsa, 307
wlan_uap_set_hidden_ssid, 306
wlan_uap_set_htcapinfo, 307
wlan_uap_set_httxcfg, 308
wlan_uap_set_scan_chan_list, 310
wlan_unregister_csi_user_callback, 346
wlan_unset_rf_test_mode, 273
wlan_version_extended, 273
wlan_wakeup_event_t, 387
wlan_wfa_basic_cli_deinit, 283
wlan_wfa_basic_cli_init, 283
wlan_wlcmgr_send_msg, 283
wlan_wmm_tx_stats_dump, 341
wlan_wowlan_cfg_ptn_match, 281

- wlan_wowlan_ptn_cfg_t, 382
- wlan_wps_ap_cancel, 349
- wlan_wps_cancel, 348
- wlan_wps_generate_pin, 347
- wlcm_d, 367
- wlcm_e, 367
- wlcm_w, 367
- wm_wlan_errno, 385
- wlan_11ax_config_t
 - wlan.h, 383
- wlan_11d.h, 393
 - wlan_enable_11d, 394
 - wlan_enable_uap_11d, 394
- wlan_add_network
 - wlan.h, 248
- wlan_auto_reconnect_config_t
 - wlan.h, 382
- wlan_auto_reconnect_disable
 - wlan.h, 303
- wlan_auto_reconnect_enable
 - wlan.h, 302
- wlan_bandcfg_t
 - wlan.h, 382
- wlan_basic_cli_deinit
 - wlan.h, 284
- wlan_basic_cli_init
 - wlan.h, 284
- wlan_bss_role
 - wifi_events.h, 240
- wlan_bss_type
 - wifi_events.h, 240
- wlan_btwt_config_t
 - wlan.h, 384
- wlan_cal_data_t
 - wlan.h, 381
- wlan_capa
 - wlan_network, 127
- wlan_cfg_rf_he_tb_tx
 - wlan.h, 317
- wlan_chanlist_t
 - wlan.h, 383
- wlan_check_11ac_capa
 - wlan.h, 351
- wlan_check_11ax_capa
 - wlan.h, 351
- wlan_check_11n_capa
 - wlan.h, 350
- wlan_cipher, 116
 - aes_128_mac, 117
 - bip_mac_256, 118
 - bip_gmac_128, 118
 - bip_gmac_256, 118
 - ccmp, 117
 - ccmp_256, 118
 - gcmap, 117
 - gcmap_256, 118
 - gtk_not_used, 118
 - none, 117
 - rsvd, 118
 - rsvd2, 119
 - sms4, 118
 - tkip, 117
 - wep104, 117
 - wep40, 117
- wlan_clear_mgmt_ie
 - wlan.h, 296
- wlan_cli_deinit
 - wlan.h, 285
- wlan_cli_init
 - wlan.h, 285
- wlan_clock_sync_gpio_tsf_t
 - wlan.h, 384
- wlan_cloud_keep_alive_enabled
 - wlan.h, 356
- wlan_cloud_keep_alive_t
 - wlan.h, 382
- wlan_config_mef
 - wlan.h, 345
- wlan_configure_delay_to_ps
 - wlan.h, 271
- wlan_configure_listen_interval
 - wlan.h, 270
- wlan_configure_null_pkt_interval
 - wlan.h, 272
- wlan_connect
 - wlan.h, 249
- wlan_connect_opt
 - wlan.h, 251
- wlan_connection_state
 - wlan.h, 388
- wlan_csi_cfg
 - wlan.h, 346
- wlan_csi_config_params_t
 - wlan.h, 384
- wlan_csi_opt
 - wlan.h, 390
- wlan_cw_mode_ctrl_t
 - wlan.h, 383
- wlan_deepsleeps_off
 - wlan.h, 275
- wlan_deepsleeps_on
 - wlan.h, 275
- wlan_deinit
 - wlan.h, 246
- wlan_disconnect
 - wlan.h, 252
- wlan_dpp_auth_init
 - wlan.h, 362
- wlan_dpp_bootstrap_gen
 - wlan.h, 361
- wlan_dpp_bootstrap_get_uri
 - wlan.h, 361
- wlan_dpp_chirp
 - wlan.h, 364
- wlan_dpp_configurator_add
 - wlan.h, 359

wlan_dpp_configurator_params
wlan.h, 360

wlan_dpp_configurator_sign
wlan.h, 364

wlan_dpp_listen
wlan.h, 362

wlan_dpp_mud_url
wlan.h, 360

wlan_dpp_pkex_add
wlan.h, 363

wlan_dpp_qr_code
wlan.h, 361

wlan_dpp_reconfig
wlan.h, 364

wlan_dpp_stop_listen
wlan.h, 363

wlan_ds_rate
wlan.h, 382

wlan_ed_mac_ctrl_t
wlan.h, 382

wlan_enable_11d
wlan_11d.h, 394

wlan_enable_disable_htc
wlan.h, 333

wlan_enable_low_pwr_mode
wlan.h, 264

wlan_enable_uap_11d
wlan_11d.h, 394

wlan_enhanced_cli_deinit
wlan.h, 286

wlan_enhanced_cli_init
wlan.h, 286

wlan_event_reason
wlan.h, 386

wlan_ext_coex_config_t
wlan.h, 383

wlan_ext_coex_stats_t
wlan.h, 383

wlanflt_cfg_t
wlan.h, 382

wlan_free_entp_cert_files
wlan.h, 350

wlan_ft_roam
wlan.h, 340

wlan_get_11ax_cfg
wlan.h, 336

wlan_get_11ax_rtxpowerlimit_legacy
wlan.h, 335

wlan_get_11d_enable_status
wlan.h, 296

wlan_get_address
wlan.h, 254

wlan_get_antcfg
wlan.h, 273

wlan_get_auto_reconnect_config
wlan.h, 303

wlan_get_average_signal_strength
wlan.h, 297

wlan_get_beacon_period
wlan.h, 276

wlan_get_board_type
wlan.h, 366

wlan_get_btwt_cfg
wlan.h, 337

wlan_get_cal_data
wlan.h, 299

wlan_get_chanlist
wlan.h, 300

wlan_get_connection_state
wlan.h, 261

wlan_get_current_bssid
wlan.h, 282

wlan_get_current_channel
wlan.h, 282

wlan_get_current_network
wlan.h, 256

wlan_get_current_network_bssid
wlan.h, 257

wlan_get_current_network_ssid
wlan.h, 257

wlan_get_current_nf
wlan.h, 245

wlan_get_current_rssi
wlan.h, 245

wlan_get_current_signal_strength
wlan.h, 297

wlan_get_current_uap_network
wlan.h, 257

wlan_get_current_uap_network_ssid
wlan.h, 258

wlan_get_data_rate
wlan.h, 277

wlan_get_delay_to_ps
wlan.h, 271

wlan_get_dtim_period
wlan.h, 276

wlan_get_ed_mac_mode
wlan.h, 266

wlan_get_entp_cert_files
wlan.h, 350

wlan_get_ext_coex_stats
wlan.h, 295

wlan_get_firmware_version_ext
wlan.h, 273

wlan_get_host_11k_status
wlan.h, 341

wlan_get_indrst_cfg
wlan.h, 365

wlan_get_listen_interval
wlan.h, 271

wlan_get_mac_address
wlan.h, 254

wlan_get_mac_address_uap
wlan.h, 254

wlan_get_mgmt_ie
wlan.h, 294

wlan_get_network
wlan.h, 259

wlan_get_network_byname
wlan.h, 260

wlan_get_network_count
wlan.h, 261

wlan_get_otp_user_data
wlan.h, 298

wlan_get_pmfcfg
wlan.h, 277

wlan_get_ps_mode
wlan.h, 282

wlan_get_region_code
wlan.h, 359

wlan_get_rf_band
wlan.h, 315

wlan_get_rf_bandwidth
wlan.h, 316

wlan_get_rf_channel
wlan.h, 314

wlan_get_rf_otp_cal_data
wlan.h, 324

wlan_get_rf_otp_mac_addr
wlan.h, 323

wlan_get_rf_per
wlan.h, 316

wlan_get_rf_radio_mode
wlan.h, 314

wlan_get_rf_rx_antenna
wlan.h, 321

wlan_get_rf_tx_antenna
wlan.h, 320

wlan_get_roaming_status
wlan.h, 270

wlan_get_scan_result
wlan.h, 263

wlan_get_signal_info
wlan.h, 351

wlan_get_sta_tx_power
wlan.h, 293

wlan_get_tsf
wlan.h, 274

wlan_get_tsf_info
wlan.h, 339

wlan_get_turbo_mode
wlan.h, 353

wlan_get_twt_report
wlan.h, 338

wlan_get_twt_setup_cfg
wlan.h, 337

wlan_get_twt_teartdown_cfg
wlan.h, 338

wlan_get_txpwrlimit
wlan.h, 301

wlan_get_txratecfg
wlan.h, 293

wlan_get_uap_address
wlan.h, 255

wlan_get_uap_channel
wlan.h, 256

wlan_get_uap_connection_state
wlan.h, 262

wlan_get_uap_ed_mac_mode
wlan.h, 266

wlan_get_uap_max_clients
wlan.h, 288

wlan_get_uap_supported_max_clients
wlan.h, 288

wlan_get_uap_turbo_mode
wlan.h, 353

wlan_get_wlan_region_code
wlan.h, 294

wlan_grant_count
wifi_ext_coex_stats_t, 56

wlan_host_11k_cfg
wlan.h, 341

wlan_host_11k_neighbor_req
wlan.h, 342

wlan_host_11v_bss_trans_query
wlan.h, 342

wlan_host_set_sta_mac_filter
wlan.h, 365

wlan_ieeeeps_config, 119
ad hoc_ awake_period, 120
bcn_miss_timeout, 120
delay_to_ps, 120
listen_interval, 119
multiple_dtim_interval, 119
ps_mode, 120
ps_null_interval, 119

wlan_ieeeeps_off
wlan.h, 275

wlan_ieeeeps_on
wlan.h, 274

wlan_independent_reset
wlan.h, 365

wlan_indrst_cfg_t
wlan.h, 385

wlan_init
wlan.h, 245

wlan_initialize_sta_network
wlan.h, 247

wlan_initialize_uap_network
wlan.h, 247

wlan_ip_config, 120
ipv4, 121
ipv6, 121
ipv6_count, 121

wlan_is_power_save_enabled
wlan.h, 272

wlan_mef_set_auto_arp
wlan.h, 344

wlan_mef_set_auto_ping
wlan.h, 344

wlan_mef_type
wlan.h, 393

wlan_message, 121
 data, 121
 id, 121
wlan_mgmt_frame_t
 wlan.h, 384
wlan_monitor_opt
 wlan.h, 390
wlan_network, 122
 acs_band, 124
 beacon_period, 127
 bss_transition_supported, 127
 bssid, 123
 bssid_specific, 125
 btm_mode, 127
 channel, 124
 channel_specific, 126
 dot11ac, 126
 dot11ax, 126
 dot11n, 126
 dtim_period, 127
 ft_1x, 127
 ft_psk, 127
 ft_sae, 127
 he_oper_chwidth, 125
 ht_capab, 124
 id, 123
 ip, 125
 mdid, 126
 name, 123
 neighbor_report_supported, 128
 role, 125
 rssi, 124
 sec_channel_offset, 124
 security, 125
 security_specific, 126
 ssid, 123
 ssid_specific, 125
 type, 125
 vht_capab, 124
 vht_oper_chwidth, 124
 wlan_capa, 127
 wps_network, 123
wlan_network_security, 128
 a_id, 137
 anonymous_identity, 133
 ca_cert2_data, 135
 ca_cert2_len, 135
 ca_cert_data, 133
 ca_cert_hash, 134
 ca_cert_len, 134
 client_cert2_data, 135
 client_cert2_len, 135
 client_cert_data, 134
 client_cert_len, 134
 client_key2_data, 135
 client_key2_len, 135
 client_key2_passwd, 135
 client_key_data, 134
 client_key_len, 134
 client_key_passwd, 134
 dh_data, 136
 dh_len, 136
 domain_match, 134
 domain_suffix_match, 135
 dpp_c_sign_key, 137
 dpp_connector, 137
 dpp_net_access_key, 138
 eap_crypto_binding, 133
 eap_password, 133
 eap_result_ind, 133
 eap_ver, 132
 fast_prov, 137
 group_cipher, 130
 group_mgmt_cipher, 130
 identities, 137
 identity, 133
 is_pmf_required, 130
 key_mgmt, 129
 mcstCipher, 130
 mfpc, 132
 mfpr, 132
 nusers, 136
 pac_opaque_encr_key, 137
 pairwise_cipher, 130
 password, 131
 password_len, 131
 passwords, 137
 peap_label, 132
 pkc, 130
 pmk, 131
 pmk_valid, 132
 psk, 130
 psk_len, 131
 pwe_derivation, 131
 sae_groups, 131
 server_cert_data, 136
 server_cert_len, 136
 server_key_data, 136
 server_key_len, 136
 server_key_passwd, 136
 tls_cipher, 133
 transition_disable, 131
 type, 129
 ucstCipher, 130
 wpa3_sb, 132
 wpa3_sb_192, 132
wlan_nlist_mode
 wifi.h, 236
wlan_nlist_report_param, 138
 bssid, 139
 btm_mode, 139
 channels, 138
 dialog_token, 139
 dst_addr, 139
 nlist_mode, 138
 num_channels, 138

- protect, 139
- wlan_pmksa_flush
 - wlan.h, 343
- wlan_pmksa_list
 - wlan.h, 343
- wlan_ps_mode
 - wlan.h, 388
- wlan_ps_state
 - wlan.h, 389
- wlan_reassociate
 - wlan.h, 252
- wlan_register_csi_user_callback
 - wlan.h, 346
- wlan_register_fw_dump_cb
 - wlan.h, 324
- wlan_remain_on_channel
 - wlan.h, 298
- wlan_remove_all_network_profiles
 - wlan.h, 247
- wlan_remove_network
 - wlan.h, 248
- wlan_rf_trigger_frame_cfg
 - wlan.h, 318
- wlan_rrm_beacon_report_data, 139
 - bits_field, 141
 - bssid, 140
 - channel, 140
 - channel_num, 140
 - duration, 140
 - last_ind, 140
 - report_detail, 141
 - ssid, 140
 - ssid_length, 140
 - token, 140
- wlan_rrm_beacon_reporting_detail
 - wifi.h, 236
- wlan_rrm_neighbor_ap_t, 141
 - bssid, 141
 - bssidInfo, 141
 - channel, 142
 - freq, 142
 - op_class, 142
 - phy_type, 142
 - ssid, 141
- wlan_rrm_neighbor_report_t, 142
 - neighbor_ap, 142
 - neighbor_cnt, 143
- wlan_rrm_scan_cb_param, 143
 - dialog_tok, 143
 - dst_addr, 143
 - protect, 143
 - rep_data, 143
- wlan_rssi_info_t
 - wlan.h, 385
- wlan_rutxpowerlimit_t
 - wlan.h, 383
- wlan_rx_mgmt_indication
 - wlan.h, 340
- wlan_save_cloud_keep_alive_params
 - wlan.h, 356
- wlan_scan
 - wlan.h, 262
- wlan_scan_channel_list_t
 - wlan.h, 381
- wlan_scan_params_v2_t
 - wlan.h, 381
- wlan_scan_result, 144
 - ap_mfpc, 149
 - ap_mfpr, 149
 - ap_pwe, 149
 - beacon_period, 149
 - bss_transition_supported, 149
 - bssid, 145
 - channel, 145
 - dot11ac, 146
 - dot11ax, 146
 - dot11n, 145
 - dtim_period, 149
 - ft_1x, 147
 - ft_1x_sha384, 148
 - ft_psk, 148
 - ft_sae, 148
 - neighbor_report_supported, 149
 - role, 145
 - rssi, 148
 - ssid, 145
 - ssid_len, 145
 - trans_bssid, 148
 - trans_ssid, 148
 - trans_ssid_len, 148
 - type, 145
 - wep, 146
 - wmm, 146
 - wpa, 146
 - wpa2, 147
 - wpa2_entp, 147
 - wpa2_entp_sha256, 147
 - wpa2_sha256, 147
 - wpa3_1x_sha256, 147
 - wpa3_1x_sha384, 147
 - wpa3_sae, 147
 - wps, 146
 - wps_session, 146
- wlan_scan_with_opt
 - wlan.h, 263
- wlan_security_type
 - wlan.h, 391
- wlan_send_debug_htc
 - wlan.h, 333
- wlan_send_hostcmd
 - wlan.h, 332
- wlan_set_11ax_cfg
 - wlan.h, 336
- wlan_set_11ax_rutxpowerlimit
 - wlan.h, 335
- wlan_set_11ax_rutxpowerlimit_legacy

wlan.h, 335
wlan_set_11ax_tol_time
wlan.h, 334
wlan_set_11ax_tx_omi
wlan.h, 334
wlan_set_11d_state
wlan.h, 359
wlan_set_antcfg
wlan.h, 272
wlan_set_auto_arp
wlan.h, 281
wlan_set_btwt_cfg
wlan.h, 336
wlan_set_cal_data
wlan.h, 266
wlan_set_chanlist
wlan.h, 300
wlan_set_chanlist_and_txpwrlimit
wlan.h, 299
wlan_set_clocksync_cfg
wlan.h, 339
wlan_set_country_code
wlan.h, 357
wlan_set_country_ie_ignore
wlan.h, 358
wlan_set_crypto_AES_CCMP_decrypt
wlan.h, 329
wlan_set_crypto_AES_CCMP_encrypt
wlan.h, 329
wlan_set_crypto_AES_ECB_decrypt
wlan.h, 327
wlan_set_crypto_AES_ECB_encrypt
wlan.h, 326
wlan_set_crypto_AES_GCMP_decrypt
wlan.h, 331
wlan_set_crypto_AES_GCMP_encrypt
wlan.h, 330
wlan_set_crypto_AES_WRAP_decrypt
wlan.h, 328
wlan_set_crypto_AES_WRAP_encrypt
wlan.h, 327
wlan_set_crypto_RC4_decrypt
wlan.h, 325
wlan_set_crypto_RC4_encrypt
wlan.h, 325
wlan_set_ed_mac_mode
wlan.h, 264
wlan_set_entp_cert_files
wlan.h, 349
wlan_set_ext_coex_config
wlan.h, 296
wlan_set_frag
wlan.h, 311
wlan_set_htcapinfo
wlan.h, 289
wlan_set_httxcfg
wlan.h, 290
wlan_set_ieeeeps_cfg
wlan.h, 270
wlan_set_indrst_cfg
wlan.h, 365
wlan_set_ipv6_ns_mef
wlan.h, 345
wlan_set_ipv6_ns_offload
wlan.h, 281
wlan_set_mac_addr
wlan.h, 268
wlan_set_mgmt_ie
wlan.h, 295
wlan_set_network_ip_byname
wlan.h, 366
wlan_set_okc
wlan.h, 342
wlan_set_packet_filters
wlan.h, 278
wlan_set_ps_cfg
wlan.h, 355
wlan_set_reassoc_control
wlan.h, 303
wlan_set_region_code
wlan.h, 358
wlan_set_region_power_cfg
wlan.h, 299
wlan_set_rf_band
wlan.h, 314
wlan_set_rf_bandwidth
wlan.h, 315
wlan_set_rf_channel
wlan.h, 313
wlan_set_rf_otf_cal_data
wlan.h, 323
wlan_set_rf_otf_mac_addr
wlan.h, 322
wlan_set_rf_radio_mode
wlan.h, 313
wlan_set_rf_rx_antenna
wlan.h, 320
wlan_set_rf_test_mode
wlan.h, 312
wlan_set_rf_tx_antenna
wlan.h, 319
wlan_set_rf_tx_cont_mode
wlan.h, 317
wlan_set_rf_tx_frame
wlan.h, 322
wlan_set_rf_tx_power
wlan.h, 321
wlan_set_rg_power_cfg
wlan.h, 353
wlan_set_roaming
wlan.h, 269
wlan_set_rssi_low_threshold
wlan.h, 347
wlan_set_rts
wlan.h, 310
wlan_set_scan_channel_gap

wlan.h, 341
 wlan_set_scan_interval
 wlan.h, 343
 wlan_set_sta_mac_addr
 wlan.h, 268
 wlan_set_sta_mac_filter
 wlan.h, 312
 wlan_set_sta_tx_power
 wlan.h, 293
 wlan_set_turbo_mode
 wlan.h, 355
 wlan_set_twt_setup_cfg
 wlan.h, 337
 wlan_set_twt_teardown_cfg
 wlan.h, 337
 wlan_set_txpwrlimit
 wlan.h, 301
 wlan_set_txratecfg
 wlan.h, 291
 wlan_set_uap_ed_mac_mode
 wlan.h, 265
 wlan_set_uap_frag
 wlan.h, 311
 wlan_set_uap_mac_addr
 wlan.h, 268
 wlan_set_uap_max_clients
 wlan.h, 289
 wlan_set_uap_rts
 wlan.h, 311
 wlan_set_uap_turbo_mode
 wlan.h, 355
 wlan_set_wwsm_txpwrlimit
 wlan.h, 294
 wlan_show_os_mem_stat
 wlan.h, 340
 wlan_sta_ampdu_rx_disable
 wlan.h, 309
 wlan_sta_ampdu_rx_enable
 wlan.h, 309
 wlan_sta_ampdu_tx_disable
 wlan.h, 308
 wlan_sta_ampdu_tx_enable
 wlan.h, 308
 wlan_start
 wlan.h, 246
 wlan_start_ap_wps_pbc
 wlan.h, 349
 wlan_start_ap_wps_pin
 wlan.h, 348
 wlan_start_cloud_keep_alive
 wlan.h, 357
 wlan_start_network
 wlan.h, 252
 wlan_start_wps_pbc
 wlan.h, 348
 wlan_start_wps_pin
 wlan.h, 347
 wlan_stop
 wlan.h, 246
 wlan_stop_cloud_keep_alive
 wlan.h, 357
 wlan_stop_network
 wlan.h, 253
 wlan_string_dup
 wlan.h, 366
 wlan_tcp_keep_alive
 wlan.h, 275
 wlan_tcp_keep_alive_t
 wlan.h, 382
 wlan_test_mode_cli_deinit
 wlan.h, 287
 wlan_test_mode_cli_init
 wlan.h, 287
 wlan_tests.h, 395
 print_txpwrlimit, 395
 test_wlan_cfg_process, 395
 wlan_tsf_info_t
 wlan.h, 384
 wlan_twt_information
 wlan.h, 338
 wlan_twt_information_t
 wlan.h, 384
 wlan_twt_report_t
 wlan.h, 384
 wlan_twt_setup_config_t
 wlan.h, 383
 wlan_twt_teardown_config_t
 wlan.h, 384
 wlan_tx_ampdu_prot_mode
 wlan.h, 344
 wlan_txpwrlimit_t
 wlan.h, 383
 wlan_txrate_setting
 wlan.h, 385
 wlan_type
 wifi-decl.h, 181
 wlan_uap_ampdu_rx_disable
 wlan.h, 310
 wlan_uap_ampdu_rx_enable
 wlan.h, 309
 wlan_uap_ampdu_tx_disable
 wlan.h, 309
 wlan_uap_ampdu_tx_enable
 wlan.h, 309
 wlan_uap_client_disassoc_t
 wlan.h, 385
 wlan_uap_ctrl_deauth
 wlan.h, 306
 wlan_uap_disconnect_sta
 wlan.h, 366
 wlan_uap_get_bandwidth
 wlan.h, 304
 wlan_uap_get_pmfcfg
 wlan.h, 278
 wlan_uap_set_bandwidth
 wlan.h, 304

wlan_uap_set_beacon_period
wlan.h, 304

wlan_uap_set_ecsa
wlan.h, 307

wlan_uap_set_hidden_ssid
wlan.h, 306

wlan_uap_set_htcapinfo
wlan.h, 307

wlan_uap_set_httxcf
wlan.h, 308

wlan_uap_set_scan_chan_list
wlan.h, 310

wlan_unregister_csi_user_callback
wlan.h, 346

wlan_unset_rf_test_mode
wlan.h, 312

wlan_version_extended
wlan.h, 273

wlan_wakeup_event_t
wlan.h, 387

wlan_wfa_basic_cli_deinit
wlan.h, 283

wlan_wfa_basic_cli_init
wlan.h, 283

wlan_wlcmgr_send_msg
wlan.h, 283

wlan_wmm_tx_stats_dump
wlan.h, 341

wlan_wowlan_cfg_ptn_match
wlan.h, 281

wlan_wowlan_ptn_cfg_t
wlan.h, 382

wlan_wps_ap_cancel
wlan.h, 349

wlan_wps_cancel
wlan.h, 348

wlan_wps_generate_pin
wlan.h, 347

wlcm_d
wlan.h, 367

wlcm_e
wlan.h, 367

wlcm_w
wlan.h, 367

wm_bin2hex
wm_utils.h, 412

wm_dhcpd_errno
dhcp-server.h, 159

wm_frac_part_of
wm_utils.h, 415

wm_hex2bin
wm_utils.h, 412

wm_int_part_of
wm_utils.h, 420

wm_net.h, 396

 ipv6_addr_addr_to_desc, 405

 ipv6_addr_state_to_desc, 404

 ipv6_addr_type_to_desc, 405

NET_BLOCKING_OFF, 409

NET_BLOCKING_ON, 409

NET_ENOBUFS, 409

NET_ERROR, 408

NET_SUCCESS, 408

net_accept, 410

net_address_types, 411

net_alloc_client_data_id, 400

net_bind, 409

net_close, 410

net_configure_address, 402

net_configure_dns, 403

net_connect, 410

net_dhcp_hostname_set, 396

net_get_if_addr, 403

net_get_if_ip_addr, 406

net_get_if_ip_mask, 406

net_get_if_ipv6_addr, 403

net_get_if_ipv6_pref_addr, 404

net_get_if_name, 405

net_get_mlan_handle, 411

net_get_sock_error, 397

net_get_sta_handle, 400

net_get_sta_interface, 399

net_get_uap_handle, 400

net_get_uap_interface, 400

net_inet_aton, 397

net_inet_ntoa, 398

net_interface_dhcp_cleanup, 402

net_interface_dhcp_stop, 402

net_interface_down, 401

net_interface_up, 401

net_ipv4stack_init, 408

net_listen, 410

net_read, 410

net_select, 409

net_shutdown, 410

net_sock_to_interface, 398

net_socket, 409

net_socket_blocking, 396

net_stack_buffer_skip, 398

net_stat, 408

net_stop_dhcp_timer, 396

net_wlan_deinit, 399

net_wlan_init, 399

net_wlan_set_mac_address, 397

net_write, 411

wm_rand_seed
osa.h, 171

wm_strtof
wm_utils.h, 416

wm_utils.h, 411

 __WM_ALIGN__, 418

dump_ascii, 419

dump_hex, 419

dump_hex_ascii, 419

dump_json, 420

ffs, 418

- fill_sequential_pattern, 417
- get_random_sequence, 415
- NORETURN, 418
- PACK_END, 418
- PACK_START, 418
- print_ascii, 419
- random_hdlr_t, 420
- random_initialize_seed, 414
- random_register_handler, 413
- random_register_seed_handler, 413
- random_unregister_handler, 413
- random_unregister_seed_handler, 414
- sample_initialise_random_seed, 415
- soft_crc32, 416
- strdup, 415
- verify_sequential_pattern, 417
- WARN_UNUSED_RET, 418
- WM_MASK, 418
- wm_bin2hex, 412
- wm_frac_part_of, 415
- wm_hex2bin, 412
- wm_int_part_of, 420
- wm_strtof, 416
- wmpanic, 412
- wm_wlan_errno
 - wlan.h, 385
- wmm
 - wlan_scan_result, 146
- wmm_ie_present
 - wifi_scan_result2, 98
- wmpanic
 - wm_utils.h, 412
- wnm_sleep_result_t, 150
 - action, 150
 - result, 150
- wpa
 - _SecurityMode_t, 12
 - wlan_scan_result, 146
- wpa2
 - _SecurityMode_t, 13
 - wlan_scan_result, 147
- wpa2_entp
 - _SecurityMode_t, 13
 - wlan_scan_result, 147
- wpa2_entp_IE_exist
 - wifi_scan_result2, 98
- wpa2_entp_sha256
 - _SecurityMode_t, 13
 - wlan_scan_result, 147
- wpa2_sha256
 - _SecurityMode_t, 13
 - wlan_scan_result, 147
- wpa3_1x_sha256
 - _SecurityMode_t, 14
 - wlan_scan_result, 147
- wpa3_1x_sha384
 - _SecurityMode_t, 14
 - wlan_scan_result, 147
- wpa3_sae
 - _SecurityMode_t, 13
 - wlan_scan_result, 147
- wpa3_sb
 - wlan_network_security, 132
- wpa3_sb_192
 - wlan_network_security, 132
- wpa_mcstCipher
 - wifi_scan_result2, 96
- wpa_ucstCipher
 - wifi_scan_result2, 96
- wpaNone
 - _SecurityMode_t, 13
- wps
 - wlan_scan_result, 146
- wps_IE_exist
 - wifi_scan_result2, 98
- wps_network
 - wlan_network, 123
- wps_session
 - wifi_scan_result2, 98
 - wlan_scan_result, 146
- wrapper_clear_media_connected_event
 - wifi.h, 206
- wrapper_wifi_assoc
 - wifi.h, 208
- wrapper_wlan_11d_clear_parsedtable
 - wifi.h, 206
- wrapper_wlan_11d_enable
 - wifi.h, 210
- wrapper_wlan_11d_support_is_enabled
 - wifi.h, 205
- wrapper_wlan_cmd_11n_addba_rspgen
 - wifi.h, 210
- wrapper_wlan_cmd_11n_ba_stream_timeout
 - wifi.h, 209
- wrapper_wlan_cmd_11n_delba_rspgen
 - wifi.h, 210
- wrapper_wlan_cmd_get_hw_spec
 - wifi.h, 205
- wrapper_wlan_ecsa_enable
 - wifi.h, 210
- wrapper_wlan_sta_ampdu_enable
 - wifi.h, 211
- wrapper_wlan_uap_11d_enable
 - wifi.h, 210
- wrapper_wlan_uap_ampdu_enable
 - wifi.h, 211
- write_index
 - csi_local_buff_statu, 17