

ESP8266_RTOS_SDK
v2.0.0

Generated by Doxygen 1.8.10

Fri Mar 2 2018 11:04:05

Contents

1	ESP8266_RTOS_SDK	1
2	Module Index	3
2.1	Modules	3
3	Data Structure Index	5
3.1	Data Structures	5
4	Module Documentation	7
4.1	WiFi Related APIs	7
4.1.1	Detailed Description	7
4.2	AirKiss APIs	8
4.2.1	Detailed Description	8
4.2.2	Enumeration Type Documentation	8
4.2.2.1	airkiss_lan_ret_t	8
4.2.3	Function Documentation	9
4.2.3.1	airkiss_lan_pack(airkiss_lan_cmdid_t ak_lan_cmdid, void *appid, void *deviceid, void *_datain, unsigned short inlength, void *_dataout, unsigned short *outlength, const airkiss_config_t *config)	9
4.2.3.2	airkiss_lan_rcv(const void *body, unsigned short length, const airkiss_config_t *config)	10
4.2.3.3	airkiss_version(void)	10
4.3	Misc APIs	11
4.3.1	Detailed Description	11
4.3.2	Macro Definition Documentation	11
4.3.2.1	IP2STR	11
4.3.3	Enumeration Type Documentation	11
4.3.3.1	dhcp_status	11
4.3.3.2	dhcps_offer_option	12
4.3.4	Function Documentation	12
4.3.4.1	os_delay_us(uint16 us)	12
4.3.4.2	os_install_putc1(void(*p)(char c))	12
4.3.4.3	os_putc(char c)	12

4.4	SoftAP APIs	13
4.4.1	Detailed Description	13
4.4.2	Function Documentation	14
4.4.2.1	wifi_softap_dhcps_start(void)	14
4.4.2.2	wifi_softap_dhcps_status(void)	14
4.4.2.3	wifi_softap_dhcps_stop(void)	14
4.4.2.4	wifi_softap_free_station_info(void)	14
4.4.2.5	wifi_softap_get_config(struct softap_config *config)	15
4.4.2.6	wifi_softap_get_config_default(struct softap_config *config)	15
4.4.2.7	wifi_softap_get_dhcps_lease(struct dhcps_lease *please)	15
4.4.2.8	wifi_softap_get_dhcps_lease_time(void)	15
4.4.2.9	wifi_softap_get_station_info(void)	16
4.4.2.10	wifi_softap_get_station_num(void)	16
4.4.2.11	wifi_softap_reset_dhcps_lease_time(void)	16
4.4.2.12	wifi_softap_set_config(struct softap_config *config)	17
4.4.2.13	wifi_softap_set_config_current(struct softap_config *config)	17
4.4.2.14	wifi_softap_set_dhcps_lease(struct dhcps_lease *please)	17
4.4.2.15	wifi_softap_set_dhcps_lease_time(uint32 minute)	18
4.4.2.16	wifi_softap_set_dhcps_offer_option(uint8 level, void *optarg)	18
4.5	Spiffs APIs	19
4.5.1	Detailed Description	19
4.5.2	Function Documentation	19
4.5.2.1	esp_spiffs_deinit(uint8 format)	19
4.5.2.2	esp_spiffs_init(struct esp_spiffs_config *config)	19
4.6	SSC APIs	20
4.6.1	Detailed Description	20
4.6.2	Function Documentation	20
4.6.2.1	ssc_attach(SscBaudRate bandrate)	20
4.6.2.2	ssc_param_len(void)	20
4.6.2.3	ssc_param_str(void)	20
4.6.2.4	ssc_parse_param(char *pLine, char *argv[])	21
4.6.2.5	ssc_register(ssc_cmd_t *cmdset, uint8 cmdnum, void(*help)(void))	21
4.7	Station APIs	22
4.7.1	Detailed Description	23
4.7.2	Typedef Documentation	23
4.7.2.1	scan_done_cb_t	23
4.7.3	Enumeration Type Documentation	24
4.7.3.1	CIPHER_TYPE	24
4.7.3.2	STATION_STATUS	24
4.7.3.3	wifi_scan_type_t	24

4.7.4	Function Documentation	24
4.7.4.1	wifi_station_ap_change(uint8 current_ap_id)	24
4.7.4.2	wifi_station_ap_number_set(uint8 ap_number)	24
4.7.4.3	wifi_station_connect(void)	25
4.7.4.4	wifi_station_dhcpc_start(void)	25
4.7.4.5	wifi_station_dhcpc_status(void)	25
4.7.4.6	wifi_station_dhcpc_stop(void)	26
4.7.4.7	wifi_station_disconnect(void)	26
4.7.4.8	wifi_station_get_ap_info(struct station_config config[])	26
4.7.4.9	wifi_station_get_auto_connect(void)	27
4.7.4.10	wifi_station_get_config(struct station_config *config)	27
4.7.4.11	wifi_station_get_config_default(struct station_config *config)	27
4.7.4.12	wifi_station_get_connect_status(void)	27
4.7.4.13	wifi_station_get_current_ap_id(void)	28
4.7.4.14	wifi_station_get_hostname(void)	28
4.7.4.15	wifi_station_get_reconnect_policy(void)	28
4.7.4.16	wifi_station_get_rssi(void)	28
4.7.4.17	wifi_station_scan(struct scan_config *config, scan_done_cb_t cb)	29
4.7.4.18	wifi_station_set_auto_connect(bool set)	29
4.7.4.19	wifi_station_set_config(struct station_config *config)	29
4.7.4.20	wifi_station_set_config_current(struct station_config *config)	30
4.7.4.21	wifi_station_set_hostname(char *name)	30
4.7.4.22	wifi_station_set_reconnect_policy(bool set)	30
4.8	System APIs	32
4.8.1	Detailed Description	33
4.8.2	Enumeration Type Documentation	33
4.8.2.1	rst_reason	33
4.8.3	Function Documentation	33
4.8.3.1	system_adc_read(void)	33
4.8.3.2	system_deep_sleep(uint32 time_in_us)	34
4.8.3.3	system_deep_sleep_set_option(uint8 option)	34
4.8.3.4	system_get_chip_id(void)	34
4.8.3.5	system_get_free_heap_size(void)	35
4.8.3.6	system_get_rst_info(void)	35
4.8.3.7	system_get_rtc_time(void)	35
4.8.3.8	system_get_sdk_version(void)	36
4.8.3.9	system_get_time(void)	37
4.8.3.10	system_get_vdd33(void)	37
4.8.3.11	system_param_load(uint16 start_sec, uint16 offset, void *param, uint16 len)	37
4.8.3.12	system_param_save_with_protect(uint16 start_sec, void *param, uint16 len)	38

4.8.3.13	system_phy_set_max_tpw(uint8 max_tpw)	38
4.8.3.14	system_phy_set_rfoption(uint8 option)	38
4.8.3.15	system_phy_set_tpw_via_vdd33(uint16 vdd33)	39
4.8.3.16	system_print_meminfo(void)	39
4.8.3.17	system_restart(void)	39
4.8.3.18	system_restore(void)	40
4.8.3.19	system_rtc_clock_calib_proc(void)	40
4.8.3.20	system_rtc_mem_read(uint8 src, void *dst, uint16 n)	40
4.8.3.21	system_rtc_mem_write(uint8 dst, const void *src, uint16 n)	41
4.8.3.22	system_uart_de_swap(void)	41
4.8.3.23	system_uart_swap(void)	41
4.9	Boot APIs	43
4.9.1	Detailed Description	43
4.9.2	Macro Definition Documentation	43
4.9.2.1	SYS_BOOT_ENHANCE_MODE	43
4.9.2.2	SYS_BOOT_NORMAL_BIN	44
4.9.2.3	SYS_BOOT_NORMAL_MODE	44
4.9.2.4	SYS_BOOT_TEST_BIN	44
4.9.3	Enumeration Type Documentation	44
4.9.3.1	flash_size_map	44
4.9.4	Function Documentation	44
4.9.4.1	system_get_boot_mode(void)	44
4.9.4.2	system_get_boot_version(void)	44
4.9.4.3	system_get_cpu_freq(void)	45
4.9.4.4	system_get_flash_size_map(void)	45
4.9.4.5	system_get_userbin_addr(void)	45
4.9.4.6	system_restart_enhance(uint8 bin_type, uint32 bin_addr)	45
4.9.4.7	system_update_cpu_freq(uint8 freq)	46
4.10	Software timer APIs	47
4.10.1	Detailed Description	47
4.10.2	Function Documentation	47
4.10.2.1	os_timer_arm(os_timer_t *ptimer, uint32 msec, bool repeat_flag)	47
4.10.2.2	os_timer_disarm(os_timer_t *ptimer)	47
4.10.2.3	os_timer_setfn(os_timer_t *ptimer, os_timer_func_t *pfunction, void *parg)	48
4.11	Common APIs	49
4.11.1	Detailed Description	51
4.11.2	Typedef Documentation	51
4.11.2.1	freedom_outside_cb_t	51
4.11.2.2	rfid_locp_cb_t	51
4.11.2.3	wifi_event_handler_cb_t	51

4.11.3 Enumeration Type Documentation	52
4.11.3.1 AUTH_MODE	52
4.11.3.2 SYSTEM_EVENT	52
4.11.3.3 WIFI_COUNTRY_POLICY	52
4.11.3.4 WIFI_INTERFACE	52
4.11.3.5 WIFI_MODE	53
4.11.3.6 WIFI_PHY_MODE	53
4.11.4 Function Documentation	53
4.11.4.1 wifi_get_ip_info(WIFI_INTERFACE if_index, struct ip_info *info)	53
4.11.4.2 wifi_get_macaddr(WIFI_INTERFACE if_index, uint8 *macaddr)	53
4.11.4.3 wifi_get_opmode(void)	53
4.11.4.4 wifi_get_opmode_default(void)	54
4.11.4.5 wifi_get_phy_mode(void)	54
4.11.4.6 wifi_get_sleep_type(void)	54
4.11.4.7 wifi_register_rfid_locp_rcv_cb(rfid_locp_cb_t cb)	54
4.11.4.8 wifi_register_send_pkt_freedom_cb(freedom_outside_cb_t cb)	55
4.11.4.9 wifi_rfid_locp_rcv_close(void)	55
4.11.4.10 wifi_rfid_locp_rcv_open(void)	55
4.11.4.11 wifi_send_pkt_freedom(uint8 *buf, uint16 len, bool sys_seq)	56
4.11.4.12 wifi_set_event_handler_cb(wifi_event_handler_cb_t cb)	56
4.11.4.13 wifi_set_ip_info(WIFI_INTERFACE if_index, struct ip_info *info)	56
4.11.4.14 wifi_set_macaddr(WIFI_INTERFACE if_index, uint8 *macaddr)	57
4.11.4.15 wifi_set_opmode(WIFI_MODE opmode)	57
4.11.4.16 wifi_set_opmode_current(WIFI_MODE opmode)	58
4.11.4.17 wifi_set_phy_mode(WIFI_PHY_MODE mode)	58
4.11.4.18 wifi_set_sleep_type(sleep_type type)	58
4.11.4.19 wifi_status_led_install(uint8 gpio_id, uint32 gpio_name, uint8 gpio_func)	59
4.11.4.20 wifi_status_led_uninstall(void)	60
4.11.4.21 wifi_unregister_rfid_locp_rcv_cb(void)	60
4.11.4.22 wifi_unregister_send_pkt_freedom_cb(void)	60
4.12 Force Sleep APIs	61
4.12.1 Detailed Description	61
4.12.2 Function Documentation	61
4.12.2.1 wifi_fpm_close(void)	61
4.12.2.2 wifi_fpm_do_sleep(uint32 sleep_time_in_us)	61
4.12.2.3 wifi_fpm_do_wakeup(void)	62
4.12.2.4 wifi_fpm_get_sleep_type(void)	62
4.12.2.5 wifi_fpm_open(void)	62
4.12.2.6 wifi_fpm_set_sleep_type(sleep_type type)	63
4.12.2.7 wifi_fpm_set_wakeup_cb(fpm_wakeup_cb cb)	63

4.13	Rate Control APIs	64
4.13.1	Detailed Description	65
4.13.2	Function Documentation	65
4.13.2.1	wifi_get_user_fixed_rate(uint8 *enable_mask, uint8 *rate)	65
4.13.2.2	wifi_get_user_limit_rate_mask(void)	65
4.13.2.3	wifi_set_user_fixed_rate(uint8 enable_mask, uint8 rate)	65
4.13.2.4	wifi_set_user_limit_rate_mask(uint8 enable_mask)	66
4.13.2.5	wifi_set_user_rate_limit(uint8 mode, uint8 ifidx, uint8 max, uint8 min)	66
4.13.2.6	wifi_set_user_sup_rate(uint8 min, uint8 max)	67
4.14	Vendor IE APIs	68
4.14.1	Detailed Description	68
4.14.2	Typedef Documentation	68
4.14.2.1	vendor_ie_rcv_cb_t	68
4.14.3	Enumeration Type Documentation	68
4.14.3.1	vendor_ie_type	68
4.14.4	Function Documentation	69
4.14.4.1	wifi_register_vnd_ie_rcv_cb(vendor_ie_rcv_cb_t cb)	69
4.14.4.2	wifi_set_vnd_ie(bool enable, vendor_ie_type type, uint8_t idx, uint8_t *vnd_ie)	69
4.14.4.3	wifi_unregister_vnd_ie_rcv_cb(void)	69
4.15	User IE APIs	71
4.15.1	Detailed Description	71
4.15.2	Typedef Documentation	71
4.15.2.1	user_ie_manufacturer_rcv_cb_t	71
4.15.3	Function Documentation	72
4.15.3.1	wifi_register_user_ie_manufacturer_rcv_cb(user_ie_manufacturer_rcv_cb_t cb)	72
4.15.3.2	wifi_set_user_ie(bool enable, uint8 *m_oui, user_ie_type type, uint8 *user_ie, uint8 len)	73
4.15.3.3	wifi_unregister_user_ie_manufacturer_rcv_cb(void)	73
4.16	Sniffer APIs	74
4.16.1	Detailed Description	74
4.16.2	Typedef Documentation	74
4.16.2.1	wifi_promiscuous_cb_t	74
4.16.3	Function Documentation	74
4.16.3.1	wifi_get_channel(void)	74
4.16.3.2	wifi_get_country(wifi_country_t *country)	75
4.16.3.3	wifi_promiscuous_enable(uint8 promiscuous)	75
4.16.3.4	wifi_promiscuous_set_mac(const uint8_t *address)	75
4.16.3.5	wifi_set_channel(uint8 channel)	76
4.16.3.6	wifi_set_country(wifi_country_t *country)	76
4.16.3.7	wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb)	76

4.17 WPS APIs	78
4.17.1 Detailed Description	78
4.17.2 Typedef Documentation	78
4.17.2.1 wps_st_cb_t	78
4.17.3 Enumeration Type Documentation	79
4.17.3.1 wps_cb_status	79
4.17.4 Function Documentation	79
4.17.4.1 wifi_set_wps_cb(wps_st_cb_t cb)	79
4.17.4.2 wifi_wps_disable(void)	79
4.17.4.3 wifi_wps_enable(WPS_TYPE_t wps_type)	79
4.17.4.4 wifi_wps_start(void)	80
4.18 Network Espconn APIs	81
4.18.1 Detailed Description	83
4.18.2 Macro Definition Documentation	83
4.18.2.1 ESPCONN_ABRT	83
4.18.2.2 ESPCONN_ARG	83
4.18.2.3 ESPCONN_CLSD	83
4.18.2.4 ESPCONN_CONN	83
4.18.2.5 ESPCONN_IF	83
4.18.2.6 ESPCONN_INPROGRESS	83
4.18.2.7 ESPCONN_ISCONN	83
4.18.2.8 ESPCONN_MAXNUM	84
4.18.2.9 ESPCONN_MEM	84
4.18.2.10 ESPCONN_OK	84
4.18.2.11 ESPCONN_RST	84
4.18.2.12 ESPCONN_RTE	84
4.18.2.13 ESPCONN_TIMEOUT	84
4.18.3 Typedef Documentation	84
4.18.3.1 dns_found_callback	84
4.18.3.2 espconn_connect_callback	84
4.18.3.3 espconn_reconnect_callback	85
4.18.3.4 espconn_recv_callback	85
4.18.4 Enumeration Type Documentation	85
4.18.4.1 espconn_level	85
4.18.4.2 espconn_option	86
4.18.4.3 espconn_state	86
4.18.4.4 espconn_type	86
4.18.5 Function Documentation	86
4.18.5.1 espconn_accept(struct espconn *espconn)	86
4.18.5.2 espconn_clear_opt(struct espconn *espconn, uint8 opt)	87

4.18.5.3	espconn_connect(struct espconn *espconn)	87
4.18.5.4	espconn_create(struct espconn *espconn)	87
4.18.5.5	espconn_delete(struct espconn *espconn)	88
4.18.5.6	espconn_disconnect(struct espconn *espconn)	88
4.18.5.7	espconn_dns_setserver(char numdns, ip_addr_t *dnserver)	88
4.18.5.8	espconn_get_connection_info(struct espconn *pespconn, remot_info **pcon_info, uint8 typeflags)	89
4.18.5.9	espconn_get_keepalive(struct espconn *espconn, uint8 level, void *optarg)	89
4.18.5.10	espconn_gethostbyname(struct espconn *pespconn, const char *hostname, ip_addr_t *addr, dns_found_callback found)	89
4.18.5.11	espconn_igmp_join(ip_addr_t *host_ip, ip_addr_t *multicast_ip)	90
4.18.5.12	espconn_igmp_leave(ip_addr_t *host_ip, ip_addr_t *multicast_ip)	90
4.18.5.13	espconn_init(void)	91
4.18.5.14	espconn_port(void)	91
4.18.5.15	espconn_recv_hold(struct espconn *pespconn)	91
4.18.5.16	espconn_recv_unhold(struct espconn *pespconn)	91
4.18.5.17	espconn_regist_connectcb(struct espconn *espconn, espconn_connect_callback connect_cb)	92
4.18.5.18	espconn_regist_disconcb(struct espconn *espconn, espconn_connect_callback discon_cb)	92
4.18.5.19	espconn_regist_reconcb(struct espconn *espconn, espconn_reconnect_callback recon_cb)	92
4.18.5.20	espconn_regist_recvcb(struct espconn *espconn, espconn_recv_callback recv_cb)	93
4.18.5.21	espconn_regist_sentcb(struct espconn *espconn, espconn_sent_callback sent_cb)	93
4.18.5.22	espconn_regist_time(struct espconn *espconn, uint32 interval, uint8 type_flag)	93
4.18.5.23	espconn_regist_write_finish(struct espconn *espconn, espconn_connect_callback write_finish_fn)	94
4.18.5.24	espconn_send(struct espconn *espconn, uint8 *psent, uint16 length)	94
4.18.5.25	espconn_sendto(struct espconn *espconn, uint8 *psent, uint16 length)	95
4.18.5.26	espconn_sent(struct espconn *espconn, uint8 *psent, uint16 length)	95
4.18.5.27	espconn_set_keepalive(struct espconn *espconn, uint8 level, void *optarg)	96
4.18.5.28	espconn_set_opt(struct espconn *espconn, uint8 opt)	96
4.18.5.29	espconn_tcp_get_max_con(void)	97
4.18.5.30	espconn_tcp_get_max_con_allow(struct espconn *espconn)	97
4.18.5.31	espconn_tcp_set_max_con(uint8 num)	97
4.18.5.32	espconn_tcp_set_max_con_allow(struct espconn *espconn, uint8 num)	98
4.19	ESP-NOW APIs	99
4.19.1	Detailed Description	100
4.19.2	Typedef Documentation	100
4.19.2.1	esp_now_recv_cb_t	100

4.19.2.2	esp_now_send_cb_t	100
4.19.3	Function Documentation	102
4.19.3.1	esp_now_add_peer(uint8 *mac_addr, uint8 role, uint8 channel, uint8 *key, uint8 key_len)	102
4.19.3.2	esp_now_deinit(void)	102
4.19.3.3	esp_now_del_peer(uint8 *mac_addr)	102
4.19.3.4	esp_now_fetch_peer(bool restart)	103
4.19.3.5	esp_now_get_cnt_info(uint8 *all_cnt, uint8 *encrypt_cnt)	103
4.19.3.6	esp_now_get_peer_channel(uint8 *mac_addr)	103
4.19.3.7	esp_now_get_peer_key(uint8 *mac_addr, uint8 *key, uint8 *key_len)	103
4.19.3.8	esp_now_get_peer_role(uint8 *mac_addr)	104
4.19.3.9	esp_now_get_self_role(void)	104
4.19.3.10	esp_now_init(void)	104
4.19.3.11	esp_now_is_peer_exist(uint8 *mac_addr)	104
4.19.3.12	esp_now_register_rcv_cb(esp_now_rcv_cb_t cb)	105
4.19.3.13	esp_now_register_send_cb(esp_now_send_cb_t cb)	105
4.19.3.14	esp_now_send(uint8 *da, uint8 *data, uint8 len)	105
4.19.3.15	esp_now_set_kok(uint8 *key, uint8 len)	106
4.19.3.16	esp_now_set_peer_channel(uint8 *mac_addr, uint8 channel)	107
4.19.3.17	esp_now_set_peer_key(uint8 *mac_addr, uint8 *key, uint8 key_len)	107
4.19.3.18	esp_now_set_peer_role(uint8 *mac_addr, uint8 role)	107
4.19.3.19	esp_now_set_self_role(uint8 role)	108
4.19.3.20	esp_now_unregister_rcv_cb(void)	108
4.19.3.21	esp_now_unregister_send_cb(void)	108
4.20	Driver APIs	109
4.20.1	Detailed Description	109
4.21	PWM Driver APIs	110
4.21.1	Detailed Description	110
4.21.2	Function Documentation	110
4.21.2.1	pwm_get_duty(uint8 channel)	110
4.21.2.2	pwm_get_period(void)	110
4.21.2.3	pwm_init(uint32 period, uint32 *duty, uint32 pwm_channel_num, uint32(*pin_↔ info_list)[3])	111
4.21.2.4	pwm_set_duty(uint32 duty, uint8 channel)	111
4.21.2.5	pwm_set_period(uint32 period)	111
4.21.2.6	pwm_start(void)	112
4.22	Smartconfig APIs	113
4.22.1	Detailed Description	113
4.22.2	Typedef Documentation	113
4.22.2.1	sc_callback_t	113

4.22.3	Enumeration Type Documentation	114
4.22.3.1	sc_status	114
4.22.3.2	sc_type	114
4.22.4	Function Documentation	114
4.22.4.1	esptouch_set_timeout(uint8 time_s)	114
4.22.4.2	smartconfig_get_version(void)	115
4.22.4.3	smartconfig_set_type(sc_type type)	115
4.22.4.4	smartconfig_start(sc_callback_t cb,...)	115
4.22.4.5	smartconfig_stop(void)	116
4.23	SPI Driver APIs	117
4.23.1	Detailed Description	117
4.23.2	Macro Definition Documentation	117
4.23.2.1	SPI_FLASH_SEC_SIZE	117
4.23.3	Typedef Documentation	118
4.23.3.1	user_spi_flash_read	118
4.23.4	Enumeration Type Documentation	118
4.23.4.1	SpiFlashOpResult	118
4.23.5	Function Documentation	118
4.23.5.1	spi_flash_erase_sector(uint16 sec)	118
4.23.5.2	spi_flash_get_id(void)	118
4.23.5.3	spi_flash_read(uint32 src_addr, uint32 *des_addr, uint32 size)	119
4.23.5.4	spi_flash_read_status(uint32 *status)	120
4.23.5.5	spi_flash_set_read_func(user_spi_flash_read read)	120
4.23.5.6	spi_flash_write(uint32 des_addr, uint32 *src_addr, uint32 size)	120
4.23.5.7	spi_flash_write_status(uint32 status_value)	120
4.24	Upgrade APIs	122
4.24.1	Detailed Description	122
4.24.2	Macro Definition Documentation	123
4.24.2.1	SPI_FLASH_SEC_SIZE	123
4.24.2.2	UPGRADE_FLAG_FINISH	123
4.24.2.3	UPGRADE_FLAG_IDLE	123
4.24.2.4	UPGRADE_FLAG_START	123
4.24.2.5	UPGRADE_FW_BIN1	123
4.24.2.6	UPGRADE_FW_BIN2	123
4.24.2.7	USER_BIN1	123
4.24.2.8	USER_BIN2	123
4.24.3	Typedef Documentation	123
4.24.3.1	upgrade_states_check_callback	123
4.24.4	Function Documentation	124
4.24.4.1	system_upgrade(uint8 *data, uint32 len)	124

4.24.4.2	system_upgrade_deinit()	125
4.24.4.3	system_upgrade_flag_check()	125
4.24.4.4	system_upgrade_flag_set(uint8 flag)	125
4.24.4.5	system_upgrade_init()	126
4.24.4.6	system_upgrade_reboot(void)	127
4.24.4.7	system_upgrade_start(struct upgrade_server_info *server)	127
4.24.4.8	system_upgrade_userbin_check(void)	127
5	Data Structure Documentation	129
5.1	_esp_event Struct Reference	129
5.1.1	Field Documentation	129
5.1.1.1	event_id	129
5.1.1.2	event_info	129
5.2	_esp_tcp Struct Reference	129
5.2.1	Field Documentation	130
5.2.1.1	connect_callback	130
5.2.1.2	disconnect_callback	130
5.2.1.3	local_ip	130
5.2.1.4	local_port	130
5.2.1.5	reconnect_callback	130
5.2.1.6	remote_ip	130
5.2.1.7	remote_port	130
5.2.1.8	write_finish_fn	130
5.3	_esp_udp Struct Reference	130
5.3.1	Field Documentation	131
5.3.1.1	local_ip	131
5.3.1.2	local_port	131
5.3.1.3	remote_ip	131
5.3.1.4	remote_port	131
5.4	_os_timer_t Struct Reference	131
5.5	_remot_info Struct Reference	131
5.5.1	Field Documentation	131
5.5.1.1	remote_ip	131
5.5.1.2	remote_port	132
5.5.1.3	state	132
5.6	airkiss_config_t Struct Reference	132
5.7	bss_info Struct Reference	132
5.7.1	Member Function Documentation	133
5.7.1.1	STAILQ_ENTRY(bss_info) next	133
5.7.2	Field Documentation	133

5.7.2.1	authmode	133
5.7.2.2	bssid	133
5.7.2.3	channel	133
5.7.2.4	freq_offset	133
5.7.2.5	group_cipher	133
5.7.2.6	is_hidden	133
5.7.2.7	pairwise_cipher	133
5.7.2.8	phy_11b	133
5.7.2.9	phy_11g	133
5.7.2.10	phy_11n	133
5.7.2.11	reserved	134
5.7.2.12	rsi	134
5.7.2.13	ssid	134
5.7.2.14	ssid_len	134
5.7.2.15	wps	134
5.8	cmd_s Struct Reference	134
5.9	dhcps_lease Struct Reference	134
5.9.1	Field Documentation	134
5.9.1.1	enable	134
5.9.1.2	end_ip	135
5.9.1.3	start_ip	135
5.10	esp_spiffs_config Struct Reference	135
5.10.1	Field Documentation	135
5.10.1.1	cache_buf_size	135
5.10.1.2	fd_buf_size	135
5.10.1.3	log_block_size	135
5.10.1.4	log_page_size	135
5.10.1.5	phys_addr	135
5.10.1.6	phys_erase_block	135
5.10.1.7	phys_size	136
5.11	espconn Struct Reference	136
5.11.1	Detailed Description	136
5.11.2	Field Documentation	136
5.11.2.1	link_cnt	136
5.11.2.2	recv_callback	136
5.11.2.3	reserve	136
5.11.2.4	sent_callback	136
5.11.2.5	state	136
5.11.2.6	type	137
5.12	Event_Info_u Union Reference	137

5.12.1	Field Documentation	137
5.12.1.1	ap_probereqrecved	137
5.12.1.2	auth_change	137
5.12.1.3	connected	137
5.12.1.4	disconnected	137
5.12.1.5	got_ip	137
5.12.1.6	scan_done	137
5.12.1.7	sta_connected	137
5.12.1.8	sta_disconnected	138
5.13	Event_SoftAPMode_ProbeReqRecved_t Struct Reference	138
5.13.1	Field Documentation	138
5.13.1.1	mac	138
5.13.1.2	rsi	138
5.14	Event_SoftAPMode_StaConnected_t Struct Reference	138
5.14.1	Field Documentation	138
5.14.1.1	aid	138
5.14.1.2	mac	138
5.15	Event_SoftAPMode_StaDisconnected_t Struct Reference	139
5.15.1	Field Documentation	139
5.15.1.1	aid	139
5.15.1.2	mac	139
5.16	Event_StaMode_AuthMode_Change_t Struct Reference	139
5.16.1	Field Documentation	139
5.16.1.1	new_mode	139
5.16.1.2	old_mode	139
5.17	Event_StaMode_Connected_t Struct Reference	139
5.17.1	Field Documentation	140
5.17.1.1	bssid	140
5.17.1.2	channel	140
5.17.1.3	ssid	140
5.17.1.4	ssid_len	140
5.18	Event_StaMode_Disconnected_t Struct Reference	140
5.18.1	Field Documentation	140
5.18.1.1	bssid	140
5.18.1.2	reason	140
5.18.1.3	ssid	140
5.18.1.4	ssid_len	140
5.19	Event_StaMode_Got_IP_t Struct Reference	141
5.19.1	Field Documentation	141
5.19.1.1	gw	141

5.19.1.2	ip	141
5.19.1.3	mask	141
5.20	Event_StaMode_ScanDone_t Struct Reference	141
5.20.1	Field Documentation	141
5.20.1.1	bss	141
5.20.1.2	status	141
5.21	ip_info Struct Reference	141
5.21.1	Field Documentation	142
5.21.1.1	gw	142
5.21.1.2	ip	142
5.21.1.3	netmask	142
5.22	pwm_param Struct Reference	142
5.22.1	Field Documentation	142
5.22.1.1	duty	142
5.22.1.2	freq	142
5.22.1.3	period	142
5.23	rst_info Struct Reference	143
5.23.1	Field Documentation	143
5.23.1.1	reason	143
5.24	scan_config Struct Reference	143
5.24.1	Field Documentation	143
5.24.1.1	bssid	143
5.24.1.2	channel	143
5.24.1.3	scan_time	143
5.24.1.4	scan_type	143
5.24.1.5	show_hidden	144
5.24.1.6	ssid	144
5.25	softap_config Struct Reference	144
5.25.1	Field Documentation	144
5.25.1.1	authmode	144
5.25.1.2	beacon_interval	144
5.25.1.3	channel	144
5.25.1.4	max_connection	144
5.25.1.5	password	144
5.25.1.6	ssid	144
5.25.1.7	ssid_hidden	145
5.25.1.8	ssid_len	145
5.26	SpiFlashChip Struct Reference	145
5.27	station_config Struct Reference	145
5.27.1	Field Documentation	145

5.27.1.1	bssid	145
5.27.1.2	bssid_set	145
5.27.1.3	password	145
5.27.1.4	ssid	146
5.28	station_info Struct Reference	146
5.28.1	Member Function Documentation	146
5.28.1.1	STAILQ_ENTRY(station_info) next	146
5.28.2	Field Documentation	146
5.28.2.1	bssid	146
5.28.2.2	ip	146
5.29	upgrade_server_info Struct Reference	146
5.29.1	Field Documentation	147
5.29.1.1	check_cb	147
5.29.1.2	check_times	147
5.29.1.3	pre_version	147
5.29.1.4	sockaddrin	147
5.29.1.5	upgrade_flag	147
5.29.1.6	upgrade_version	147
5.29.1.7	url	147
5.30	wifi_active_scan_time_t Struct Reference	147
5.30.1	Detailed Description	147
5.30.2	Field Documentation	148
5.30.2.1	max	148
5.30.2.2	min	148
5.31	wifi_country_t Struct Reference	148
5.31.1	Field Documentation	148
5.31.1.1	cc	148
5.31.1.2	nchan	148
5.31.1.3	policy	148
5.31.1.4	schan	148
5.32	wifi_scan_time_t Union Reference	148
5.32.1	Detailed Description	149
5.32.2	Field Documentation	149
5.32.2.1	active	149
5.32.2.2	passive	149
Index		151

Chapter 1

ESP8266_RTOS_SDK

- Misc APIs : misc APIs
- WiFi APIs : WiFi related APIs
 - SoftAP APIs : ESP8266 Soft-AP APIs
 - Station APIs : ESP8266 station APIs
 - Common APIs : WiFi common APIs
 - Force Sleep APIs : WiFi Force Sleep APIs
 - Rate Control APIs : WiFi Rate Control APIs
 - User IE APIs : WiFi User IE APIs
 - Sniffer APIs : WiFi sniffer APIs
 - WPS APIs : WiFi WPS APIs
 - Smartconfig APIs : SmartConfig APIs
 - AirKiss APIs : AirKiss APIs
- Spiffs APIs : Spiffs APIs
- SSC APIs : Simple Serial Command APIs
- System APIs : System APIs
 - Boot APIs : Boot mode APIs
 - Upgrade APIs : Firmware upgrade (FOTA) APIs
- Software timer APIs : Software timer APIs
- Network Espconn APIs : Network espconn APIs
- ESP-NOW APIs : ESP-NOW APIs
- Driver APIs : Driver APIs
 - PWM Driver APIs : PWM driver APIs
 - UART Driver APIs : UART driver APIs
 - GPIO Driver APIs : GPIO driver APIs
 - SPI Driver APIs : SPI Flash APIs
 - Hardware timer APIs : Hardware timer APIs

void user_init(void) is the entrance function of the application.

Attention

1. It is recommended that users set the timer to the periodic mode for periodic checks.
 - (1). In freeRTOS timer or os_timer, do not delay by while(1) or in the manner that will block the thread.
 - (2). The timer callback should not occupy CPU more than 15ms.
 - (3). os_timer_t should not define a local variable, it has to be global variable or memory got by malloc.
2. Since esp_iot_rtos_sdk_v1.0.4, functions are stored in CACHE by default, need not be added ICACHE↔_FLASH_ATTR any more. The interrupt functions can also be stored in CACHE. If users want to store some frequently called functions in RAM, please add IRAM_ATTR before functions' name.
3. Network programming use socket, please do not bind to the same port.
 - (1). If users want to create 3 or more than 3 TCP connections, please add "TCP_WND = 2 x TCP_MSS;" in "user_init".
4. Priority of the RTOS SDK is 15. xTaskCreate is an interface of freeRTOS. For details of the freeRTOS and APIs of the system, please visit <http://www.freertos.org>
 - (1). When using xTaskCreate to create a task, the task stack range is [176, 512].
 - (2). If an array whose length is over 60 bytes is used in a task, it is suggested that users use malloc and free rather than local variable to allocate array. Large local variables could lead to task stack overflow.
 - (3). The RTOS SDK takes some priorities. Priority of the pp task is 13; priority of precise timer(ms) thread is 12; priority of the TCP/IP task is 10; priority of the freeRTOS timer is 2; priority of the idle task is 0.
 - (4). Users can use tasks with priorities from 1 to 9.
 - (5). Do not revise FreeRTOSConfig.h, configurations are decided by source code inside the RTOS SDK, users can not change it.

Chapter 2

Module Index

2.1 Modules

Here is a list of all modules:

WiFi Related APIs	7
AirKiss APIs	8
SoftAP APIs	13
Station APIs	22
Common APIs	49
Force Sleep APIs	61
Rate Control APIs	64
Vendor IE APIs	68
User IE APIs	71
Sniffer APIs	74
WPS APIs	78
Smartconfig APIs	113
Misc APIs	11
Spiffs APIs	19
SSC APIs	20
System APIs	32
Boot APIs	43
Upgrade APIs	122
Software timer APIs	47
Network Espconn APIs	81
ESP-NOW APIs	99
Driver APIs	109
PWM Driver APIs	110
SPI Driver APIs	117

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

<code>_esp_event</code>	129
<code>_esp_tcp</code>	129
<code>_esp_udp</code>	130
<code>_os_timer_t</code>	131
<code>_remot_info</code>	131
<code>airkiss_config_t</code>	132
<code>bss_info</code>	132
<code>cmd_s</code>	134
<code>dhcps_lease</code>	134
<code>esp_spiffs_config</code>	135
<code>espconn</code>	136
<code>Event_Info_u</code>	137
<code>Event_SoftAPMode_ProbeReqRecved_t</code>	138
<code>Event_SoftAPMode_StaConnected_t</code>	138
<code>Event_SoftAPMode_StaDisconnected_t</code>	139
<code>Event_StaMode_AuthMode_Change_t</code>	139
<code>Event_StaMode_Connected_t</code>	139
<code>Event_StaMode_Disconnected_t</code>	140
<code>Event_StaMode_Got_IP_t</code>	141
<code>Event_StaMode_ScanDone_t</code>	141
<code>ip_info</code>	141
<code>pwm_param</code>	142
<code>rst_info</code>	143
<code>scan_config</code>	143
<code>softap_config</code>	144
<code>SpiFlashChip</code>	145
<code>station_config</code>	145
<code>station_info</code>	146
<code>upgrade_server_info</code>	146
<code>wifi_active_scan_time_t</code>	
Range of active scan times per channel	147
<code>wifi_country_t</code>	148
<code>wifi_scan_time_t</code>	
Aggregate of active & passive scan time per channel	148

Chapter 4

Module Documentation

4.1 WiFi Related APIs

WiFi APIs.

Modules

- [AirKiss APIs](#)
AirKiss APIs.
- [SoftAP APIs](#)
ESP8266 Soft-AP APIs.
- [Station APIs](#)
ESP8266 station APIs.
- [Common APIs](#)
WiFi common APIs.
- [Force Sleep APIs](#)
WiFi Force Sleep APIs.
- [Rate Control APIs](#)
WiFi Rate Control APIs.
- [Vendor IE APIs](#)
WiFi Vendor IE APIs.
- [User IE APIs](#)
WiFi User IE APIs.
- [Sniffer APIs](#)
WiFi sniffer APIs.
- [WPS APIs](#)
ESP8266 WPS APIs.
- [Smartconfig APIs](#)
SmartConfig APIs.

4.1.1 Detailed Description

WiFi APIs.

4.2 AirKiss APIs

AirKiss APIs.

Enumerations

- enum `airkiss_lan_ret_t` {
`AIRKISS_LAN_ERR_OVERFLOW` = -5, `AIRKISS_LAN_ERR_CMD` = -4, `AIRKISS_LAN_ERR_PAKE` = -3,
`AIRKISS_LAN_ERR_PARA` = -2,
`AIRKISS_LAN_ERR_PKG` = -1, `AIRKISS_LAN_CONTINUE` = 0, `AIRKISS_LAN_SSDP_REQ` = 1, `AIRKISS_LAN_PAKE_READY` = 2 }
- enum `airkiss_lan_cmdid_t` { `AIRKISS_LAN_SSDP_REQ_CMD` = 0x1, `AIRKISS_LAN_SSDP_RESP_CMD` = 0x1001, `AIRKISS_LAN_SSDP_NOTIFY_CMD` = 0x1002 }

Functions

- const char * `airkiss_version` (void)
Get the version information of AirKiss lib.
- int `airkiss_lan_rcv` (const void *body, unsigned short length, const `airkiss_config_t` *config)
Parse the UDP packet sent by AirKiss.
- int `airkiss_lan_pack` (`airkiss_lan_cmdid_t` cmdid, void *appid, void *deviceid, void *_datain, unsigned short inlength, void *_dataout, unsigned short *outlength, const `airkiss_config_t` *config)
Packaging the UDP packet.

4.2.1 Detailed Description

AirKiss APIs.

API `airkiss_lan_rcv` and `airkiss_lan_pack` are provided for the function that AirKiss can detect the ESP8266 devices in LAN, more details about AirKiss please refer to WeChat : <http://iot.weixin.qq.com>.

Workflow : Create a UDP transmission. When UDP data is received, call API `airkiss_lan_rcv` and input the UDP data, if the `airkiss_lan_rcv` returns `AIRKISS_LAN_SSDP_REQ`, `airkiss_lan_pack` can be called to make a response packet.

4.2.2 Enumeration Type Documentation

4.2.2.1 enum `airkiss_lan_ret_t`

Enumerator

- `AIRKISS_LAN_ERR_OVERFLOW`** the length of the data buffer is lack
- `AIRKISS_LAN_ERR_CMD`** Do not support the type of instruction
- `AIRKISS_LAN_ERR_PAKE`** Error reading data package
- `AIRKISS_LAN_ERR_PARA`** Error function passing parameters
- `AIRKISS_LAN_ERR_PKG`** Packet data error
- `AIRKISS_LAN_CONTINUE`** Message format is correct
- `AIRKISS_LAN_SSDP_REQ`** Find equipment request packet is received
- `AIRKISS_LAN_PAKE_READY`** Packet packaging complete

4.2.3 Function Documentation

4.2.3.1 `int airkiss_lan_pack (airkiss_lan_cmdid_t ak_lan_cmdid, void * appid, void * deviceid, void * _datain, unsigned short inlength, void * _dataout, unsigned short * outlength, const airkiss_config_t * config)`

Packaging the UDP packet.

Parameters

<i>airkiss_lan_↔ cmdid_t</i>	ak_lan_cmdid : type of the packet.
<i>void*</i>	appid : Vendor's Wechat public number id, got from WeChat.
<i>void*</i>	deviceid : device model id, got from WeChat.
<i>void*</i>	_datain : user data waiting for packet assembly.
<i>unsigned</i>	short inlength : the lenth of user data.
<i>void*</i>	_dataout : data buffer addr, to store the packet got by _datain packet assembly.
<i>unsigned</i>	short* outlength : the size of data buffer.
<i>const</i>	airkiss_config_t* config : input struct airkiss_config_t

Returns

- >=0 : succeed (reference [airkiss_lan_ret_t](#))
- <0 : error code (reference [airkiss_lan_ret_t](#))

4.2.3.2 int airkiss_lan_recv (const void * body, unsigned short length, const airkiss_config_t * config)

Parse the UDP packet sent by AirKiss.

Parameters

<i>const</i>	void* body : the start of the UDP message body data pointer.
<i>unsigned</i>	short length : the effective length of data.
<i>const</i>	airkiss_config_t* config : input struct airkiss_config_t

Returns

- >=0 : succeed (reference [airkiss_lan_ret_t](#))
- <0 : error code (reference [airkiss_lan_ret_t](#))

4.2.3.3 const char* airkiss_version (void)

Get the version information of AirKiss lib.

Attention

The lenth of version is unknown

Parameters

<i>null.</i>	
--------------	--

Returns

the version information of AirKiss lib

4.3 Misc APIs

misc APIs

Data Structures

- struct [dhcps_lease](#)

Macros

- #define **MAC2STR**(a) (a)[0], (a)[1], (a)[2], (a)[3], (a)[4], (a)[5]
- #define **MACSTR** "%02x:%02x:%02x:%02x:%02x:%02x"
- #define **IP2STR**(ipaddr)
- #define **IPSTR** "%d.%d.%d.%d"

Enumerations

- enum [dhcp_status](#) { [DHCP_STOPPED](#), [DHCP_STARTED](#) }
- enum [dhcps_offer_option](#) { [OFFER_START](#) = 0x00, [OFFER_ROUTER](#) = 0x01, [OFFER_END](#) }

Functions

- void [os_delay_us](#) (uint16 us)
Delay function, maximum value: 65535 us.
- void [os_install_putc1](#) (void(*)(char c))
Register the print output function.
- void [os_putc](#) (char c)
Print a character. Start from from UART0 by default.

4.3.1 Detailed Description

misc APIs

4.3.2 Macro Definition Documentation

4.3.2.1 #define IP2STR(ipaddr)

Value:

```
ip4_addr1_16(ipaddr), \
ip4_addr2_16(ipaddr), \
ip4_addr3_16(ipaddr), \
ip4_addr4_16(ipaddr)
```

4.3.3 Enumeration Type Documentation

4.3.3.1 enum dhcp_status

Enumerator

- **DHCP_STOPPED** disable DHCP
- **DHCP_STARTED** enable DHCP

4.3.3.2 enum dhcp_option

Enumerator

OFFER_START DHCP offer option start

OFFER_ROUTER DHCP offer router, only support this option now

OFFER_END DHCP offer option start

4.3.4 Function Documentation

4.3.4.1 void os_delay_us (uint16 us)

Delay function, maximum value: 65535 us.

Parameters

<i>uint16</i>	us : delay time, uint: us, maximum value: 65535 us
---------------	--

Returns

null

4.3.4.2 void os_install_putc1 (void(*)(char c) p)

Register the print output function.

Attention

os_install_putc1((void *)uart1_write_char) in uart_init will set printf to print from UART 1, otherwise, printf will start from UART 0 by default.

Parameters

<i>void(*)(char</i>	c) - pointer of print function
---------------------	--------------------------------

Returns

null

4.3.4.3 void os_putc (char c)

Print a character. Start from from UART0 by default.

Parameters

<i>char</i>	c - character to be printed
-------------	-----------------------------

Returns

null

4.4 SoftAP APIs

ESP8266 Soft-AP APIs.

Data Structures

- struct [softap_config](#)
- struct [station_info](#)

Functions

- bool [wifi_softap_get_config](#) (struct [softap_config](#) *config)
Get the current configuration of the ESP8266 WiFi soft-AP.
- bool [wifi_softap_get_config_default](#) (struct [softap_config](#) *config)
Get the configuration of the ESP8266 WiFi soft-AP saved in the flash.
- bool [wifi_softap_set_config](#) (struct [softap_config](#) *config)
Set the configuration of the WiFi soft-AP and save it to the Flash.
- bool [wifi_softap_set_config_current](#) (struct [softap_config](#) *config)
Set the configuration of the WiFi soft-AP; the configuration will not be saved to the Flash.
- uint8 [wifi_softap_get_station_num](#) (void)
Get the number of stations connected to the ESP8266 soft-AP.
- struct [station_info](#) * [wifi_softap_get_station_info](#) (void)
Get the information of stations connected to the ESP8266 soft-AP, including MAC and IP.
- void [wifi_softap_free_station_info](#) (void)
Free the space occupied by [station_info](#) when [wifi_softap_get_station_info](#) is called.
- bool [wifi_softap_dhcps_start](#) (void)
Enable the ESP8266 soft-AP DHCP server.
- bool [wifi_softap_dhcps_stop](#) (void)
Disable the ESP8266 soft-AP DHCP server. The DHCP is enabled by default.
- enum [dhcp_status](#) [wifi_softap_dhcps_status](#) (void)
Get the ESP8266 soft-AP DHCP server status.
- bool [wifi_softap_get_dhcps_lease](#) (struct [dhcps_lease](#) *please)
Query the IP range that can be got from the ESP8266 soft-AP DHCP server.
- bool [wifi_softap_set_dhcps_lease](#) (struct [dhcps_lease](#) *please)
Set the IP range of the ESP8266 soft-AP DHCP server.
- uint32 [wifi_softap_get_dhcps_lease_time](#) (void)
Get ESP8266 soft-AP DHCP server lease time.
- bool [wifi_softap_set_dhcps_lease_time](#) (uint32 minute)
Set ESP8266 soft-AP DHCP server lease time, default is 120 minutes.
- bool [wifi_softap_reset_dhcps_lease_time](#) (void)
Reset ESP8266 soft-AP DHCP server lease time which is 120 minutes by default.
- bool [wifi_softap_set_dhcps_offer_option](#) (uint8 level, void *optarg)
Set the ESP8266 soft-AP DHCP server option.

4.4.1 Detailed Description

ESP8266 Soft-AP APIs.

Attention

To call APIs related to ESP8266 soft-AP has to enable soft-AP mode first ([wifi_set_opmode](#))

4.4.2 Function Documentation

4.4.2.1 `bool wifi_softap_dhcps_start (void)`

Enable the ESP8266 soft-AP DHCP server.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP related API (`wifi_set_ip_info`) influence each other, if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.4.2.2 `enum dhcp_status wifi_softap_dhcps_status (void)`

Get the ESP8266 soft-AP DHCP server status.

Parameters

<i>null</i>

Returns

enum dhcp_status

4.4.2.3 `bool wifi_softap_dhcps_stop (void)`

Disable the ESP8266 soft-AP DHCP server. The DHCP is enabled by default.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.4.2.4 `void wifi_softap_free_station_info (void)`

Free the space occupied by [station_info](#) when `wifi_softap_get_station_info` is called.

Attention

The ESP8266 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP8266 station.

Parameters

<i>null</i>

Returns

null

4.4.2.5 `bool wifi_softap_get_config (struct softap_config * config)`

Get the current configuration of the ESP8266 WiFi soft-AP.

Parameters

<i>struct</i>	softap_config *config : ESP8266 soft-AP configuration
---------------	---

Returns

true : succeed
false : fail

4.4.2.6 `bool wifi_softap_get_config_default (struct softap_config * config)`

Get the configuration of the ESP8266 WiFi soft-AP saved in the flash.

Parameters

<i>struct</i>	softap_config *config : ESP8266 soft-AP configuration
---------------	---

Returns

true : succeed
false : fail

4.4.2.7 `bool wifi_softap_get_dhcps_lease (struct dhcps_lease * please)`

Query the IP range that can be got from the ESP8266 soft-AP DHCP server.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

<i>struct</i>	dhcps_lease *please : IP range of the ESP8266 soft-AP DHCP server.
---------------	--

Returns

true : succeed
false : fail

4.4.2.8 `uint32 wifi_softap_get_dhcps_lease_time (void)`

Get ESP8266 soft-AP DHCP server lease time.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

<i>null</i>

Returns

lease time, uint: minute.

4.4.2.9 struct station_info* wifi_softap_get_station_info (void)

Get the information of stations connected to the ESP8266 soft-AP, including MAC and IP.

Attention

wifi_softap_get_station_info depends on DHCP, it can only be used when DHCP is enabled, so it can not get the static IP.

Parameters

<i>null</i>

Returns

struct station_info* : station information structure

4.4.2.10 uint8 wifi_softap_get_station_num (void)

Get the number of stations connected to the ESP8266 soft-AP.

Attention

The ESP8266 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP8266 station.

Parameters

<i>null</i>

Returns

the number of stations connected to the ESP8266 soft-AP

4.4.2.11 bool wifi_softap_reset_dhcps_lease_time (void)

Reset ESP8266 soft-AP DHCP server lease time which is 120 minutes by default.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.4.2.12 bool wifi_softap_set_config (struct softap_config * config)

Set the configuration of the WiFi soft-AP and save it to the Flash.

Attention

1. This configuration will be saved in flash system parameter area if changed
2. The ESP8266 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP8266 station.

Parameters

<i>struct</i>	softap_config *config : ESP8266 soft-AP configuration
---------------	---

Returns

true : succeed
false : fail

4.4.2.13 bool wifi_softap_set_config_current (struct softap_config * config)

Set the configuration of the WiFi soft-AP; the configuration will not be saved to the Flash.

Attention

The ESP8266 is limited to only one channel, so when in the soft-AP+station mode, the soft-AP will adjust its channel automatically to be the same as the channel of the ESP8266 station.

Parameters

<i>struct</i>	softap_config *config : ESP8266 soft-AP configuration
---------------	---

Returns

true : succeed
false : fail

4.4.2.14 bool wifi_softap_set_dhcp_lease (struct dhcp_lease * lease)

Set the IP range of the ESP8266 soft-AP DHCP server.

Attention

1. The IP range should be in the same sub-net with the ESP8266 soft-AP IP address.
 2. This API should only be called when the DHCP server is disabled ([wifi_softap_dhcp_stop](#)).
 3. This configuration will only take effect the next time when the DHCP server is enabled ([wifi_softap_dhcp_start](#)).
- If the DHCP server is disabled again, this API should be called to set the IP range.
 - Otherwise, when the DHCP server is enabled later, the default IP range will be used.

Parameters

<i>struct</i>	<code>dhcp_lease</code> *please : IP range of the ESP8266 soft-AP DHCP server.
---------------	--

Returns

true : succeed
false : fail

4.4.2.15 `bool wifi_softap_set_dhcp_lease_time (uint32 minute)`

Set ESP8266 soft-AP DHCP server lease time, default is 120 minutes.

Attention

This API can only be called during ESP8266 soft-AP DHCP server enabled.

Parameters

<i>uint32</i>	minute : lease time, uint: minute, range:[1, 2880].
---------------	---

Returns

true : succeed
false : fail

4.4.2.16 `bool wifi_softap_set_dhcp_offer_option (uint8 level, void * optarg)`

Set the ESP8266 soft-AP DHCP server option.

Example:

```
uint8 mode = 0;
wifi_softap_set_dhcp_offer_option(OFFER_ROUTER, &mode);
```

Parameters

<i>uint8</i>	level : OFFER_ROUTER, set the router option.
<i>void*</i>	optarg : <ul style="list-style-type: none"> • bit0, 0 disable the router information; • bit0, 1 enable the router information.

Returns

true : succeed
false : fail

4.5 Spiffs APIs

Spiffs APIs.

Data Structures

- struct [esp_spiffs_config](#)

Functions

- sint32 [esp_spiffs_init](#) (struct [esp_spiffs_config](#) *config)
Initialize spiffs.
- void [esp_spiffs_deinit](#) (uint8 format)
Deinitialize spiffs.

4.5.1 Detailed Description

Spiffs APIs.

More details about spiffs on <https://github.com/pellepl/spiffs>

4.5.2 Function Documentation

4.5.2.1 void esp_spiffs_deinit (uint8 format)

Deinitialize spiffs.

Parameters

<i>uint8</i>	format : 0, only deinit; otherwise, deinit spiffs and format.
--------------	---

Returns

null

4.5.2.2 sint32 esp_spiffs_init (struct esp_spiffs_config * config)

Initialize spiffs.

Parameters

<i>struct</i>	esp_spiffs_config *config : ESP8266 spiffs configuration
---------------	--

Returns

0 : succeed
otherwise : fail

4.6 SSC APIs

SSC APIs.

Functions

- void `ssc_attach` (`SscBaudRate` bandrate)
Initial the ssc function.
- int `ssc_param_len` (void)
Get the length of the simple serial command.
- char * `ssc_param_str` (void)
Get the simple serial command string.
- int `ssc_parse_param` (char *pLine, char *argv[])
Parse the simple serial command (ssc).
- void `ssc_register` (`ssc_cmd_t` *cmdset, uint8 cmdnum, void(*help)(void))
Register the user-defined simple serial command (ssc) set.

4.6.1 Detailed Description

SSC APIs.

SSC means simple serial command. SSC APIs allows users to define their own command, users can refer to `spiffs_test/test_main.c`.

4.6.2 Function Documentation

4.6.2.1 void ssc_attach (SscBaudRate bandrate)

Initial the ssc function.

Parameters

<i>SscBaudRate</i>	bandrate : baud rate
--------------------	----------------------

Returns

null

4.6.2.2 int ssc_param_len (void)

Get the length of the simple serial command.

Parameters

<i>null</i>

Returns

length of the command.

4.6.2.3 char* ssc_param_str (void)

Get the simple serial command string.

Parameters

<i>null</i>	
-------------	--

Returns

the command.

4.6.2.4 `int ssc_parse_param (char * pLine, char * argv[])`

Parse the simple serial command (ssc).

Parameters

<i>char</i>	*pLine : [input] the ssc string
<i>char</i>	*argv[] : [output] parameters of the ssc

Returns

the number of parameters.

4.6.2.5 `void ssc_register (ssc_cmd_t * cmdset, uint8 cmdnum, void(*)(void) help)`

Register the user-defined simple serial command (ssc) set.

Parameters

<i>ssc_cmd_t</i>	*cmdset : the ssc set
<i>uint8</i>	cmdnum : number of commands
<i>void</i>	(* help)(void) : callback of user-guide

Returns

null

4.7 Station APIs

ESP8266 station APIs.

Data Structures

- struct [station_config](#)
- struct [wifi_active_scan_time_t](#)
Range of active scan times per channel.
- union [wifi_scan_time_t](#)
Aggregate of active & passive scan time per channel.
- struct [scan_config](#)
- struct [bss_info](#)

Typedefs

- typedef void(* [scan_done_cb_t](#)) (void *arg, STATUS status)
Callback function for wifi_station_scan.

Enumerations

- enum [wifi_scan_type_t](#) { [WIFI_SCAN_TYPE_ACTIVE](#) = 0, [WIFI_SCAN_TYPE_PASSIVE](#) }
- enum [CIPHER_TYPE](#) { [CIPHER_NONE](#) = 0, [CIPHER_WEP40](#), [CIPHER_WEP104](#), [CIPHER_TKIP](#), [CIPHER_CCMP](#), [CIPHER_TKIP_CCMP](#), [CIPHER_UNKNOWN](#) }
- enum [STATION_STATUS](#) { [STATION_IDLE](#) = 0, [STATION_CONNECTING](#), [STATION_WRONG_PASSWORD](#), [STATION_NO_AP_FOUND](#), [STATION_CONNECT_FAIL](#), [STATION_GOT_IP](#) }

Functions

- bool [wifi_station_get_config](#) (struct [station_config](#) *config)
Get the current configuration of the ESP8266 WiFi station.
- bool [wifi_station_get_config_default](#) (struct [station_config](#) *config)
Get the configuration parameters saved in the Flash of the ESP8266 WiFi station.
- bool [wifi_station_set_config](#) (struct [station_config](#) *config)
Set the configuration of the ESP8266 station and save it to the Flash.
- bool [wifi_station_set_config_current](#) (struct [station_config](#) *config)
Set the configuration of the ESP8266 station. And the configuration will not be saved to the Flash.
- bool [wifi_station_connect](#) (void)
Connect the ESP8266 WiFi station to the AP.
- bool [wifi_station_disconnect](#) (void)
Disconnect the ESP8266 WiFi station from the AP.
- bool [wifi_station_scan](#) (struct [scan_config](#) *config, [scan_done_cb_t](#) cb)
Scan all available APs.
- bool [wifi_station_get_auto_connect](#) (void)
Check if the ESP8266 station will connect to the recorded AP automatically when the power is on.
- bool [wifi_station_set_auto_connect](#) (bool set)
Set whether the ESP8266 station will connect to the recorded AP automatically when the power is on. It will do so by default.

- bool [wifi_station_get_reconnect_policy](#) (void)
Check whether the ESP8266 station will reconnect to the AP after disconnection.
- bool [wifi_station_set_reconnect_policy](#) (bool set)
Set whether the ESP8266 station will reconnect to the AP after disconnection. It will do so by default.
- [STATION_STATUS](#) [wifi_station_get_connect_status](#) (void)
Get the connection status of the ESP8266 WiFi station.
- uint8 [wifi_station_get_current_ap_id](#) (void)
Get the information of APs (5 at most) recorded by ESP8266 station.
- bool [wifi_station_ap_change](#) (uint8 current_ap_id)
Switch the ESP8266 station connection to a recorded AP.
- bool [wifi_station_ap_number_set](#) (uint8 ap_number)
Set the number of APs that can be recorded in the ESP8266 station. When the ESP8266 station is connected to an AP, the SSID and password of the AP will be recorded.
- uint8 [wifi_station_get_ap_info](#) (struct [station_config](#) config[])
Get the information of APs (5 at most) recorded by ESP8266 station.
- sint8 [wifi_station_get_rssi](#) (void)
Get rssi of the AP which ESP8266 station connected to.
- bool [wifi_station_dhcpc_start](#) (void)
Enable the ESP8266 station DHCP client.
- bool [wifi_station_dhcpc_stop](#) (void)
Disable the ESP8266 station DHCP client.
- enum [dhcp_status](#) [wifi_station_dhcpc_status](#) (void)
Get the ESP8266 station DHCP client status.
- bool [wifi_station_set_hostname](#) (char *name)
Set ESP8266 station DHCP hostname.
- char * [wifi_station_get_hostname](#) (void)
Get ESP8266 station DHCP hostname.

4.7.1 Detailed Description

ESP8266 station APIs.

Attention

To call APIs related to ESP8266 station has to enable station mode first ([wifi_set_opmode](#))

4.7.2 Typedef Documentation

4.7.2.1 typedef void(* scan_done_cb_t) (void *arg, STATUS status)

Callback function for [wifi_station_scan](#).

Parameters

<i>void</i>	*arg : information of APs that are found; save them as linked list; refer to struct bss_info
<i>STATUS</i>	status : status of scanning

Returns

null

4.7.3 Enumeration Type Documentation

4.7.3.1 enum CIPHER_TYPE

Enumerator

CIPHER_NONE the cipher type is none
CIPHER_WEP40 the cipher type is WEP40
CIPHER_WEP104 the cipher type is WEP104
CIPHER_TKIP the cipher type is TKIP
CIPHER_CCMP the cipher type is CCMP
CIPHER_TKIP_CCMP the cipher type is TKIP and CCMP
CIPHER_UNKNOWN the cipher type is unknown

4.7.3.2 enum STATION_STATUS

Enumerator

STATION_IDLE ESP8266 station idle
STATION_CONNECTING ESP8266 station is connecting to AP
STATION_WRONG_PASSWORD the password is wrong
STATION_NO_AP_FOUND ESP8266 station can not find the target AP
STATION_CONNECT_FAIL ESP8266 station fail to connect to AP
STATION_GOT_IP ESP8266 station got IP address from AP

4.7.3.3 enum wifi_scan_type_t

Enumerator

WIFI_SCAN_TYPE_ACTIVE active scan
WIFI_SCAN_TYPE_PASSIVE passive scan

4.7.4 Function Documentation

4.7.4.1 bool wifi_station_ap_change (uint8 current_ap_id)

Switch the ESP8266 station connection to a recorded AP.

Parameters

<i>uint8</i>	<i>new_ap_id</i> : AP's record id, start counting from 0.
--------------	---

Returns

true : succeed
false : fail

4.7.4.2 bool wifi_station_ap_number_set (uint8 ap_number)

Set the number of APs that can be recorded in the ESP8266 station. When the ESP8266 station is connected to an AP, the SSID and password of the AP will be recorded.

Attention

This configuration will be saved in the Flash system parameter area if changed.

Parameters

<i>uint8</i>	ap_number : the number of APs that can be recorded (MAX: 5)
--------------	---

Returns

true : succeed
false : fail

4.7.4.3 bool wifi_station_connect (void)

Connect the ESP8266 WiFi station to the AP.

Attention

1. This API should be called when the ESP8266 station is enabled, and the system initialization is completed. Do not call this API in user_init.
2. If the ESP8266 is connected to an AP, call wifi_station_disconnect to disconnect.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.7.4.4 bool wifi_station_dhcp_start (void)

Enable the ESP8266 station DHCP client.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP API ((wifi_set_ip_info)) influence each other, and if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.7.4.5 enum dhcp_status wifi_station_dhcp_status (void)

Get the ESP8266 station DHCP client status.

Parameters

<i>null</i>

Returns

enum dhcp_status

4.7.4.6 bool wifi_station_dhcpc_stop (void)

Disable the ESP8266 station DHCP client.

Attention

1. The DHCP is enabled by default.
2. The DHCP and the static IP API ((wifi_set_ip_info)) influence each other, and if the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.7.4.7 bool wifi_station_disconnect (void)

Disconnect the ESP8266 WiFi station from the AP.

Attention

This API should be called when the ESP8266 station is enabled, and the system initialization is completed. Do not call this API in user_init.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.7.4.8 uint8 wifi_station_get_ap_info (struct station_config config[])

Get the information of APs (5 at most) recorded by ESP8266 station.

Example:

```
struct station_config config[5];
uint8_t i = wifi_station_get_ap_info(config);
```

Parameters

<i>struct</i>	station_config config[] : information of the APs, the array size should be 5.
---------------	---

Returns

The number of APs recorded.

4.7.4.9 `bool wifi_station_get_auto_connect (void)`

Check if the ESP8266 station will connect to the recorded AP automatically when the power is on.

Parameters

<i>null</i>	
-------------	--

Returns

true : connect to the AP automatically
false : not connect to the AP automatically

4.7.4.10 `bool wifi_station_get_config (struct station_config * config)`

Get the current configuration of the ESP8266 WiFi station.

Parameters

<i>struct</i>	station_config *config : ESP8266 station configuration
---------------	--

Returns

true : succeed
false : fail

4.7.4.11 `bool wifi_station_get_config_default (struct station_config * config)`

Get the configuration parameters saved in the Flash of the ESP8266 WiFi station.

Parameters

<i>struct</i>	station_config *config : ESP8266 station configuration
---------------	--

Returns

true : succeed
false : fail

4.7.4.12 `STATION_STATUS wifi_station_get_connect_status (void)`

Get the connection status of the ESP8266 WiFi station.

Parameters

<i>null</i>

Returns

the status of connection

4.7.4.13 `uint8 wifi_station_get_current_ap_id (void)`

Get the information of APs (5 at most) recorded by ESP8266 station.

Parameters

<i>struct</i>	<code>station_config config[]</code> : information of the APs, the array size should be 5.
---------------	--

Returns

The number of APs recorded.

4.7.4.14 `char* wifi_station_get_hostname (void)`

Get ESP8266 station DHCP hostname.

Parameters

<i>null</i>

Returns

the hostname of ESP8266 station

4.7.4.15 `bool wifi_station_get_reconnect_policy (void)`

Check whether the ESP8266 station will reconnect to the AP after disconnection.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.7.4.16 `sint8 wifi_station_get_rssi (void)`

Get rssi of the AP which ESP8266 station connected to.

Parameters

<i>null</i>

Returns

31 : fail, invalid value.
others : succeed, value of rssi. In general, rssi value < 10

4.7.4.17 `bool wifi_station_scan (struct scan_config * config, scan_done_cb_t cb)`

Scan all available APs.

Attention

This API should be called when the ESP8266 station is enabled, and the system initialization is completed. Do not call this API in `user_init`.

Parameters

<i>struct</i>	<code>scan_config</code> *config : configuration of scanning
<i>struct</i>	<code>scan_done_cb_t</code> cb : callback of scanning

Returns

true : succeed
false : fail

4.7.4.18 `bool wifi_station_set_auto_connect (bool set)`

Set whether the ESP8266 station will connect to the recorded AP automatically when the power is on. It will do so by default.

Attention

1. If this API is called in `user_init`, it is effective immediately after the power is on. If it is called in other places, it will be effective the next time when the power is on.
2. This configuration will be saved in Flash system parameter area if changed.

Parameters

<i>bool</i>	<p>set : If it will automatically connect to the AP when the power is on</p> <ul style="list-style-type: none"> • true : it will connect automatically • false: it will not connect automatically
-------------	---

Returns

true : succeed
false : fail

4.7.4.19 `bool wifi_station_set_config (struct station_config * config)`

Set the configuration of the ESP8266 station and save it to the Flash.

Attention

1. This API can be called only when the ESP8266 station is enabled.
2. If `wifi_station_set_config` is called in `user_init`, there is no need to call `wifi_station_connect`. The ESP8266 station will automatically connect to the AP (router) after the system initialization. Otherwise, `wifi_station_connect` should be called.
3. Generally, `station_config.bssid_set` needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.
4. This configuration will be saved in the Flash system parameter area if changed.

Parameters

<i>struct</i>	station_config *config : ESP8266 station configuration
---------------	--

Returns

true : succeed
false : fail

4.7.4.20 bool wifi_station_set_config_current (struct station_config * config)

Set the configuration of the ESP8266 station. And the configuration will not be saved to the Flash.

Attention

1. This API can be called only when the ESP8266 station is enabled.
2. If wifi_station_set_config_current is called in user_init , there is no need to call wifi_station_connect. The ESP8266 station will automatically connect to the AP (router) after the system initialization. Otherwise, wifi_station_connect should be called.
3. Generally, [station_config.bssid_set](#) needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

Parameters

<i>struct</i>	station_config *config : ESP8266 station configuration
---------------	--

Returns

true : succeed
false : fail

4.7.4.21 bool wifi_station_set_hostname (char * name)

Set ESP8266 station DHCP hostname.

Parameters

<i>char</i>	*name : hostname of ESP8266 station
-------------	-------------------------------------

Returns

true : succeed
false : fail

4.7.4.22 bool wifi_station_set_reconnect_policy (bool set)

Set whether the ESP8266 station will reconnect to the AP after disconnection. It will do so by default.

Attention

If users want to call this API, it is suggested that users call this API in user_init.

Parameters

<i>bool</i>	set : if it's true, it will enable reconnection; if it's false, it will disable reconnection.
-------------	---

Returns

true : succeed
false : fail

4.8 System APIs

System APIs.

Modules

- [Boot APIs](#)
boot APIs
- [Upgrade APIs](#)
Firmware upgrade (FOTA) APIs.

Data Structures

- struct [rst_info](#)

Enumerations

- enum [rst_reason](#) {
REASON_DEFAULT_RST = 0, REASON_WDT_RST, REASON_EXCEPTION_RST, REASON_SOFT_WDT_RST,
REASON_SOFT_RESTART, REASON_DEEP_SLEEP_AWAKE, REASON_EXT_SYS_RST }

Functions

- struct [rst_info](#) * [system_get_rst_info](#) (void)
Get the reason of restart.
- const char * [system_get_sdk_version](#) (void)
Get information of the SDK version.
- void [system_restore](#) (void)
Reset to default settings.
- void [system_restart](#) (void)
Restart system.
- void [system_deep_sleep](#) (uint32 time_in_us)
Set the chip to deep-sleep mode.
- bool [system_deep_sleep_set_option](#) (uint8 option)
Call this API before system_deep_sleep to set the activity after the next deep-sleep wakeup.
- uint32 [system_get_time](#) (void)
Get system time, unit: microsecond.
- void [system_print_meminfo](#) (void)
Print the system memory distribution, including data/rodata/bss/heap.
- uint32 [system_get_free_heap_size](#) (void)
Get the size of available heap.
- uint32 [system_get_chip_id](#) (void)
Get the chip ID.
- uint32 [system_rtc_clock_cal_proc](#) (void)
Get the RTC clock cycle.
- uint32 [system_get_rtc_time](#) (void)
Get RTC time, unit: RTC clock cycle.
- bool [system_rtc_mem_read](#) (uint8 src, void *dst, uint16 n)
Read user data from the RTC memory.

- bool `system_rtc_mem_write` (uint8 dst, const void *src, uint16 n)
Write user data to the RTC memory.
- void `system_uart_swap` (void)
UART0 swap.
- void `system_uart_de_swap` (void)
Disable UART0 swap.
- uint16 `system_adc_read` (void)
Measure the input voltage of TOUT pin 6, unit : 1/1024 V.
- uint16 `system_get_vdd33` (void)
Measure the power voltage of VDD3P3 pin 3 and 4, unit : 1/1024 V.
- bool `system_param_save_with_protect` (uint16 start_sec, void *param, uint16 len)
Write data into flash with protection.
- bool `system_param_load` (uint16 start_sec, uint16 offset, void *param, uint16 len)
Read the data saved into flash with the read/write protection.
- void `system_phy_set_max_tpw` (uint8 max_tpw)
Set the maximum value of RF TX Power, unit : 0.25dBm.
- void `system_phy_set_tpw_via_vdd33` (uint16 vdd33)
Adjust the RF TX Power according to VDD33, unit : 1/1024 V.
- void `system_phy_set_rfoption` (uint8 option)
Enable RF or not when wakeup from deep-sleep.

4.8.1 Detailed Description

System APIs.

4.8.2 Enumeration Type Documentation

4.8.2.1 enum rst_reason

Enumerator

- REASON_DEFAULT_RST** normal startup by power on
- REASON_WDT_RST** hardware watch dog reset
- REASON_EXCEPTION_RST** exception reset, GPIO status won't change
- REASON_SOFT_WDT_RST** software watch dog reset, GPIO status won't change
- REASON_SOFT_RESTART** software restart ,system_restart , GPIO status won't change
- REASON_DEEP_SLEEP_AWAKE** wake up from deep-sleep
- REASON_EXT_SYS_RST** external system reset

4.8.3 Function Documentation

4.8.3.1 uint16 system_adc_read (void)

Measure the input voltage of TOUT pin 6, unit : 1/1024 V.

Attention

1. `system_adc_read` can only be called when the TOUT pin is connected to the external circuitry, and the TOUT pin input voltage should be limited to 0~1.0V.
2. When the TOUT pin is connected to the external circuitry, the 107th byte (`vdd33_const`) of `esp_init_data`↔`_default.bin`(0~127byte) should be set as the real power voltage of VDD3P3 pin 3 and 4.
3. The unit of `vdd33_const` is 0.1V, the effective value range is [18, 36]; if `vdd33_const` is in [0, 18) or (36, 255), 3.3V is used to optimize RF by default.

Parameters

<i>null</i>

Returns

Input voltage of TOUT pin 6, unit : 1/1024 V

4.8.3.2 void system_deep_sleep (uint32 time_in_us)

Set the chip to deep-sleep mode.

The device will automatically wake up after the deep-sleep time set by the users. Upon waking up, the device boots up from user_init.

Attention

1. XPD_DCDC should be connected to EXT_RSTB through 0 ohm resistor in order to support deep-sleep wakeup.
2. system_deep_sleep(0): there is no wake up timer; in order to wake up, connect a GPIO to pin RST, the chip will wake up by a falling-edge on pin RST

Parameters

<i>uint32</i>	time_in_us : deep-sleep time, unit: microsecond
---------------	---

Returns

null

4.8.3.3 bool system_deep_sleep_set_option (uint8 option)

Call this API before system_deep_sleep to set the activity after the next deep-sleep wakeup.

If this API is not called, default to be system_deep_sleep_set_option(1).

Parameters

<i>uint8</i>	option :
0	: Radio calibration after the deep-sleep wakeup is decided by byte 108 of esp_init_data_↔ default.bin (0~127byte).
1	: Radio calibration will be done after the deep-sleep wakeup. This will lead to stronger current.
2	: Radio calibration will not be done after the deep-sleep wakeup. This will lead to weaker current.
4	: Disable radio calibration after the deep-sleep wakeup (the same as modem-sleep). This will lead to the weakest current, but the device can't receive or transmit data after waking up.

Returns

true : succeed
false : fail

4.8.3.4 uint32 system_get_chip_id (void)

Get the chip ID.

Parameters

<i>null</i>

Returns

The chip ID.

4.8.3.5 uint32 system_get_free_heap_size (void)

Get the size of available heap.

Parameters

<i>null</i>

Returns

Available heap size.

4.8.3.6 struct rst_info* system_get_rst_info (void)

Get the reason of restart.

Parameters

<i>null</i>

Returns

struct rst_info* : information of the system restart

4.8.3.7 uint32 system_get_rtc_time (void)

Get RTC time, unit: RTC clock cycle.

Example: If system_get_rtc_time returns 10 (it means 10 RTC cycles), and system_rtc_clock_cal_proc returns 5.75 (it means 5.75 microseconds per RTC clock cycle), (then the actual time is $10 \times 5.75 = 57.5$ microseconds).

Attention

System time will return to zero because of system_restart, but the RTC time still goes on. If the chip is reset by pin EXT_RST or pin CHIP_EN (including the deep-sleep wakeup), situations are shown as below:

1. reset by pin EXT_RST : RTC memory won't change, RTC timer returns to zero
2. watchdog reset : RTC memory won't change, RTC timer won't change
3. system_restart : RTC memory won't change, RTC timer won't change
4. power on : RTC memory is random value, RTC timer starts from zero
5. reset by pin CHIP_EN : RTC memory is random value, RTC timer starts from zero

Parameters

<i>null</i>

Returns

RTC time.

4.8.3.8 `const char* system_get_sdk_version (void)`

Get information of the SDK version.

Parameters

<i>null</i>

Returns

Information of the SDK version.

4.8.3.9 uint32 system_get_time (void)

Get system time, unit: microsecond.

Parameters

<i>null</i>

Returns

System time, unit: microsecond.

4.8.3.10 uint16 system_get_vdd33 (void)

Measure the power voltage of VDD3P3 pin 3 and 4, unit : 1/1024 V.

Attention

1. system_get_vdd33 depends on RF, please do not use it if RF is disabled.
2. system_get_vdd33 can only be called when TOUT pin is suspended.
3. The 107th byte in esp_init_data_default.bin (0~127byte) is named as "vdd33_const", when TOUT pin is suspended vdd33_const must be set as 0xFF, that is 255.

Parameters

<i>null</i>

Returns

Power voltage of VDD33, unit : 1/1024 V

4.8.3.11 bool system_param_load (uint16 start_sec, uint16 offset, void * param, uint16 len)

Read the data saved into flash with the read/write protection.

Flash read/write has to be 4-bytes aligned.

Read/write protection of flash: use 3 sectors (4KB per sector) to save 4KB data with protect, sector 0 and sector 1 are data sectors, back up each other, save data alternately, sector 2 is flag sector, point out which sector is keeping the latest data, sector 0 or sector 1.

Parameters

<i>uint16</i>	start_sec : start sector (sector 0) of the 3 sectors used for flash read/write protection. It cannot be sector 1 or sector 2. • For example, in IOT_Demo, the 3 sectors (3 * 4KB) starting from flash 0x3D000 can be used for flash read/write protection. The parameter start_sec is 0x3D, and it cannot be 0x3E or 0x3F.
<i>uint16</i>	offset : offset of data saved in sector
<i>void</i>	*param : data pointer
<i>uint16</i>	len : data length, offset + len =< 4 * 1024

Returns

true : succeed
false : fail

4.8.3.12 bool system_param_save_with_protect (uint16 start_sec, void * param, uint16 len)

Write data into flash with protection.

Flash read/write has to be 4-bytes aligned.

Protection of flash read/write : use 3 sectors (4KBytes per sector) to save 4KB data with protect, sector 0 and sector 1 are data sectors, back up each other, save data alternately, sector 2 is flag sector, point out which sector is keeping the latest data, sector 0 or sector 1.

Parameters

<i>uint16</i>	start_sec : start sector (sector 0) of the 3 sectors which are used for flash read/write protection. • For example, in IOT_Demo we can use the 3 sectors (3 * 4KB) starting from flash 0x3D000 for flash read/write protection, so the parameter start_sec should be 0x3D
<i>void</i>	*param : pointer of the data to be written
<i>uint16</i>	len : data length, should be less than a sector, which is 4 * 1024

Returns

true : succeed
false : fail

4.8.3.13 void system_phy_set_max_tpw (uint8 max_tpw)

Set the maximum value of RF TX Power, unit : 0.25dBm.

Parameters

<i>uint8</i>	max_tpw : the maximum value of RF Tx Power, unit : 0.25dBm, range [0, 82]. It can be set refer to the 34th byte (target_power_qdb_0) of esp_init_data_default.bin(0~127byte)
--------------	--

Returns

null

4.8.3.14 void system_phy_set_rfoption (uint8 option)

Enable RF or not when wakeup from deep-sleep.

Attention

1. This API can only be called in user_rf_pre_init.
2. Function of this API is similar to system_deep_sleep_set_option, if they are both called, it will disregard system_deep_sleep_set_option which is called before deep-sleep, and refer to system_phy_set_rfoption which is called when deep-sleep wake up.
3. Before calling this API, system_deep_sleep_set_option should be called once at least.

Parameters

<i>uint8</i>	option : <ul style="list-style-type: none"> • 0 : Radio calibration after deep-sleep wake up depends on esp_init_data_default.bin (0~127byte) byte 108. • 1 : Radio calibration is done after deep-sleep wake up; this increases the current consumption. • 2 : No radio calibration after deep-sleep wake up; this reduces the current consumption. • 4 : Disable RF after deep-sleep wake up, just like modem sleep; this has the least current consumption; the device is not able to transmit or receive data after wake up.
--------------	--

Returns

null

4.8.3.15 void system_phy_set_tpw_via_vdd33 (uint16 vdd33)

Adjust the RF TX Power according to VDD33, unit : 1/1024 V.

Attention

1. When TOUT pin is suspended, VDD33 can be measured by system_get_vdd33.
2. When TOUT pin is connected to the external circuitry, system_get_vdd33 can not be used to measure VDD33.

Parameters

<i>uint16</i>	vdd33 : VDD33, unit : 1/1024V, range [1900, 3300]
---------------	---

Returns

null

4.8.3.16 void system_print_meminfo (void)

Print the system memory distribution, including data/rodata/bss/heap.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.8.3.17 void system_restart (void)

Restart system.

Parameters

<i>null</i>

Returns

null

4.8.3.18 void system_restore (void)

Reset to default settings.

Reset to default settings of the following APIs : wifi_station_set_auto_connect, wifi_set_phy_mode, wifi_softap_↔ set_config related, wifi_station_set_config related, and wifi_set_opmode.

Parameters

<i>null</i>

Returns

null

4.8.3.19 uint32 system_rtc_clock_cali_proc (void)

Get the RTC clock cycle.

Attention

1. The RTC clock cycle has decimal part.
2. The RTC clock cycle will change according to the temperature, so RTC timer is not very precise.

Parameters

<i>null</i>

Returns

RTC clock period (unit: microsecond), bit11~ bit0 are decimal.

4.8.3.20 bool system_rtc_mem_read (uint8 src, void * dst, uint16 n)

Read user data from the RTC memory.

The user data segment (512 bytes, as shown below) is used to store user data.

|<— system data(256 bytes) —>|<----- user data(512 bytes) ----->|

Attention

Read and write unit for data stored in the RTC memory is 4 bytes.
src_addr is the block number (4 bytes per block). So when reading data at the beginning of the user data segment, src_addr will be $256/4 = 64$, n will be data length.

Parameters

<i>uint8</i>	src : source address of rtc memory, src_addr >= 64
<i>void</i>	*dst : data pointer
<i>uint16</i>	n : data length, unit: byte

Returns

true : succeed
false : fail

4.8.3.21 bool system_rtc_mem_write (uint8 dst, const void * src, uint16 n)

Write user data to the RTC memory.

During deep-sleep, only RTC is working. So users can store their data in RTC memory if it is needed. The user data segment below (512 bytes) is used to store the user data.

|<--- system data(256 bytes) --->|<----- user data(512 bytes) ----->|

Attention

Read and write unit for data stored in the RTC memory is 4 bytes.
src_addr is the block number (4 bytes per block). So when storing data at the beginning of the user data segment, src_addr will be 256/4 = 64, n will be data length.

Parameters

<i>uint8</i>	src : source address of rtc memory, src_addr >= 64
<i>void</i>	*dst : data pointer
<i>uint16</i>	n : data length, unit: byte

Returns

true : succeed
false : fail

4.8.3.22 void system_uart_de_swap (void)

Disable UART0 swap.

Use the original UART0, not MTCK and MTDO.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.8.3.23 void system_uart_swap (void)

UART0 swap.

Use MTCK as UART0 RX, MTDO as UART0 TX, so ROM log will not output from this new UART0. We also need to use MTDO (U0RTS) and MTCK (U0CTS) as UART0 in hardware.

Parameters

<i>null</i>

Returns

null

4.9 Boot APIs

boot APIs

Macros

- `#define SYS_BOOT_ENHANCE_MODE 0`
- `#define SYS_BOOT_NORMAL_MODE 1`
- `#define SYS_BOOT_NORMAL_BIN 0`
- `#define SYS_BOOT_TEST_BIN 1`
- `#define SYS_CPU_80MHZ 80`
- `#define SYS_CPU_160MHZ 160`

Enumerations

- enum `flash_size_map` {
`FLASH_SIZE_4M_MAP_256_256 = 0, FLASH_SIZE_2M, FLASH_SIZE_8M_MAP_512_512, FLASH_SIZE_16M_MAP_512_512,`
`FLASH_SIZE_32M_MAP_512_512, FLASH_SIZE_16M_MAP_1024_1024, FLASH_SIZE_32M_MAP_1024_1024, FLASH_SIZE_32M_MAP_2048_2048,`
`FLASH_SIZE_64M_MAP_1024_1024, FLASH_SIZE_128M_MAP_1024_1024 }`

Functions

- `uint8 system_get_boot_version` (void)
Get information of the boot version.
- `uint32 system_get_userbin_addr` (void)
Get the address of the current running user bin (user1.bin or user2.bin).
- `uint8 system_get_boot_mode` (void)
Get the boot mode.
- `bool system_restart_enhance` (uint8 bin_type, uint32 bin_addr)
Restarts the system, and enters the enhanced boot mode.
- `flash_size_map system_get_flash_size_map` (void)
Get the current Flash size and Flash map.
- `bool system_update_cpu_freq` (uint8 freq)
Set CPU frequency. Default is 80MHz.
- `uint8 system_get_cpu_freq` (void)
Get CPU frequency.

4.9.1 Detailed Description

boot APIs

4.9.2 Macro Definition Documentation

4.9.2.1 `#define SYS_BOOT_ENHANCE_MODE 0`

It can load and run firmware at any address, for Espressif factory test bin

4.9.2.2 #define SYS_BOOT_NORMAL_BIN 0

user1.bin or user2.bin

4.9.2.3 #define SYS_BOOT_NORMAL_MODE 1

It can only load and run at some addresses of user1.bin (or user2.bin)

4.9.2.4 #define SYS_BOOT_TEST_BIN 1

can only be Espressif test bin

4.9.3 Enumeration Type Documentation

4.9.3.1 enum flash_size_map

Enumerator

FLASH_SIZE_4M_MAP_256_256 Flash size : 4Mbits. Map : 256KBytes + 256KBytes

FLASH_SIZE_2M Flash size : 2Mbits. Map : 256KBytes

FLASH_SIZE_8M_MAP_512_512 Flash size : 8Mbits. Map : 512KBytes + 512KBytes

FLASH_SIZE_16M_MAP_512_512 Flash size : 16Mbits. Map : 512KBytes + 512KBytes

FLASH_SIZE_32M_MAP_512_512 Flash size : 32Mbits. Map : 512KBytes + 512KBytes

FLASH_SIZE_16M_MAP_1024_1024 Flash size : 16Mbits. Map : 1024KBytes + 1024KBytes

FLASH_SIZE_32M_MAP_1024_1024 Flash size : 32Mbits. Map : 1024KBytes + 1024KBytes

FLASH_SIZE_32M_MAP_2048_2048 attention: don't support now ,just compatible for nodemcu; Flash size : 32Mbits. Map : 2048KBytes + 2048KBytes

FLASH_SIZE_64M_MAP_1024_1024 Flash size : 64Mbits. Map : 1024KBytes + 1024KBytes

FLASH_SIZE_128M_MAP_1024_1024 Flash size : 128Mbits. Map : 1024KBytes + 1024KBytes

4.9.4 Function Documentation

4.9.4.1 uint8 system_get_boot_mode (void)

Get the boot mode.

Parameters

<i>null</i>

Returns

```
#define SYS_BOOT_ENHANCE_MODE 0
```

```
#define SYS_BOOT_NORMAL_MODE 1
```

4.9.4.2 uint8 system_get_boot_version (void)

Get information of the boot version.

Attention

If boot version ≥ 1.3 , users can enable the enhanced boot mode (refer to system_restart_enhance).

Parameters

<i>null</i>

Returns

Information of the boot version.

4.9.4.3 uint8 system_get_cpu_freq (void)

Get CPU frequency.

Parameters

<i>null</i>

Returns

CPU frequency, unit : MHz.

4.9.4.4 flash_size_map system_get_flash_size_map (void)

Get the current Flash size and Flash map.

Flash map depends on the selection when compiling, more details in document "2A-ESP8266__IOT_SDK_User↔_Manual"

Parameters

<i>null</i>

Returns

enum flash_size_map

4.9.4.5 uint32 system_get_userbin_addr (void)

Get the address of the current running user bin (user1.bin or user2.bin).

Parameters

<i>null</i>

Returns

The address of the current running user bin.

4.9.4.6 bool system_restart_enhance (uint8 bin_type, uint32 bin_addr)

Restarts the system, and enters the enhanced boot mode.

Attention

SYS_BOOT_TEST_BIN is used for factory test during production; users can apply for the test bin from Espressif Systems.

Parameters

<i>uint8</i>	bin_type : type of bin <ul style="list-style-type: none"> • #define SYS_BOOT_NORMAL_BIN 0 // user1.bin or user2.bin • #define SYS_BOOT_TEST_BIN 1 // can only be Espressif test bin
<i>uint32</i>	bin_addr : starting address of the bin file

Returns

true : succeed
false : fail

4.9.4.7 bool system_update_cpu_freq (uint8 freq)

Set CPU frequency. Default is 80MHz.

System bus frequency is 80MHz, will not be affected by CPU frequency. The frequency of UART, SPI, or other peripheral devices, are divided from system bus frequency, so they will not be affected by CPU frequency either.

Parameters

<i>uint8</i>	freq : CPU frequency, 80 or 160.
--------------	----------------------------------

Returns

true : succeed
false : fail

4.10 Software timer APIs

Software timer APIs.

Functions

- void `os_timer_setfn` (`os_timer_t` *ptimer, `os_timer_func_t` *pfunction, void *parg)
Set the timer callback function.
- void `os_timer_arm` (`os_timer_t` *ptimer, uint32 msec, bool repeat_flag)
Enable the millisecond timer.
- void `os_timer_disarm` (`os_timer_t` *ptimer)
Disarm the timer.

4.10.1 Detailed Description

Software timer APIs.

Timers of the following interfaces are software timers. Functions of the timers are executed during the tasks. Since a task can be stopped, or be delayed because there are other tasks with higher priorities, the following `os_timer` interfaces cannot guarantee the precise execution of the timers.

- For the same timer, `os_timer_arm` (or `os_timer_arm_us`) cannot be invoked repeatedly. `os_timer_disarm` should be invoked first.
- `os_timer_setfn` can only be invoked when the timer is not enabled, i.e., after `os_timer_disarm` or before `os_timer_arm` (or `os_timer_arm_us`).

4.10.2 Function Documentation

4.10.2.1 void `os_timer_arm` (`os_timer_t` * *ptimer*, uint32 msec, bool *repeat_flag*)

Enable the millisecond timer.

Parameters

<i>os_timer_t</i>	*ptimer : timer structure
<i>uint32_t</i>	milliseconds : Timing, unit: millisecond, range: 5 ~ 0x68DB8
<i>bool</i>	repeat_flag : Whether the timer will be invoked repeatedly or not

Returns

null

4.10.2.2 void `os_timer_disarm` (`os_timer_t` * *ptimer*)

Disarm the timer.

Parameters

<i>os_timer_t</i>	*ptimer : Timer structure
-------------------	---------------------------

Returns

null

4.10.2.3 void os_timer_setfn (os_timer_t * ptimer, os_timer_func_t * pfunction, void * parg)

Set the timer callback function.

Attention

1. The callback function must be set in order to enable the timer.
2. Operating system scheduling is disabled in timer callback.

Parameters

<i>os_timer_t</i>	*ptimer : Timer structure
<i>os_timer_func_t</i>	*pfunction : timer callback function
<i>void</i>	*parg : callback function parameter

Returns

null

4.11 Common APIs

WiFi common APIs.

Data Structures

- struct [wifi_country_t](#)
- struct [ip_info](#)
- struct [Event_StaMode_ScanDone_t](#)
- struct [Event_StaMode_Connected_t](#)
- struct [Event_StaMode_Disconnected_t](#)
- struct [Event_StaMode_AuthMode_Change_t](#)
- struct [Event_StaMode_Got_IP_t](#)
- struct [Event_SoftAPMode_StaConnected_t](#)
- struct [Event_SoftAPMode_StaDisconnected_t](#)
- struct [Event_SoftAPMode_ProbeReqRecved_t](#)
- union [Event_Info_u](#)
- struct [_esp_event](#)

Typedefs

- typedef struct [_esp_event](#) **System_Event_t**
- typedef void(* [wifi_event_handler_cb_t](#)) ([System_Event_t](#) *event)

The Wi-Fi event handler.
- typedef void(* [freedom_outside_cb_t](#)) (uint8 status)

Callback of sending user-define 802.11 packets.
- typedef void(* [rfid_locp_cb_t](#)) (uint8 *frm, int len, sint8 rssi)

RFID LOCP (Location Control Protocol) receive callback .

Enumerations

- enum [WIFI_MODE](#) {
 - [NULL_MODE](#) = 0, [STATION_MODE](#), [SOFTAP_MODE](#), [STATIONAP_MODE](#),
 - [MAX_MODE](#)** }
- enum [AUTH_MODE](#) {
 - [AUTH_OPEN](#) = 0, [AUTH_WEP](#), [AUTH_WPA_PSK](#), [AUTH_WPA2_PSK](#),
 - [AUTH_WPA_WPA2_PSK](#), **[AUTH_MAX](#)** }
- enum [WIFI_COUNTRY_POLICY](#) { [WIFI_COUNTRY_POLICY_AUTO](#), [WIFI_COUNTRY_POLICY_MANU](#)↵
[AL](#) }
- enum [WIFI_INTERFACE](#) { [STATION_IF](#) = 0, [SOFTAP_IF](#), **[MAX_IF](#)** }
- enum [WIFI_PHY_MODE](#) { [PHY_MODE_11B](#) = 1, [PHY_MODE_11G](#) = 2, [PHY_MODE_11N](#) = 3 }
- enum [SYSTEM_EVENT](#) {
 - [EVENT_STAMODE_SCAN_DONE](#) = 0, [EVENT_STAMODE_CONNECTED](#), [EVENT_STAMODE_DISCO](#)↵
[NNECTED](#), [EVENT_STAMODE_AUTHMODE_CHANGE](#),
 - [EVENT_STAMODE_GOT_IP](#), [EVENT_STAMODE_DHCP_TIMEOUT](#), [EVENT_SOFTAPMODE_STACO](#)↵
[NNECTED](#), [EVENT_SOFTAPMODE_STADISCONNECTED](#),
 - [EVENT_SOFTAPMODE_PROBEREQRECVD](#), **[EVENT_MAX](#)** }
- enum {
 - [REASON_UNSPECIFIED](#)** = 1, **[REASON_AUTH_EXPIRE](#)** = 2, **[REASON_AUTH_LEAVE](#)** = 3, **[REASON](#)**↵
[ASSOC_EXPIRE](#) = 4,
 - [REASON_ASSOC_TOOMANY](#)** = 5, **[REASON_NOT_AUTHED](#)** = 6, **[REASON_NOT_ASSOCED](#)** = 7, **[RE](#)**↵
[ASON_ASSOC_LEAVE](#) = 8,
 - [REASON_ASSOC_NOT_AUTHED](#)** = 9, **[REASON_DISASSOC_PWRCAP_BAD](#)** = 10, **[REASON_DISAS](#)**↵

- SOC_SUPCHAN_BAD = 11, REASON_IE_INVALID = 13,
 REASON_MIC_FAILURE = 14, REASON_4WAY_HANDSHAKE_TIMEOUT = 15, REASON_GROUP_KEY_UPDATE_TIMEOUT = 16, REASON_IE_IN_4WAY_DIFFERS = 17,
 REASON_GROUP_CIPHER_INVALID = 18, REASON_PAIRWISE_CIPHER_INVALID = 19, REASON_AKMP_INVALID = 20, REASON_UNSUPP_RSN_IE_VERSION = 21,
 REASON_INVALID_RSN_IE_CAP = 22, REASON_802_1X_AUTH_FAILED = 23, REASON_CIPHER_SUITE_REJECTED = 24, REASON_BEACON_TIMEOUT = 200,
 REASON_NO_AP_FOUND = 201, REASON_AUTH_FAIL = 202, REASON_ASSOC_FAIL = 203, REASON_HANDSHAKE_TIMEOUT = 204 }
- enum `sleep_type` { NONE_SLEEP_T = 0, LIGHT_SLEEP_T, MODEM_SLEEP_T }

Functions

- [WIFI_MODE wifi_get_opmode](#) (void)
Get the current operating mode of the WiFi.
- [WIFI_MODE wifi_get_opmode_default](#) (void)
Get the operating mode of the WiFi saved in the Flash.
- bool [wifi_set_opmode](#) (WIFI_MODE opmode)
Set the WiFi operating mode, and save it to Flash.
- bool [wifi_set_opmode_current](#) (WIFI_MODE opmode)
Set the WiFi operating mode, and will not save it to Flash.
- bool [wifi_get_ip_info](#) (WIFI_INTERFACE if_index, struct [ip_info](#) *info)
Get the IP address of the ESP8266 WiFi station or the soft-AP interface.
- bool [wifi_set_ip_info](#) (WIFI_INTERFACE if_index, struct [ip_info](#) *info)
Set the IP address of the ESP8266 WiFi station or the soft-AP interface.
- bool [wifi_get_macaddr](#) (WIFI_INTERFACE if_index, uint8 *macaddr)
Get MAC address of the ESP8266 WiFi station or the soft-AP interface.
- bool [wifi_set_macaddr](#) (WIFI_INTERFACE if_index, uint8 *macaddr)
Set MAC address of the ESP8266 WiFi station or the soft-AP interface.
- void [wifi_status_led_install](#) (uint8 gpio_id, uint32 gpio_name, uint8 gpio_func)
Install the WiFi status LED.
- void [wifi_status_led_uninstall](#) (void)
Uninstall the WiFi status LED.
- [WIFI_PHY_MODE wifi_get_phy_mode](#) (void)
Get the ESP8266 physical mode (802.11b/g/n).
- bool [wifi_set_phy_mode](#) (WIFI_PHY_MODE mode)
Set the ESP8266 physical mode (802.11b/g/n).
- bool [wifi_set_event_handler_cb](#) (wifi_event_handler_cb_t cb)
Register the Wi-Fi event handler.
- sint32 [wifi_register_send_pkt_freedom_cb](#) (freedom_outside_cb_t cb)
Register a callback for sending user-define 802.11 packets.
- void [wifi_unregister_send_pkt_freedom_cb](#) (void)
Unregister the callback for sending user-define 802.11 packets.
- sint32 [wifi_send_pkt_freedom](#) (uint8 *buf, uint16 len, bool sys_seq)
Send user-define 802.11 packets.
- sint32 [wifi_rfid_locp_rcv_open](#) (void)
Enable RFID LOCP (Location Control Protocol) to receive WDS packets.
- void [wifi_rfid_locp_rcv_close](#) (void)
Disable RFID LOCP (Location Control Protocol).
- sint32 [wifi_register_rfid_locp_rcv_cb](#) (rfid_locp_cb_t cb)
Register a callback of receiving WDS packets.
- void [wifi_unregister_rfid_locp_rcv_cb](#) (void)

Unregister the callback of receiving WDS packets.

- bool `wifi_set_sleep_type` (sleep_type type)

Sets sleep type.

- sleep_type `wifi_get_sleep_type` (void)

Gets sleep type.

4.11.1 Detailed Description

WiFi common APIs.

The Flash system parameter area is the last 16KB of the Flash.

4.11.2 Typedef Documentation

4.11.2.1 typedef void(* freedom_outside_cb_t) (uint8 status)

Callback of sending user-define 802.11 packets.

Parameters

<i>uint8</i>	status : 0, packet sending succeed; otherwise, fail.
--------------	--

Returns

null

4.11.2.2 typedef void(* rfid_locp_cb_t) (uint8 *frm, int len, sint8 rssi)

RFID LOCP (Location Control Protocol) receive callback .

Parameters

<i>uint8</i>	*frm : point to the head of 802.11 packet
<i>int</i>	len : packet length
<i>int</i>	rssi : signal strength

Returns

null

4.11.2.3 typedef void(* wifi_event_handler_cb_t) (System_Event_t *event)

The Wi-Fi event handler.

Attention

No complex operations are allowed in callback. If users want to execute any complex operations, please post message to another task instead.

Parameters

<code>System_Event_t</code>	*event : WiFi event
-----------------------------	---------------------

Returns

null

4.11.3 Enumeration Type Documentation

4.11.3.1 enum AUTH_MODE

Enumerator

- AUTH_OPEN** authenticate mode : open
- AUTH_WEP** authenticate mode : WEP
- AUTH_WPA_PSK** authenticate mode : WPA_PSK
- AUTH_WPA2_PSK** authenticate mode : WPA2_PSK
- AUTH_WPA_WPA2_PSK** authenticate mode : WPA_WPA2_PSK

4.11.3.2 enum SYSTEM_EVENT

Enumerator

- EVENT_STAMODE_SCAN_DONE** ESP8266 station finish scanning AP
- EVENT_STAMODE_CONNECTED** ESP8266 station connected to AP
- EVENT_STAMODE_DISCONNECTED** ESP8266 station disconnected to AP
- EVENT_STAMODE_AUTHMODE_CHANGE** the auth mode of AP connected by ESP8266 station changed
- EVENT_STAMODE_GOT_IP** ESP8266 station got IP from connected AP
- EVENT_STAMODE_DHCP_TIMEOUT** ESP8266 station dhcp client got IP timeout
- EVENT_SOFTAPMODE_STACONNECTED** a station connected to ESP8266 soft-AP
- EVENT_SOFTAPMODE_STADISCONNECTED** a station disconnected to ESP8266 soft-AP
- EVENT_SOFTAPMODE_PROBEREQRECVED** Receive probe request packet in soft-AP interface

4.11.3.3 enum WIFI_COUNTRY_POLICY

Enumerator

- WIFI_COUNTRY_POLICY_AUTO** Country policy is auto, use the country info of AP to which the station is connected
- WIFI_COUNTRY_POLICY_MANUAL** Country policy is manual, always use the configured country info

4.11.3.4 enum WIFI_INTERFACE

Enumerator

- STATION_IF** ESP8266 station interface
- SOFTAP_IF** ESP8266 soft-AP interface

4.11.3.5 enum WIFI_MODE

Enumerator

- NULL_MODE** null mode
- STATION_MODE** WiFi station mode
- SOFTAP_MODE** WiFi soft-AP mode
- STATIONAP_MODE** WiFi station + soft-AP mode

4.11.3.6 enum WIFI_PHY_MODE

Enumerator

- PHY_MODE_11B** 802.11b
- PHY_MODE_11G** 802.11g
- PHY_MODE_11N** 802.11n

4.11.4 Function Documentation

4.11.4.1 bool wifi_get_ip_info (WIFI_INTERFACE if_index, struct ip_info * info)

Get the IP address of the ESP8266 WiFi station or the soft-AP interface.

Attention

Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.

Parameters

<i>WIFI_INTERF</i> _↔ <i>ACE</i>	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
<i>struct</i>	ip_info *info : the IP information obtained.

Returns

- true : succeed
- false : fail

4.11.4.2 bool wifi_get_macaddr (WIFI_INTERFACE if_index, uint8 * macaddr)

Get MAC address of the ESP8266 WiFi station or the soft-AP interface.

Parameters

<i>WIFI_INTERF</i> _↔ <i>ACE</i>	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
<i>uint8</i>	*macaddr : the MAC address.

Returns

- true : succeed
- false : fail

4.11.4.3 WIFI_MODE wifi_get_opmode (void)

Get the current operating mode of the WiFi.

Parameters

<i>null</i>

Returns

WiFi operating modes:

- 0x01: station mode;
- 0x02: soft-AP mode
- 0x03: station+soft-AP mode

4.11.4.4 **WIFI_MODE** `wifi_get_opmode_default (void)`

Get the operating mode of the WiFi saved in the Flash.

Parameters

<i>null</i>

Returns

WiFi operating modes:

- 0x01: station mode;
- 0x02: soft-AP mode
- 0x03: station+soft-AP mode

4.11.4.5 **WIFI_PHY_MODE** `wifi_get_phy_mode (void)`

Get the ESP8266 physical mode (802.11b/g/n).

Parameters

<i>null</i>

Returns

enum WIFI_PHY_MODE

4.11.4.6 `sleep_type` `wifi_get_sleep_type (void)`

Gets sleep type.

Parameters

<i>null</i>

Returns

sleep type

4.11.4.7 `sint32` `wifi_register_rfid_locp_rcv_cb (rfid_locp_cb_t cb)`

Register a callback of receiving WDS packets.

Register a callback of receiving WDS packets. Only if the first MAC address of the WDS packet is a multicast address.

Parameters

<i>rfid_locp_cb_t</i>	cb : callback
-----------------------	---------------

Returns

0, succeed;
otherwise, fail.

4.11.4.8 sint32 wifi_register_send_pkt_freedom_cb (freedom_outside_cb_t cb)

Register a callback for sending user-define 802.11 packets.

Attention

Only after the previous packet was sent, entered the *freedom_outside_cb_t*, the next packet is allowed to send.

Parameters

<i>freedom_↔ outside_cb_t</i>	cb : sent callback
-----------------------------------	--------------------

Returns

0, succeed;
-1, fail.

4.11.4.9 void wifi_rfid_locp_recv_close (void)

Disable RFID LOCP (Location Control Protocol) .

Parameters

<i>null</i>

Returns

null

4.11.4.10 sint32 wifi_rfid_locp_recv_open (void)

Enable RFID LOCP (Location Control Protocol) to receive WDS packets.

Parameters

<i>null</i>

Returns

0, succeed;
otherwise, fail.

4.11.4.11 sint32 wifi_send_pkt_freedom (uint8 * buf, uint16 len, bool sys_seq)

Send user-define 802.11 packets.

Attention

1. Packet has to be the whole 802.11 packet, does not include the FCS. The length of the packet has to be longer than the minimum length of the header of 802.11 packet which is 24 bytes, and less than 1400 bytes.
2. Duration area is invalid for user, it will be filled in SDK.
3. The rate of sending packet is same as the management packet which is the same as the system rate of sending packets.
4. Only after the previous packet was sent, entered the sent callback, the next packet is allowed to send. Otherwise, wifi_send_pkt_freedom will return fail.

Parameters

<i>uint8</i>	*buf : pointer of packet
<i>uint16</i>	len : packet length
<i>bool</i>	sys_seq : follow the system's 802.11 packets sequence number or not, if it is true, the sequence number will be increased 1 every time a packet sent.

Returns

0, succeed;
-1, fail.

4.11.4.12 bool wifi_set_event_handler_cb (wifi_event_handler_cb_t cb)

Register the Wi-Fi event handler.

Parameters

<i>wifi_event_handler_cb_t</i>	cb : callback function
--------------------------------	------------------------

Returns

true : succeed
false : fail

4.11.4.13 bool wifi_set_ip_info (WIFI_INTERFACE if_index, struct ip_info * info)

Set the IP address of the ESP8266 WiFi station or the soft-AP interface.

Attention

1. Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.
2. To set static IP, users need to disable DHCP first (wifi_station_dhcpc_stop or wifi_softap_dhcps_stop):
 - If the DHCP is enabled, the static IP will be disabled; if the static IP is enabled, the DHCP will be disabled. It depends on the latest configuration.

Parameters

<i>WIFI_INTERF</i> ↔ <i>ACE</i>	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
<i>struct</i>	ip_info *info : the IP information obtained.

Returns

true : succeed

false : fail

4.11.4.14 bool wifi_set_macaddr (WIFI_INTERFACE if_index, uint8 * macaddr)

Set MAC address of the ESP8266 WiFi station or the soft-AP interface.

Attention

1. This API can only be called in user_init.
2. Users need to enable the target interface (station or soft-AP) by wifi_set_opmode first.
3. ESP8266 soft-AP and station have different MAC addresses, do not set them to be the same.
 - The bit0 of the first byte of ESP8266 MAC address can not be 1. For example, the MAC address can set to be "1a:XX:XX:XX:XX:XX", but can not be "15:XX:XX:XX:XX:XX".

Parameters

<i>WIFI_INTERF</i> ↔ <i>ACE</i>	if_index : get the IP address of the station or the soft-AP interface, 0x00 for STATION_IF, 0x01 for SOFTAP_IF.
<i>uint8</i>	*macaddr : the MAC address.

Returns

true : succeed

false : fail

4.11.4.15 bool wifi_set_opmode (WIFI_MODE opmode)

Set the WiFi operating mode, and save it to Flash.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, and save it to Flash. The default mode is soft-AP mode.

Attention

This configuration will be saved in the Flash system parameter area if changed.

Parameters

<i>uint8</i>	opmode : WiFi operating modes: <ul style="list-style-type: none"> • 0x01: station mode; • 0x02: soft-AP mode • 0x03: station+soft-AP mode
--------------	--

Returns

true : succeed

false : fail

4.11.4.16 bool wifi_set_opmode_current (WIFI_MODE *opmode*)

Set the WiFi operating mode, and will not save it to Flash.

Set the WiFi operating mode as station, soft-AP or station+soft-AP, and the mode won't be saved to the Flash.

Parameters

<i>uint8</i>	opmode : WiFi operating modes: <ul style="list-style-type: none"> • 0x01: station mode; • 0x02: soft-AP mode • 0x03: station+soft-AP mode
--------------	--

Returns

true : succeed

false : fail

4.11.4.17 bool wifi_set_phy_mode (WIFI_PHY_MODE *mode*)

Set the ESP8266 physical mode (802.11b/g/n).

Attention

The ESP8266 soft-AP only supports bg.

Parameters

<i>WIFI_PHY_MODE</i>	mode : physical mode
----------------------	----------------------

Returns

true : succeed

false : fail

4.11.4.18 bool wifi_set_sleep_type (sleep_type *type*)

Sets sleep type.

Set NONE_SLEEP_T to disable sleep. Default to be Modem sleep.

Attention

Sleep function only takes effect in station-only mode.

Parameters

<i>sleep_type</i>	type : sleep type
-------------------	-------------------

Returns

true : succeed

false : fail

4.11.4.19 void wifi_status_led_install (uint8 *gpio_id*, uint32 *gpio_name*, uint8 *gpio_func*)

Install the WiFi status LED.

Parameters

<i>uint8</i>	gpio_id : GPIO ID
<i>uint8</i>	gpio_name : GPIO mux name
<i>uint8</i>	gpio_func : GPIO function

Returns

null

4.11.4.20 void wifi_status_led_uninstall (void)

Uninstall the WiFi status LED.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.11.4.21 void wifi_unregister_rfid_locp_rcv_cb (void)

Unregister the callback of receiving WDS packets.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.11.4.22 void wifi_unregister_send_pkt_freedom_cb (void)

Unregister the callback for sending user-define 802.11 packets.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.12 Force Sleep APIs

WiFi Force Sleep APIs.

Typedefs

- typedef void(* **fpm_wakeup_cb**) (void)

Functions

- void [wifi_fpm_open](#) (void)
Enable force sleep function.
- void [wifi_fpm_close](#) (void)
Disable force sleep function.
- void [wifi_fpm_do_wakeup](#) (void)
Wake ESP8266 up from MODEM_SLEEP_T force sleep.
- void [wifi_fpm_set_wakeup_cb](#) (fpm_wakeup_cb cb)
Set a callback of waken up from force sleep because of time out.
- sint8 [wifi_fpm_do_sleep](#) (uint32 sleep_time_in_us)
Force ESP8266 enter sleep mode, and it will wake up automatically when time out.
- void [wifi_fpm_set_sleep_type](#) (sleep_type type)
Set sleep type for force sleep function.
- sleep_type [wifi_fpm_get_sleep_type](#) (void)
Get sleep type of force sleep function.

4.12.1 Detailed Description

WiFi Force Sleep APIs.

4.12.2 Function Documentation

4.12.2.1 void [wifi_fpm_close](#) (void)

Disable force sleep function.

Parameters

<i>null</i>

Returns

null

4.12.2.2 sint8 [wifi_fpm_do_sleep](#) (uint32 *sleep_time_in_us*)

Force ESP8266 enter sleep mode, and it will wake up automatically when time out.

Attention

1. This API can only be called when force sleep function is enabled, after calling [wifi_fpm_open](#). This API can not be called after calling [wifi_fpm_close](#).
2. If this API returned 0 means that the configuration is set successfully, but the ESP8266 will not enter sleep mode immediately, it is going to sleep in the system idle task. Please do not call other WiFi related function right after calling this API.

Parameters

<i>uint32</i>	<p>sleep_time_in_us : sleep time, ESP8266 will wake up automatically when time out. Unit: us. Range: 10000 ~ 268435455(0xFFFFFFFF).</p> <ul style="list-style-type: none"> • If sleep_time_in_us is 0xFFFFFFFF, the ESP8266 will sleep till • if wifi_fpm_set_sleep_type is set to be LIGHT_SLEEP_T, ESP8266 can wake up by GPIO. • if wifi_fpm_set_sleep_type is set to be MODEM_SLEEP_T, ESP8266 can wake up by wifi_fpm_do_wakeup.
---------------	--

Returns

- 0, setting succeed;
- 1, fail to sleep, sleep status error;
- 2, fail to sleep, force sleep function is not enabled.

4.12.2.3 void wifi_fpm_do_wakeup (void)

Wake ESP8266 up from MODEM_SLEEP_T force sleep.

Attention

This API can only be called when MODEM_SLEEP_T force sleep function is enabled, after calling wifi_fpm_↔_open. This API can not be called after calling wifi_fpm_close.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.12.2.4 sleep_type wifi_fpm_get_sleep_type (void)

Get sleep type of force sleep function.

Parameters

<i>null</i>	
-------------	--

Returns

sleep type

4.12.2.5 void wifi_fpm_open (void)

Enable force sleep function.

Attention

Force sleep function is disabled by default.

Parameters

<i>null</i>

Returns

null

4.12.2.6 void wifi_fpm_set_sleep_type (sleep_type type)

Set sleep type for force sleep function.

Attention

This API can only be called before wifi_fpm_open.

Parameters

<i>sleep_type</i>	type : sleep type
-------------------	-------------------

Returns

null

4.12.2.7 void wifi_fpm_set_wakeup_cb (fpm_wakeup_cb cb)

Set a callback of waken up from force sleep because of time out.

Attention

1. This API can only be called when force sleep function is enabled, after calling wifi_fpm_open. This API can not be called after calling wifi_fpm_close.
2. fpm_wakeup_cb_func will be called after system woke up only if the force sleep time out (wifi_fpm_do_sleep and the parameter is not 0xFFFFFFFF).
3. fpm_wakeup_cb_func will not be called if woke up by wifi_fpm_do_wakeup from MODEM_SLEEP_T type force sleep.

Parameters

<i>void</i>	(*fpm_wakeup_cb_func)(void) : callback of waken up
-------------	--

Returns

null

4.13 Rate Control APIs

WiFi Rate Control APIs.

Macros

- #define **FIXED_RATE_MASK_NONE** 0x00
- #define **FIXED_RATE_MASK_STA** 0x01
- #define **FIXED_RATE_MASK_AP** 0x02
- #define **FIXED_RATE_MASK_ALL** 0x03
- #define **RC_LIMIT_11B** 0
- #define **RC_LIMIT_11G** 1
- #define **RC_LIMIT_11N** 2
- #define **RC_LIMIT_P2P_11G** 3
- #define **RC_LIMIT_P2P_11N** 4
- #define **RC_LIMIT_NUM** 5
- #define **LIMIT_RATE_MASK_NONE** 0x00
- #define **LIMIT_RATE_MASK_STA** 0x01
- #define **LIMIT_RATE_MASK_AP** 0x02
- #define **LIMIT_RATE_MASK_ALL** 0x03

Enumerations

- enum **FIXED_RATE** {
PHY_RATE_48 = 0x8, **PHY_RATE_24** = 0x9, **PHY_RATE_12** = 0xA, **PHY_RATE_6** = 0xB,
PHY_RATE_54 = 0xC, **PHY_RATE_36** = 0xD, **PHY_RATE_18** = 0xE, **PHY_RATE_9** = 0xF }
- enum **support_rate** {
RATE_11B5M = 0, **RATE_11B11M** = 1, **RATE_11B1M** = 2, **RATE_11B2M** = 3,
RATE_11G6M = 4, **RATE_11G12M** = 5, **RATE_11G24M** = 6, **RATE_11G48M** = 7,
RATE_11G54M = 8, **RATE_11G9M** = 9, **RATE_11G18M** = 10, **RATE_11G36M** = 11 }
- enum **RATE_11B_ID** { **RATE_11B_B11M** = 0, **RATE_11B_B5M** = 1, **RATE_11B_B2M** = 2, **RATE_11B_B1M** = 3 }
- enum **RATE_11G_ID** {
RATE_11G_G54M = 0, **RATE_11G_G48M** = 1, **RATE_11G_G36M** = 2, **RATE_11G_G24M** = 3,
RATE_11G_G18M = 4, **RATE_11G_G12M** = 5, **RATE_11G_G9M** = 6, **RATE_11G_G6M** = 7,
RATE_11G_B5M = 8, **RATE_11G_B2M** = 9, **RATE_11G_B1M** = 10 }
- enum **RATE_11N_ID** {
RATE_11N_MCS7S = 0, **RATE_11N_MCS7** = 1, **RATE_11N_MCS6** = 2, **RATE_11N_MCS5** = 3,
RATE_11N_MCS4 = 4, **RATE_11N_MCS3** = 5, **RATE_11N_MCS2** = 6, **RATE_11N_MCS1** = 7,
RATE_11N_MCS0 = 8, **RATE_11N_B5M** = 9, **RATE_11N_B2M** = 10, **RATE_11N_B1M** = 11 }

Functions

- sint32 [wifi_set_user_fixed_rate](#) (uint8 enable_mask, uint8 rate)
Set the fixed rate and mask of sending data from ESP8266.
- int [wifi_get_user_fixed_rate](#) (uint8 *enable_mask, uint8 *rate)
Get the fixed rate and mask of ESP8266.
- sint32 [wifi_set_user_sup_rate](#) (uint8 min, uint8 max)
Set the support rate of ESP8266.
- bool [wifi_set_user_rate_limit](#) (uint8 mode, uint8 ifidx, uint8 max, uint8 min)
Limit the initial rate of sending data from ESP8266.
- uint8 [wifi_get_user_limit_rate_mask](#) (void)
Get the interfaces of ESP8266 whose rate of sending data is limited by wifi_set_user_rate_limit.
- bool [wifi_set_user_limit_rate_mask](#) (uint8 enable_mask)
Set the interfaces of ESP8266 whose rate of sending packets is limited by wifi_set_user_rate_limit.

4.13.1 Detailed Description

WiFi Rate Control APIs.

4.13.2 Function Documentation

4.13.2.1 `int wifi_get_user_fixed_rate (uint8 * enable_mask, uint8 * rate)`

Get the fixed rate and mask of ESP8266.

Parameters

<i>uint8</i>	*enable_mask : pointer of the enable_mask
<i>uint8</i>	*rate : pointer of the fixed rate

Returns

0 : succeed
otherwise : fail

4.13.2.2 `uint8 wifi_get_user_limit_rate_mask (void)`

Get the interfaces of ESP8266 whose rate of sending data is limited by wifi_set_user_rate_limit.

Parameters

<i>null</i>	
-------------	--

Returns

LIMIT_RATE_MASK_NONE - disable the limitation on both ESP8266 station and soft-AP
 LIMIT_RATE_MASK_STA - enable the limitation on ESP8266 station
 LIMIT_RATE_MASK_AP - enable the limitation on ESP8266 soft-AP
 LIMIT_RATE_MASK_ALL - enable the limitation on both ESP8266 station and soft-AP

4.13.2.3 `sint32 wifi_set_user_fixed_rate (uint8 enable_mask, uint8 rate)`

Set the fixed rate and mask of sending data from ESP8266.

Attention

1. Only if the corresponding bit in enable_mask is 1, ESP8266 station or soft-AP will send data in the fixed rate.
2. If the enable_mask is 0, both ESP8266 station and soft-AP will not send data in the fixed rate.
3. ESP8266 station and soft-AP share the same rate, they can not be set into the different rate.

Parameters

<i>uint8</i>	enable_mask : 0x00 - disable the fixed rate <ul style="list-style-type: none"> • 0x01 - use the fixed rate on ESP8266 station • 0x02 - use the fixed rate on ESP8266 soft-AP • 0x03 - use the fixed rate on ESP8266 station and soft-AP
<i>uint8</i>	rate : value of the fixed rate

Returns

0 : succeed
 otherwise : fail

4.13.2.4 bool wifi_set_user_limit_rate_mask (uint8 enable_mask)

Set the interfaces of ESP8266 whose rate of sending packets is limited by wifi_set_user_rate_limit.

Parameters

<i>uint8</i>	enable_mask : <ul style="list-style-type: none"> • LIMIT_RATE_MASK_NONE - disable the limitation on both ESP8266 station and soft-AP • LIMIT_RATE_MASK_STA - enable the limitation on ESP8266 station • LIMIT_RATE_MASK_AP - enable the limitation on ESP8266 soft-AP • LIMIT_RATE_MASK_ALL - enable the limitation on both ESP8266 station and soft-AP
--------------	---

Returns

true : succeed
 false : fail

4.13.2.5 bool wifi_set_user_rate_limit (uint8 mode, uint8 ifidx, uint8 max, uint8 min)

Limit the initial rate of sending data from ESP8266.

Example: wifi_set_user_rate_limit(RC_LIMIT_11G, 0, RATE_11G_G18M, RATE_11G_G6M);

Attention

The rate of retransmission is not limited by this API.

Parameters

<i>uint8</i>	mode : WiFi mode <ul style="list-style-type: none"> • #define RC_LIMIT_11B 0 • #define RC_LIMIT_11G 1 • #define RC_LIMIT_11N 2
<i>uint8</i>	ifidx : interface of ESP8266 <ul style="list-style-type: none"> • 0x00 - ESP8266 station • 0x01 - ESP8266 soft-AP

<i>uint8</i>	max : the maximum value of the rate, according to the enum rate corresponding to the first parameter mode.
<i>uint8</i>	min : the minimum value of the rate, according to the enum rate corresponding to the first parameter mode.

Returns

0 : succeed
otherwise : fail

4.13.2.6 sint32 wifi_set_user_sup_rate (uint8 min, uint8 max)

Set the support rate of ESP8266.

Set the rate range in the IE of support rate in ESP8266's beacon, probe req/resp and other packets. Tell other devices about the rate range supported by ESP8266 to limit the rate of sending packets from other devices. Example : `wifi_set_user_sup_rate(RATE_11G6M, RATE_11G24M);`

Attention

This API can only support 802.11g now, but it will support 802.11b in next version.

Parameters

<i>uint8</i>	min : the minimum value of the support rate, according to enum support_rate.
<i>uint8</i>	max : the maximum value of the support rate, according to enum support_rate.

Returns

0 : succeed
otherwise : fail

4.14 Vendor IE APIs

WiFi Vendor IE APIs.

Typedefs

- typedef void(* [vendor_ie_rcv_cb_t](#)) ([vendor_ie_type](#) type, const uint8 sa[6], const uint8 *vnd_ie, sint32 rssi)
Vendor IE received callback.

Enumerations

- enum [vendor_ie_type](#) {
VND_IE_TYPE_BEACON = 0, VND_IE_TYPE_PROBE_REQ, VND_IE_TYPE_PROBE_RESP, VND_IE_TYPE_ASSOC_REQ,
VND_IE_TYPE_ASSOC_RESP, VND_IE_TYPE_NUM }

Functions

- bool [wifi_set_vnd_ie](#) (bool enable, [vendor_ie_type](#) type, uint8_t idx, uint8_t *vnd_ie)
Set Vendor IE of ESP8266.
- sint32 [wifi_register_vnd_ie_rcv_cb](#) ([vendor_ie_rcv_cb_t](#) cb)
Register vendor IE received callback.
- void [wifi_unregister_vnd_ie_rcv_cb](#) (void)
Unregister vendor IE received callback.

4.14.1 Detailed Description

WiFi Vendor IE APIs.

4.14.2 Typedef Documentation

4.14.2.1 typedef void(* [vendor_ie_rcv_cb_t](#)) ([vendor_ie_type](#) type, const uint8 sa[6], const uint8 *vnd_ie, sint32 rssi)

Vendor IE received callback.

Parameters

<i>vendor_ie_type</i>	type : type of vendor IE.
<i>const</i>	uint8 sa[6] : source address of the packet.
<i>uint8</i>	*vnd_ie : pointer of vendor IE.
<i>sint32</i>	rssi : signal strength.

Returns

null

4.14.3 Enumeration Type Documentation

4.14.3.1 enum [vendor_ie_type](#)

Enumerator

VND_IE_TYPE_BEACON beacon

VND_IE_TYPE_PROBE_REQ probe request
VND_IE_TYPE_PROBE_RESP probe response
VND_IE_TYPE_ASSOC_REQ associate request
VND_IE_TYPE_ASSOC_RESP associate response

4.14.4 Function Documentation

4.14.4.1 `sint32 wifi_register_vnd_ie_rcv_cb (vendor_ie_rcv_cb_t cb)`

Register vendor IE received callback.

Parameters

<code>vendor_ie_rcv_cb_t</code>	cb : callback
---------------------------------	---------------

Returns

0 : succeed
-1 : fail

4.14.4.2 `bool wifi_set_vnd_ie (bool enable, vendor_ie_type type, uint8_t idx, uint8_t * vnd_ie)`

Set Vendor IE of ESP8266.

The Vendor IE will be added to the target packets of `vendor_ie_type`.

Parameters

<code>bool</code>	enable : <ul style="list-style-type: none"> true, enable the corresponding vendor-specific IE function, all parameters below have to be set. false, disable the corresponding vendor-specific IE function and release the resource, only the parameter "type" below has to be set.
<code>uint8_t</code>	type : IE type. If it is <code>VND_IE_TYPE_BEACON</code> , please disable the IE function and enable again to take the configuration effect immediately .
<code>uint8_t</code>	idx : vendor-specific IE index, 0 or 1. Only support two vendor-specific IEs in one frame.
<code>uint8_t</code>	*vnd_ie : vendor-specific information elements, need to input the whole 802.11 IE including Element ID, Length, Organization Identifier and Vendor-specific Content.

Returns

true : succeed
false : fail

4.14.4.3 `void wifi_unregister_vnd_ie_rcv_cb (void)`

Unregister vendor IE received callback.

Parameters

<i>null</i>

Returns

null

4.15 User IE APIs

WiFi User IE APIs.

Typedefs

- typedef void(* [user_ie_manufacturer_rcv_cb_t](#)) (user_ie_type type, const uint8 sa[6], const uint8 m_oui[3], uint8 *ie, uint8 ie_len, sint32 rssi)

User IE received callback.

Enumerations

- enum **user_ie_type** {
USER_IE_BEACON = 0, **USER_IE_PROBE_REQ**, **USER_IE_PROBE_RESP**, **USER_IE_ASSOC_REQ**,
USER_IE_ASSOC_RESP, **USER_IE_MAX** }

Functions

- bool [wifi_set_user_ie](#) (bool enable, uint8 *m_oui, user_ie_type type, uint8 *user_ie, uint8 len)
Set user IE of ESP8266.
- sint32 [wifi_register_user_ie_manufacturer_rcv_cb](#) ([user_ie_manufacturer_rcv_cb_t](#) cb)
Register user IE received callback.
- void [wifi_unregister_user_ie_manufacturer_rcv_cb](#) (void)
Unregister user IE received callback.

4.15.1 Detailed Description

WiFi User IE APIs.

4.15.2 Typedef Documentation

- 4.15.2.1 typedef void(* [user_ie_manufacturer_rcv_cb_t](#)) (user_ie_type type, const uint8 sa[6], const uint8 m_oui[3], uint8 *ie, uint8 ie_len, sint32 rssi)

User IE received callback.

Parameters

<i>user_ie_type</i>	type : type of user IE.
<i>const</i>	uint8 sa[6] : source address of the packet.
<i>const</i>	uint8 m_oui[3] : factory tag.
<i>uint8</i>	*user_ie : pointer of user IE.
<i>uint8</i>	ie_len : length of user IE.
<i>sint32</i>	rssi : signal strength.

Returns

null

4.15.3 Function Documentation

4.15.3.1 `sint32 wifi_register_user_ie_manufacturer_rcv_cb (user_ie_manufacturer_rcv_cb_t cb)`

Register user IE received callback.

Parameters

<i>user_ie_↔</i> <i>manufacturer_↔</i> <i>recv_cb_t</i>	cb : callback
---	---------------

Returns

0 : succeed
-1 : fail

4.15.3.2 bool wifi_set_user_ie (bool enable, uint8 * m_oui, user_ie_type type, uint8 * user_ie, uint8 len)

Set user IE of ESP8266.

The user IE will be added to the target packets of user_ie_type.

Parameters

<i>bool</i>	enable : <ul style="list-style-type: none"> • true, enable the corresponding user IE function, all parameters below have to be set. • false, disable the corresponding user IE function and release the resource, only the parameter "type" below has to be set.
<i>uint8</i>	*m_oui : factory tag, apply for it from Espressif System.
<i>user_ie_type</i>	type : IE type. If it is USER_IE_BEACON, please disable the IE function and enable again to take the configuration effect immediately .
<i>uint8</i>	*user_ie : user-defined information elements, need not input the whole 802.11 IE, need only the user-define part.
<i>uint8</i>	len : length of user IE, 247 bytes at most.

Returns

true : succeed
false : fail

4.15.3.3 void wifi_unregister_user_ie_manufacturer_recv_cb (void)

Unregister user IE received callback.

Parameters

<i>null</i>

Returns

null

4.16 Sniffer APIs

WiFi sniffer APIs.

Typedefs

- typedef void(* [wifi_promiscuous_cb_t](#)) (uint8 *buf, uint16 len)
The RX callback function in the promiscuous mode.

Functions

- void [wifi_set_promiscuous_rx_cb](#) ([wifi_promiscuous_cb_t](#) cb)
Register the RX callback function in the promiscuous mode.
- uint8 [wifi_get_channel](#) (void)
Get the channel number for sniffer functions.
- bool [wifi_set_channel](#) (uint8 channel)
Set the channel number for sniffer functions.
- bool [wifi_promiscuous_set_mac](#) (const uint8_t *address)
Set the MAC address filter for the sniffer mode.
- void [wifi_promiscuous_enable](#) (uint8 promiscuous)
Enable the promiscuous mode.
- bool [wifi_set_country](#) ([wifi_country_t](#) *country)
configure country info
- bool [wifi_get_country](#) ([wifi_country_t](#) *country)
get the current country info

4.16.1 Detailed Description

WiFi sniffer APIs.

4.16.2 Typedef Documentation

4.16.2.1 typedef void(* [wifi_promiscuous_cb_t](#)) (uint8 *buf, uint16 len)

The RX callback function in the promiscuous mode.

Each time a packet is received, the callback function will be called.

Parameters

<i>uint8</i>	*buf : the data received
<i>uint16</i>	len : data length

Returns

null

4.16.3 Function Documentation

4.16.3.1 uint8 [wifi_get_channel](#) (void)

Get the channel number for sniffer functions.

Parameters

<i>null</i>

Returns

channel number

4.16.3.2 bool `wifi_get_country (wifi_country_t * country)`

get the current country info

Parameters

<i>wifi_country_t</i>	*country: country info
---------------------------------------	------------------------

Returns

0 : succeed
-1 : fail

4.16.3.3 void `wifi_promiscuous_enable (uint8 promiscuous)`

Enable the promiscuous mode.

Attention

1. The promiscuous mode can only be enabled in the ESP8266 station mode. Do not call this API in `user_init`.
2. When in the promiscuous mode, the ESP8266 station and soft-AP are disabled.
3. Call `wifi_station_disconnect` to disconnect before enabling the promiscuous mode.
4. Don't call any other APIs when in the promiscuous mode. Call `wifi_promiscuous_enable(0)` to quit sniffer before calling other APIs.

Parameters

<i>uint8</i>	promiscuous : <ul style="list-style-type: none"> • 0: to disable the promiscuous mode • 1: to enable the promiscuous mode
--------------	---

Returns

null

4.16.3.4 bool `wifi_promiscuous_set_mac (const uint8_t * address)`

Set the MAC address filter for the sniffer mode.

Attention

This filter works only for the current sniffer mode. If users disable and then enable the sniffer mode, and then enable sniffer, they need to set the MAC address filter again.

Parameters

<i>const</i>	uint8_t *address : MAC address
--------------	--------------------------------

Returns

true : succeed
false : fail

4.16.3.5 bool wifi_set_channel (uint8 channel)

Set the channel number for sniffer functions.

Parameters

<i>uint8</i>	channel : channel number
--------------	--------------------------

Returns

true : succeed
false : fail

4.16.3.6 bool wifi_set_country (wifi_country_t * country)

configure country info

Attention

1. The default country is {.cc="CN", .schan=1, .nchan=13, policy=WIFI_COUNTRY_POLICY_AUTO}
2. When the country policy is WIFI_COUNTRY_POLICY_AUTO, use the country info of AP to which the station is connected. E.g. if the configured country info is {.cc="USA", .schan=1, .nchan=11}, the country info of the AP to which the station is connected is {.cc="JP", .schan=1, .nchan=14}, then our country info is {.cc="JP", .schan=1, .nchan=14}. If the station disconnected from the AP, the country info back to {.cc="USA", .schan=1, .nchan=11} again.
3. When the country policy is WIFI_COUNTRY_POLICY_MANUAL, always use the configured country info.
4. When the country info is changed because of configuration or because the station connects to a different external AP, the country IE in probe response/beacon of the soft-AP is changed also.
5. The country configuration is not stored into flash

Parameters

<i>wifi_country_t</i>	*country: the configured country info
---------------------------------------	---------------------------------------

Returns

0 : succeed
-1 : fail

4.16.3.7 void wifi_set_promiscuous_rx_cb (wifi_promiscuous_cb_t cb)

Register the RX callback function in the promiscuous mode.

Each time a packet is received, the registered callback function will be called.

Parameters

<i>wifi_↔</i> <i>promiscuous_↔</i> <i>cb_t</i>	cb : callback
--	---------------

Returns

null

4.17 WPS APIs

ESP8266 WPS APIs.

Typedefs

- typedef enum wps_type **WPS_TYPE_t**
- typedef void(* **wps_st_cb_t**) (int status)
WPS callback.

Enumerations

- enum **wps_type** {
WPS_TYPE_DISABLE = 0, **WPS_TYPE_PBC**, **WPS_TYPE_PIN**, **WPS_TYPE_DISPLAY**,
WPS_TYPE_MAX }
- enum **wps_cb_status** {
WPS_CB_ST_SUCCESS = 0, **WPS_CB_ST_FAILED**, **WPS_CB_ST_TIMEOUT**, **WPS_CB_ST_WEP**,
WPS_CB_ST_SCAN_ERR }

Functions

- bool **wifi_wps_enable** (WPS_TYPE_t wps_type)
Enable Wi-Fi WPS function.
- bool **wifi_wps_disable** (void)
Disable Wi-Fi WPS function and release resource it taken.
- bool **wifi_wps_start** (void)
WPS starts to work.
- bool **wifi_set_wps_cb** (**wps_st_cb_t** cb)
Set WPS callback.

4.17.1 Detailed Description

ESP8266 WPS APIs.

WPS can only be used when ESP8266 station is enabled.

4.17.2 Typedef Documentation

4.17.2.1 typedef void(* wps_st_cb_t) (int status)

WPS callback.

Parameters

<i>int</i>	<p>status : status of WPS, enum wps_cb_status.</p> <ul style="list-style-type: none"> • If parameter status == WPS_CB_ST_SUCCESS in WPS callback, it means WPS got AP's information, user can call <code>wifi_wps_disable</code> to disable WPS and release resource, then call <code>wifi_station_connect</code> to connect to target AP. • Otherwise, it means that WPS fail, user can create a timer to retry WPS by <code>wifi_wps_start</code> after a while, or call <code>wifi_wps_disable</code> to disable WPS and release resource.
------------	---

Returns

null

4.17.3 Enumeration Type Documentation

4.17.3.1 enum wps_cb_status

Enumerator

WPS_CB_ST_SUCCESS WPS succeed

WPS_CB_ST_FAILED WPS fail

WPS_CB_ST_TIMEOUT WPS timeout, fail

WPS_CB_ST_WEP WPS failed because that WEP is not supported

WPS_CB_ST_SCAN_ERR can not find the target WPS AP

4.17.4 Function Documentation

4.17.4.1 bool wifi_set_wps_cb (wps_st_cb_t cb)

Set WPS callback.

Attention

WPS can only be used when ESP8266 station is enabled.

Parameters

<i>wps_st_cb_t</i>	cb : callback.
--------------------	----------------

Returns

true : WPS starts to work successfully, but does not mean WPS succeed.
false : fail

4.17.4.2 bool wifi_wps_disable (void)

Disable Wi-Fi WPS function and release resource it taken.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.17.4.3 bool wifi_wps_enable (WPS_TYPE_t wps_type)

Enable Wi-Fi WPS function.

Attention

WPS can only be used when ESP8266 station is enabled.

Parameters

<i>WPS_TYPE_t</i>	wps_type : WPS type, so far only WPS_TYPE_PBC is supported
-------------------	--

Returns

true : succeed
false : fail

4.17.4.4 bool wifi_wps_start (void)

WPS starts to work.

Attention

WPS can only be used when ESP8266 station is enabled.

Parameters

<i>null</i>

Returns

true : WPS starts to work successfully, but does not mean WPS succeed.
false : fail

4.18 Network Espconn APIs

Network espconn APIs.

Data Structures

- struct `_esp_tcp`
- struct `_esp_udp`
- struct `_remot_info`
- struct `espconn`

Macros

- #define `ESPCONN_OK` 0
- #define `ESPCONN_MEM` -1
- #define `ESPCONN_TIMEOUT` -3
- #define `ESPCONN_RTE` -4
- #define `ESPCONN_INPROGRESS` -5
- #define `ESPCONN_MAXNUM` -7
- #define `ESPCONN_ABRT` -8
- #define `ESPCONN_RST` -9
- #define `ESPCONN_CLSD` -10
- #define `ESPCONN_CONN` -11
- #define `ESPCONN_ARG` -12
- #define `ESPCONN_IF` -14
- #define `ESPCONN_ISCONN` -15

Typedefs

- typedef void(* `espconn_connect_callback`) (void *arg)
Connect callback.
- typedef void(* `espconn_reconnect_callback`) (void *arg, sint8 err)
Reconnect callback.
- typedef struct `_esp_tcp esp_tcp`
- typedef struct `_esp_udp esp_udp`
- typedef struct `_remot_info remot_info`
- typedef void(* `espconn_rcv_callback`) (void *arg, char *pdata, unsigned short len)
- typedef void(* `espconn_sent_callback`) (void *arg)
- typedef void(* `dns_found_callback`) (const char *name, ip_addr_t *ipaddr, void *callback_arg)
Callback which is invoked when a hostname is found.

Enumerations

- enum `espconn_type` { `ESPCONN_INVALID` = 0, `ESPCONN_TCP` = 0x10, `ESPCONN_UDP` = 0x20 }
- enum `espconn_state` { `ESPCONN_NONE`, `ESPCONN_WAIT`, `ESPCONN_LISTEN`, `ESPCONN_CONNECT`, `ESPCONN_WRITE`, `ESPCONN_READ`, `ESPCONN_CLOSE` }
- enum `espconn_option` { `ESPCONN_START` = 0x00, `ESPCONN_REUSEADDR` = 0x01, `ESPCONN_NODELAY` = 0x02, `ESPCONN_COPY` = 0x04, `ESPCONN_KEEPA_LIVE` = 0x08, `ESPCONN_END` }
- enum `espconn_level` { `ESPCONN_KEEPI_DLE`, `ESPCONN_KEEPI_NTVL`, `ESPCONN_KEEPI_CNT` }
- enum { `ESPCONN_IDLE` = 0, `ESPCONN_CLIENT`, `ESPCONN_SERVER`, `ESPCONN_BOTH`, `ESPCONN_MAX` }

Functions

- void `espconn_init` (void)
espconn initialization.
- sint8 `espconn_connect` (struct `espconn` *`espconn`)
Connect to a TCP server (ESP8266 acting as TCP client).
- sint8 `espconn_disconnect` (struct `espconn` *`espconn`)
Disconnect a TCP connection.
- sint8 `espconn_delete` (struct `espconn` *`espconn`)
Delete a transmission.
- sint8 `espconn_accept` (struct `espconn` *`espconn`)
Creates a TCP server (i.e. accepts connections).
- sint8 `espconn_create` (struct `espconn` *`espconn`)
Create UDP transmission.
- uint8 `espconn_tcp_get_max_con` (void)
Get maximum number of how many TCP connections are allowed.
- sint8 `espconn_tcp_set_max_con` (uint8 num)
Set the maximum number of how many TCP connection is allowed.
- sint8 `espconn_tcp_get_max_con_allow` (struct `espconn` *`espconn`)
Get the maximum number of TCP clients which are allowed to connect to ESP8266 TCP server.
- sint8 `espconn_tcp_set_max_con_allow` (struct `espconn` *`espconn`, uint8 num)
Set the maximum number of TCP clients allowed to connect to ESP8266 TCP server.
- sint8 `espconn_regist_time` (struct `espconn` *`espconn`, uint32 interval, uint8 type_flag)
Register timeout interval of ESP8266 TCP server.
- sint8 `espconn_get_connection_info` (struct `espconn` *`pespconn`, `remot_info` **`pcon_info`, uint8 typeflags)
Get the information about a TCP connection or UDP transmission.
- sint8 `espconn_regist_sentcb` (struct `espconn` *`espconn`, `espconn_sent_callback` `sent_cb`)
Register data sent callback which will be called back when data are successfully sent.
- sint8 `espconn_regist_write_finish` (struct `espconn` *`espconn`, `espconn_connect_callback` `write_finish_fn`)
Register a callback which will be called when all sending TCP data is completely write into write-buffer or sent.
- sint8 `espconn_send` (struct `espconn` *`espconn`, uint8 *`psent`, uint16 length)
Send data through network.
- sint8 `espconn_sent` (struct `espconn` *`espconn`, uint8 *`psent`, uint16 length)
Send data through network.
- sint16 `espconn_sendto` (struct `espconn` *`espconn`, uint8 *`psent`, uint16 length)
Send UDP data.
- sint8 `espconn_regist_connectcb` (struct `espconn` *`espconn`, `espconn_connect_callback` `connect_cb`)
Register connection function which will be called back under successful TCP connection.
- sint8 `espconn_regist_recvcb` (struct `espconn` *`espconn`, `espconn_recv_callback` `recv_cb`)
register data receive function which will be called back when data are received.
- sint8 `espconn_regist_reconcb` (struct `espconn` *`espconn`, `espconn_reconnect_callback` `recon_cb`)
Register reconnect callback.
- sint8 `espconn_regist_disconcb` (struct `espconn` *`espconn`, `espconn_connect_callback` `discon_cb`)
Register disconnection function which will be called back under successful TCP disconnection.
- uint32 `espconn_port` (void)
Get an available port for network.
- sint8 `espconn_set_opt` (struct `espconn` *`espconn`, uint8 opt)
Set option of TCP connection.
- sint8 `espconn_clear_opt` (struct `espconn` *`espconn`, uint8 opt)
Clear option of TCP connection.
- sint8 `espconn_set_keepalive` (struct `espconn` *`espconn`, uint8 level, void *`optarg`)

- Set configuration of TCP keep alive.*

 - sint8 `espconn_get_keepalive` (struct `espconn` *`espconn`, uint8 level, void *`optarg`)

Get configuration of TCP keep alive.
- err_t `espconn_gethostbyname` (struct `espconn` *`pespconn`, const char *`hostname`, ip_addr_t *`addr`, `dns_found_callback` found)

DNS function.
- sint8 `espconn_igmp_join` (ip_addr_t *`host_ip`, ip_addr_t *`multicast_ip`)

Join a multicast group.
- sint8 `espconn_igmp_leave` (ip_addr_t *`host_ip`, ip_addr_t *`multicast_ip`)

Leave a multicast group.
- sint8 `espconn_recv_hold` (struct `espconn` *`pespconn`)

Puts in a request to block the TCP receive function.
- sint8 `espconn_recv_unhold` (struct `espconn` *`pespconn`)

Unblock TCP receiving data (i.e. undo `espconn_recv_hold`).
- void `espconn_dns_setserver` (char numdns, ip_addr_t *`dnserver`)

Set default DNS server. Two DNS server is allowed to be set.

4.18.1 Detailed Description

Network espconn APIs.

4.18.2 Macro Definition Documentation

4.18.2.1 #define ESPCONN_ABRT -8

Connection aborted.

4.18.2.2 #define ESPCONN_ARG -12

Illegal argument.

4.18.2.3 #define ESPCONN_CLSD -10

Connection closed.

4.18.2.4 #define ESPCONN_CONN -11

Not connected.

4.18.2.5 #define ESPCONN_IF -14

UDP send error.

4.18.2.6 #define ESPCONN_INPROGRESS -5

Operation in progress.

4.18.2.7 #define ESPCONN_ISCONN -15

Already connected.

4.18.2.8 #define ESPCONN_MAXNUM -7

Total number exceeds the maximum limitation.

4.18.2.9 #define ESPCONN_MEM -1

Out of memory.

4.18.2.10 #define ESPCONN_OK 0

No error, everything OK.

4.18.2.11 #define ESPCONN_RST -9

Connection reset.

4.18.2.12 #define ESPCONN_RTE -4

Routing problem.

4.18.2.13 #define ESPCONN_TIMEOUT -3

Timeout.

4.18.3 Typedef Documentation

4.18.3.1 typedef void(* dns_found_callback) (const char *name, ip_addr_t *ipaddr, void *callback_arg)

Callback which is invoked when a hostname is found.

Parameters

<i>const</i>	char *name : hostname
<i>ip_addr_t</i>	*ipaddr : IP address of the hostname, or to be NULL if the name could not be found (or on any other error).
<i>void</i>	*callback_arg : callback argument.

Returns

null

4.18.3.2 typedef void(* espconn_connect_callback) (void *arg)

Connect callback.

Callback which will be called if successful listening (ESP8266 as TCP server) or connection (ESP8266 as TCP client) callback, register by espconn_regist_connectcb.

Attention

The pointer "void *arg" may be different in different callbacks, please don't use this pointer directly to distinguish one from another in multiple connections, use remote_ip and remote_port in espconn instead.

Parameters

<i>void</i>	*arg : pointer corresponding structure espconn.
-------------	---

Returns

null

4.18.3.3 typedef void(* espconn_reconnect_callback) (void *arg, sint8 err)

Reconnect callback.

Enter this callback when error occurred, TCP connection broke. This callback is registered by espconn_regist_reconcb.

Attention

The pointer "void *arg" may be different in different callbacks, please don't use this pointer directly to distinguish one from another in multiple connections, use remote_ip and remote_port in espconn instead.

Parameters

<i>void</i>	*arg : pointer corresponding structure espconn.
<i>sint8</i>	err : error code <ul style="list-style-type: none"> • ESPCONN_TIMEOUT - Timeout • ESPCONN_ABRT - TCP connection aborted • ESPCONN_RST - TCP connection abort • ESPCONN_CLSD - TCP connection closed • ESPCONN_CONN - TCP connection • ESPCONN_HANDSHAKE - TCP SSL handshake fail • ESPCONN_PROTO_MSG - SSL application invalid

Returns

null

4.18.3.4 typedef void(* espconn_rcv_callback) (void *arg, char *pdata, unsigned short len)

A callback prototype to inform about events for a espconn

4.18.4 Enumeration Type Documentation

4.18.4.1 enum espconn_level

Enumerator

ESPCONN_KEEPIDLE TCP keep-alive interval, unit : second.

ESPCONN_KEEPINTVL packet interval during TCP keep-alive, unit : second.

ESPCONN_KEEPCNT maximum packet retry count of TCP keep-alive.

4.18.4.2 enum espconn_option

Enumerator

ESPCONN_START no option, start enum.

ESPCONN_REUSEADDR free memory after TCP disconnection happen, need not wait 2 minutes.

ESPCONN_NODELAY disable nagle algorithm during TCP data transmission, quicken the data transmission.

ESPCONN_COPY enable espconn_regist_write_finish, enter write_finish_callback means that the data espconn_send sending was written into 2920 bytes write-buffer waiting for sending or already sent.

ESPCONN_KEEPAIVE enable TCP keep alive.

ESPCONN_END no option, end enum.

4.18.4.3 enum espconn_state

Current state of the espconn.

Enumerator

ESPCONN_NONE idle state, no connection

ESPCONN_WAIT ESP8266 is as TCP client, and waiting for connection

ESPCONN_LISTEN ESP8266 is as TCP server, and waiting for connection

ESPCONN_CONNECT connected

ESPCONN_WRITE sending data

ESPCONN_READ receiving data

ESPCONN_CLOSE connection closed

4.18.4.4 enum espconn_type

Protocol family and type of the espconn

Enumerator

ESPCONN_INVALID invalid type

ESPCONN_TCP TCP

ESPCONN_UDP UDP

4.18.5 Function Documentation

4.18.5.1 sint8 espconn_accept (struct espconn * espconn)

Creates a TCP server (i.e. accepts connections).

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.2 `sint8 espconn_clear_opt (struct espconn * espconn, uint8 opt)`

Clear option of TCP connection.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>uint8</i>	opt : enum espconn_option

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.3 `sint8 espconn_connect (struct espconn * espconn)`

Connect to a TCP server (ESP8266 acting as TCP client).

Attention

If espconn_connect fail, returns non-0 value, there is no connection, so it won't enter any espconn callback.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure, the espconn to listen to the connection
---------------	--

Returns

0 : succeed

Non-0 : error code

- ESPCONN_RTE - Routing Problem
- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.4 `sint8 espconn_create (struct espconn * espconn)`

Create UDP transmission.

Attention

Parameter remote_ip and remote_port need to be set, do not set to be 0.

Parameters

<i>struct</i>	espconn *espconn : the UDP control block structure
---------------	--

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ISCONN - Already connected
- ESPCONN_ARG - illegal argument, can't find the corresponding UDP transmission according to structure espconn

4.18.5.5 sint8 espconn_delete (struct espconn * espconn)

Delete a transmission.

Attention

Corresponding creation API :

- TCP: espconn_accept,
- UDP: espconn_create

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding network according to structure espconn
- ESPCONN_INPROGRESS - the connection is still in progress, please call espconn_disconnect to disconnect before delete it.

4.18.5.6 sint8 espconn_disconnect (struct espconn * espconn)

Disconnect a TCP connection.

Attention

Don't call this API in any espconn callback. If needed, please use system task to trigger espconn_disconnect.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.7 void espconn_dns_setserver (char numdns, ip_addr_t * dnsserver)

Set default DNS server. Two DNS server is allowed to be set.

Attention

Only if ESP8266 DHCP client is disabled (wifi_station_dhcpc_stop), this API can be used.

Parameters

<i>char</i>	numdns : DNS server ID, 0 or 1
<i>ip_addr_t</i>	*dnsserver : DNS server IP

Returns

null

4.18.5.8 `sint8 espconn_get_connection_info (struct espconn * pespconn, remot_info ** pcon_info, uint8 typeflags)`

Get the information about a TCP connection or UDP transmission.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
<i>remot_info</i>	**pcon_info : connect to client info
<i>uint8</i>	typeflags : 0, regular server; 1, ssl server

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding transmission according to structure espconn

4.18.5.9 `sint8 espconn_get_keepalive (struct espconn * espconn, uint8 level, void * optarg)`

Get configuration of TCP keep alive.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>uint8</i>	level : enum espconn_level
<i>void*</i>	optarg : value of parameter

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.10 `err_t espconn_gethostbyname (struct espconn * pespconn, const char * hostname, ip_addr_t * addr, dns_found_callback found)`

DNS function.

Parse a hostname (string) to an IP address.

Parameters

<i>struct</i>	espconn *pespconn : espconn to parse a hostname.
---------------	--

<i>const</i>	char *hostname : the hostname.
<i>ip_addr_t</i>	*addr : IP address.
<i>dns_found_↔ callback</i>	found : callback of DNS

Returns

err_t :

- ESPCONN_OK - succeed
- ESPCONN_INPROGRESS - error code : already connected
- ESPCONN_ARG - error code : illegal argument, can't find network transmission according to structure espconn

4.18.5.11 `sint8 espconn_igmp_join (ip_addr_t * host_ip, ip_addr_t * multicast_ip)`

Join a multicast group.

Attention

This API can only be called after the ESP8266 station connects to a router.

Parameters

<i>ip_addr_t</i>	*host_ip : IP of UDP host
<i>ip_addr_t</i>	*multicast_ip : IP of multicast group

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory

4.18.5.12 `sint8 espconn_igmp_leave (ip_addr_t * host_ip, ip_addr_t * multicast_ip)`

Leave a multicast group.

Attention

This API can only be called after the ESP8266 station connects to a router.

Parameters

<i>ip_addr_t</i>	*host_ip : IP of UDP host
<i>ip_addr_t</i>	*multicast_ip : IP of multicast group

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory

4.18.5.13 void espconn_init (void)

espconn initialization.

Attention

Please call this API in user_init, if you need to use espconn functions.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.18.5.14 uint32 espconn_port (void)

Get an available port for network.

Parameters

<i>null</i>	
-------------	--

Returns

Port number.

4.18.5.15 sint8 espconn_recv_hold (struct espconn * pespconn)

Puts in a request to block the TCP receive function.

Attention

The function does not act immediately; we recommend calling it while reserving 5*1460 bytes of memory. This API can be called more than once.

Parameters

<i>struct</i>	espconn *espconn : corresponding TCP connection structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn.

4.18.5.16 sint8 espconn_recv_unhold (struct espconn * pespconn)

Unblock TCP receiving data (i.e. undo espconn_recv_hold).

Attention

This API takes effect immediately.

Parameters

<i>struct</i>	espconn *espconn : corresponding TCP connection structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn.

4.18.5.17 `sint8 espconn_regist_connectcb (struct espconn * espconn, espconn_connect_callback connect_cb)`

Register connection function which will be called back under successful TCP connection.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>espconn_↔</i> <i>connect_↔</i> <i>callback</i>	connect_cb : registered callback function

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.18 `sint8 espconn_regist_disconcb (struct espconn * espconn, espconn_connect_callback discon_cb)`

Register disconnection function which will be called back under successful TCP disconnection.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>espconn_↔</i> <i>connect_↔</i> <i>callback</i>	discon_cb : registered callback function

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.19 `sint8 espconn_regist_reconcb (struct espconn * espconn, espconn_reconnect_callback recon_cb)`

Register reconnect callback.

Attention

espconn_reconnect_callback is more like a network-broken error handler; it handles errors that occurs in any phase of the connection. For instance, if espconn_send fails, espconn_reconnect_callback will be called because the network is broken.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>espconn_↔</i> <i>reconnect_↔</i> <i>callback</i>	recon_cb : registered callback function

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.20 `sint8 espconn_regist_recvcb (struct espconn * espconn, espconn_recv_callback recv_cb)`

register data receive function which will be called back when data are received.

Parameters

<i>struct</i>	espconn *espconn : the network transmission structure
<i>espconn_recv_↔</i> <i>_callback</i>	recv_cb : registered callback function

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.21 `sint8 espconn_regist_sentcb (struct espconn * espconn, espconn_sent_callback sent_cb)`

Register data sent callback which will be called back when data are successfully sent.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
<i>espconn_sent_↔</i> <i>_callback</i>	sent_cb : registered callback function which will be called if the data is successfully sent

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding transmission according to structure espconn

4.18.5.22 `sint8 espconn_regist_time (struct espconn * espconn, uint32 interval, uint8 type_flag)`

Register timeout interval of ESP8266 TCP server.

Attention

1. If timeout is set to 0, timeout will be disable and ESP8266 TCP server will not disconnect TCP clients has stopped communication. This usage of timeout=0, is deprecated.
2. This timeout interval is not very precise, only as reference.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>uint32</i>	interval : timeout interval, unit: second, maximum: 7200 seconds
<i>uint8</i>	type_flag : 0, set for all connections; 1, set for a specific connection <ul style="list-style-type: none"> • If the type_flag set to be 0, please call this API after espconn_accept, before listened a TCP connection. • If the type_flag set to be 1, the first parameter *espconn is the specific connection.

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.23 sint8 espconn_regist_write_finish (struct espconn * espconn, espconn_connect_callback write_finish_fn)

Register a callback which will be called when all sending TCP data is completely write into write-buffer or sent.

Need to call espconn_set_opt to enable write-buffer first.

Attention

1. write-buffer is used to keep TCP data that waiting to be sent, queue number of the write-buffer is 8 which means that it can keep 8 packets at most. The size of write-buffer is 2920 bytes.
2. Users can enable it by using espconn_set_opt.
3. Users can call espconn_send to send the next packet in write_finish_callback instead of using espconn_send_callback.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
<i>espconn_callback</i>	write_finish_fn : registered callback function which will be called if the data is completely write into write buffer or sent.

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.24 sint8 espconn_send (struct espconn * espconn, uint8 * psent, uint16 length)

Send data through network.

Attention

1. Please call espconn_send after espconn_send_callback of the pre-packet.
2. If it is a UDP transmission, it is suggested to set espconn->proto.udp->remote_ip and remote_port before every calling of espconn_send.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
<i>uint8</i>	*psent : pointer of data
<i>uint16</i>	length : data length

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ARG - illegal argument, can't find the corresponding network transmission according to structure espconn
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.18.5.25 `sint16 espconn_sendto (struct espconn * espconn, uint8 * psent, uint16 length)`

Send UDP data.

Parameters

<i>struct</i>	espconn *espconn : the UDP structure
<i>uint8</i>	*psent : pointer of data
<i>uint16</i>	length : data length

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.18.5.26 `sint8 espconn_sent (struct espconn * espconn, uint8 * psent, uint16 length)`

Send data through network.

This API is deprecated, please use espconn_send instead.

Attention

1. Please call espconn_sent after espconn_sent_callback of the pre-packet.
2. If it is a UDP transmission, it is suggested to set espconn->proto.udp->remote_ip and remote_port before every calling of espconn_sent.

Parameters

<i>struct</i>	espconn *espconn : the network connection structure
<i>uint8</i>	*psent : pointer of data

<i>uint16</i>	length : data length
---------------	----------------------

Returns

0 : succeed

Non-0 : error code

- ESPCONN_MEM - Out of memory
- ESPCONN_ARG - illegal argument, can't find the corresponding network transmission according to structure `espcconn`
- ESPCONN_MAXNUM - buffer of sending data is full
- ESPCONN_IF - send UDP data fail

4.18.5.27 `sint8 espcconn_set_keepalive (struct espcconn * espcconn, uint8 level, void * optarg)`

Set configuration of TCP keep alive.

Attention

In general, we need not call this API. If needed, please call it in `espcconn_connect_callback` and call `espcconn_set_opt` to enable keep alive first.

Parameters

<i>struct</i>	<code>espcconn *espcconn</code> : the TCP connection structure
<i>uint8</i>	<code>level</code> : To do TCP keep-alive detection every <code>ESPCONN_KEEPIDLE</code> . If there is no response, retry <code>ESPCONN_KEEPCNT</code> times every <code>ESPCONN_KEEPINTVL</code> . If still no response, considers it as TCP connection broke, goes into <code>espcconn_reconnect_callback</code> . Notice, keep alive interval is not precise, only for reference, it depends on priority.
<i>void*</i>	<code>optarg</code> : value of parameter

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure `espcconn`

4.18.5.28 `sint8 espcconn_set_opt (struct espcconn * espcconn, uint8 opt)`

Set option of TCP connection.

Attention

In general, we need not call this API. If call `espconn_set_opt`, please call it in `espconn_connect_callback`.

Parameters

<i>struct</i>	espconn *espconn : the TCP connection structure
<i>uint8</i>	opt : option of TCP connection, refer to enum <code>espconn_option</code> <ul style="list-style-type: none"> • bit 0: 1: free memory after TCP disconnection happen need not wait 2 minutes; • bit 1: 1: disable nagle algorithm during TCP data transmission, quiken the data transmission. • bit 2: 1: enable <code>espconn_regist_write_finish</code>, enter write finish callback means the data <code>espconn_send</code> sending was written into 2920 bytes write-buffer waiting for sending or already sent. • bit 3: 1: enable TCP keep alive

Returns

0 : succeed

Non-0 : error code

- `ESPCONN_ARG` - illegal argument, can't find the corresponding TCP connection according to structure `espconn`

4.18.5.29 `uint8 espconn_tcp_get_max_con (void)`

Get maximum number of how many TCP connections are allowed.

Parameters

<i>null</i>	
-------------	--

Returns

Maximum number of how many TCP connections are allowed.

4.18.5.30 `sint8 espconn_tcp_get_max_con_allow (struct espconn * espconn)`

Get the maximum number of TCP clients which are allowed to connect to ESP8266 TCP server.

Parameters

<i>struct</i>	espconn *espconn : the TCP server structure
---------------	---

Returns

0 : succeed

Non-0 : error code

- `ESPCONN_ARG` - illegal argument, can't find the corresponding TCP connection according to structure `espconn`

4.18.5.31 `sint8 espconn_tcp_set_max_con (uint8 num)`

Set the maximum number of how many TCP connection is allowed.

Parameters

<i>uint8</i>	num : Maximum number of how many TCP connection is allowed.
--------------	---

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.18.5.32 `sint8 espconn_tcp_set_max_con_allow (struct espconn * espconn, uint8 num)`

Set the maximum number of TCP clients allowed to connect to ESP8266 TCP server.

Parameters

<i>struct</i>	espconn *espconn : the TCP server structure
<i>uint8</i>	num : Maximum number of TCP clients which are allowed

Returns

0 : succeed

Non-0 : error code

- ESPCONN_ARG - illegal argument, can't find the corresponding TCP connection according to structure espconn

4.19 ESP-NOW APIs

ESP-NOW APIs.

Typedefs

- typedef void(* [esp_now_rcv_cb_t](#)) (uint8 *mac_addr, uint8 *data, uint8 len)
ESP-NOW send callback.
- typedef void(* [esp_now_send_cb_t](#)) (uint8 *mac_addr, uint8 status)
ESP-NOW send callback.

Enumerations

- enum [esp_now_role](#) { [ESP_NOW_ROLE_IDLE](#) = 0, [ESP_NOW_ROLE_CONTROLLER](#), [ESP_NOW_ROLE_SLAVE](#), [ESP_NOW_ROLE_MAX](#) }

Functions

- sint32 [esp_now_init](#) (void)
ESP-NOW initialization.
- sint32 [esp_now_deinit](#) (void)
Deinitialize ESP-NOW.
- sint32 [esp_now_register_send_cb](#) ([esp_now_send_cb_t](#) cb)
Register ESP-NOW send callback.
- sint32 [esp_now_unregister_send_cb](#) (void)
Unregister ESP-NOW send callback.
- sint32 [esp_now_register_rcv_cb](#) ([esp_now_rcv_cb_t](#) cb)
Register ESP-NOW receive callback.
- sint32 [esp_now_unregister_rcv_cb](#) (void)
Unregister ESP-NOW receive callback.
- sint32 [esp_now_send](#) (uint8 *mac_addr, uint8 *data, uint8 len)
Send ESP-NOW packet.
- sint32 [esp_now_add_peer](#) (uint8 *mac_addr, uint8 role, uint8 channel, uint8 *key, uint8 key_len)
Add an ESP-NOW peer, store MAC address of target device into ESP-NOW MAC list.
- sint32 [esp_now_del_peer](#) (uint8 *mac_addr)
Delete an ESP-NOW peer, delete MAC address of the device from ESP-NOW MAC list.
- sint32 [esp_now_set_self_role](#) (uint8 role)
Set ESP-NOW role of device itself.
- sint32 [esp_now_get_self_role](#) (void)
Get ESP-NOW role of device itself.
- sint32 [esp_now_set_peer_role](#) (uint8 *mac_addr, uint8 role)
Set ESP-NOW role for a target device. If it is set multiple times, new role will cover the old one.
- sint32 [esp_now_get_peer_role](#) (uint8 *mac_addr)
Get ESP-NOW role of a target device.
- sint32 [esp_now_set_peer_channel](#) (uint8 *mac_addr, uint8 channel)
Record channel information of a ESP-NOW device.
- sint32 [esp_now_get_peer_channel](#) (uint8 *mac_addr)
Get channel information of a ESP-NOW device.
- sint32 [esp_now_set_peer_key](#) (uint8 *mac_addr, uint8 *key, uint8 key_len)
Set ESP-NOW key for a target device.

- sint32 `esp_now_get_peer_key` (uint8 *mac_addr, uint8 *key, uint8 *key_len)
Get ESP-NOW key of a target device.
- uint8 * `esp_now_fetch_peer` (bool restart)
Get MAC address of ESP-NOW device.
- sint32 `esp_now_is_peer_exist` (uint8 *mac_addr)
Check if target device exists or not.
- sint32 `esp_now_get_cnt_info` (uint8 *all_cnt, uint8 *encrypt_cnt)
Get the total number of ESP-NOW devices which are associated, and the number count of encrypted devices.
- sint32 `esp_now_set_kok` (uint8 *key, uint8 len)
Set the encrypt key of communication key.

4.19.1 Detailed Description

ESP-NOW APIs.

Attention

1. ESP-NOW do not support broadcast and multicast.
2. ESP-NOW is targeted to Smart-Light project, so it is suggested that slave role corresponding to soft-AP or soft-AP+station mode, controller role corresponding to station mode.
3. When ESP8266 is in soft-AP+station mode, it will communicate through station interface if it is in slave role, and communicate through soft-AP interface if it is in controller role.
4. ESP-NOW can not wake ESP8266 up from sleep, so if the target ESP8266 station is in sleep, ESP-NOW communication will fail.
5. In station mode, ESP8266 supports 10 encrypt ESP-NOW peers at most, with the unencrypted peers, it can be 20 peers in total at most.
6. In the soft-AP mode or soft-AP + station mode, the ESP8266 supports 6 encrypt ESP-NOW peers at most, with the unencrypted peers, it can be 20 peers in total at most.

4.19.2 Typedef Documentation

4.19.2.1 typedef void(* esp_now_rcv_cb_t) (uint8 *mac_addr, uint8 *data, uint8 len)

ESP-NOW send callback.

Attention

The status will be OK, if ESP-NOW send packet successfully. But users need to make sure by themselves that key of communication is correct.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device
<i>uint8</i>	*data : data received
<i>uint8</i>	len : data length

Returns

null

4.19.2.2 typedef void(* esp_now_send_cb_t) (uint8 *mac_addr, uint8 status)

ESP-NOW send callback.

Attention

The status will be OK, if ESP-NOW send packet successfully. But users need to make sure by themselves that key of communication is correct.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device
<i>uint8</i>	status : status of ESP-NOW sending packet, 0, OK; 1, fail.

Returns

null

4.19.3 Function Documentation

4.19.3.1 sint32 esp_now_add_peer (uint8 * mac_addr, uint8 role, uint8 channel, uint8 * key, uint8 key_len)

Add an ESP-NOW peer, store MAC address of target device into ESP-NOW MAC list.

Parameters

<i>uint8</i>	*mac_addr : MAC address of device
<i>uint8</i>	role : role type of device, enum esp_now_role
<i>uint8</i>	channel : channel of device
<i>uint8</i>	*key : 16 bytes key which is needed for ESP-NOW communication
<i>uint8</i>	key_len : length of key, has to be 16 bytes now

Returns

0 : succeed
Non-0 : fail

4.19.3.2 sint32 esp_now_deinit (void)

Deinitialize ESP-NOW.

Parameters

<i>null</i>	
-------------	--

Returns

0 : succeed
Non-0 : fail

4.19.3.3 sint32 esp_now_del_peer (uint8 * mac_addr)

Delete an ESP-NOW peer, delete MAC address of the device from ESP-NOW MAC list.

Parameters

<i>u8</i>	*mac_addr : MAC address of device
-----------	-----------------------------------

Returns

0 : succeed
Non-0 : fail

4.19.3.4 `uint8* esp_now_fetch_peer (bool restart)`

Get MAC address of ESP-NOW device.

Get MAC address of ESP-NOW device which is pointed now, and move the pointer to next one in ESP-NOW MAC list or move the pointer to the first one in ESP-NOW MAC list.

Attention

1. This API can not re-entry
2. Parameter has to be true when you call it the first time.

Parameters

<i>bool</i>	<code>restart</code> : true, move pointer to the first one in ESP-NOW MAC list; false, move pointer to the next one in ESP-NOW MAC list
-------------	---

Returns

NULL, no ESP-NOW devices exist
 Otherwise, MAC address of ESP-NOW device which is pointed now

4.19.3.5 `sint32 esp_now_get_cnt_info (uint8 * all_cnt, uint8 * encrypt_cnt)`

Get the total number of ESP-NOW devices which are associated, and the number count of encrypted devices.

Parameters

<i>uint8</i>	<code>*all_cnt</code> : total number of ESP-NOW devices which are associated.
<i>uint8</i>	<code>*encrypt_cnt</code> : number count of encrypted devices

Returns

0 : succeed
 Non-0 : fail

4.19.3.6 `sint32 esp_now_get_peer_channel (uint8 * mac_addr)`

Get channel information of a ESP-NOW device.

Attention

ESP-NOW communication needs to be at the same channel.

Parameters

<i>uint8</i>	<code>*mac_addr</code> : MAC address of target device.
--------------	--

Returns

1 ~ 13 (some area may get 14) : channel number
 Non-0 : fail

4.19.3.7 `sint32 esp_now_get_peer_key (uint8 * mac_addr, uint8 * key, uint8 * key_len)`

Get ESP-NOW key of a target device.

If it is set multiple times, new key will cover the old one.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device.
<i>uint8</i>	*key : pointer of key, buffer size has to be 16 bytes at least
<i>uint8</i>	key_len : key length

Returns

0 : succeed
 > 0 : find target device but can't get key
 < 0 : fail

4.19.3.8 sint32 esp_now_get_peer_role (uint8 * mac_addr)

Get ESP-NOW role of a target device.

Parameters

<i>uint8</i>	*mac_addr : MAC address of device.
--------------	------------------------------------

Returns

ESP_NOW_ROLE_CONTROLLER, role type : controller
 ESP_NOW_ROLE_SLAVE, role type : slave
 otherwise : fail

4.19.3.9 sint32 esp_now_get_self_role (void)

Get ESP-NOW role of device itself.

Parameters

<i>uint8</i>	role : role type of device, enum esp_now_role.
--------------	--

Returns

0 : succeed
 Non-0 : fail

4.19.3.10 sint32 esp_now_init (void)

ESP-NOW initialization.

Parameters

<i>null</i>	
-------------	--

Returns

0 : succeed
 Non-0 : fail

4.19.3.11 sint32 esp_now_is_peer_exist (uint8 * mac_addr)

Check if target device exists or not.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device.
--------------	---

Returns

- 0 : device does not exist
- < 0 : error occur, check fail
- > 0 : device exists

4.19.3.12 sint32 esp_now_register_rcv_cb (esp_now_rcv_cb_t cb)

Register ESP-NOW receive callback.

Parameters

<i>esp_now_rcv← _cb_t</i>	cb : receive callback
-------------------------------	-----------------------

Returns

- 0 : succeed
- Non-0 : fail

4.19.3.13 sint32 esp_now_register_send_cb (esp_now_send_cb_t cb)

Register ESP-NOW send callback.

Parameters

<i>esp_now← send_cb_t</i>	cb : send callback
-------------------------------	--------------------

Returns

- 0 : succeed
- Non-0 : fail

4.19.3.14 sint32 esp_now_send (uint8 * da, uint8 * data, uint8 len)

Send ESP-NOW packet.

Parameters

<i>uint8</i>	*da : destination MAC address. If it's NULL, send packet to all MAC addresses recorded by ESP-NOW; otherwise, send packet to target MAC address.
<i>uint8</i>	*data : data need to send
<i>uint8</i>	len : data length

Returns

- 0 : succeed
- Non-0 : fail

4.19.3.15 sint32 esp_now_set_kok (uint8 * key, uint8 len)

Set the encrypt key of communication key.

All ESP-NOW devices share the same encrypt key. If users do not set the encrypt key, ESP-NOW communication key will be encrypted by a default key.

Parameters

<i>uint8</i>	*key : pointer of encrypt key.
<i>uint8</i>	len : key length, has to be 16 bytes now.

Returns

0 : succeed
Non-0 : fail

4.19.3.16 `sint32 esp_now_set_peer_channel (uint8 * mac_addr, uint8 channel)`

Record channel information of a ESP-NOW device.

When communicate with this device,

- call `esp_now_get_peer_channel` to get its channel first,
- then call `wifi_set_channel` to be in the same channel and do communication.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device.
<i>uint8</i>	channel : channel, usually to be 1 ~ 13, some area may use channel 14.

Returns

0 : succeed
Non-0 : fail

4.19.3.17 `sint32 esp_now_set_peer_key (uint8 * mac_addr, uint8 * key, uint8 key_len)`

Set ESP-NOW key for a target device.

If it is set multiple times, new key will cover the old one.

Parameters

<i>uint8</i>	*mac_addr : MAC address of target device.
<i>uint8</i>	*key : 16 bytes key which is needed for ESP-NOW communication, if it is NULL, current key will be reset to be none.
<i>uint8</i>	key_len : key length, has to be 16 bytes now

Returns

0 : succeed
Non-0 : fail

4.19.3.18 `sint32 esp_now_set_peer_role (uint8 * mac_addr, uint8 role)`

Set ESP-NOW role for a target device. If it is set multiple times, new role will cover the old one.

Parameters

<i>uint8</i>	*mac_addr : MAC address of device.
<i>uint8</i>	role : role type, enum esp_now_role.

Returns

0 : succeed
Non-0 : fail

4.19.3.19 sint32 esp_now_set_self_role (uint8 role)

Set ESP-NOW role of device itself.

Parameters

<i>uint8</i>	role : role type of device, enum esp_now_role.
--------------	--

Returns

0 : succeed
Non-0 : fail

4.19.3.20 sint32 esp_now_unregister_rcv_cb (void)

Unregister ESP-NOW receive callback.

Parameters

<i>null</i>	
-------------	--

Returns

0 : succeed
Non-0 : fail

4.19.3.21 sint32 esp_now_unregister_send_cb (void)

Unregister ESP-NOW send callback.

Parameters

<i>null</i>	
-------------	--

Returns

0 : succeed
Non-0 : fail

4.20 Driver APIs

Driver APIs.

Modules

- [PWM Driver APIs](#)
PWM driver APIs.
- [SPI Driver APIs](#)
SPI Flash APIs.

4.20.1 Detailed Description

Driver APIs.

4.21 PWM Driver APIs

PWM driver APIs.

Data Structures

- struct [pwm_param](#)

Macros

- #define **PWM_DEPTH** 1023

Functions

- void [pwm_init](#) (uint32 period, uint32 *duty, uint32 pwm_channel_num, uint32(*pin_info_list)[3])
PWM function initialization, including GPIO, frequency and duty cycle.
- void [pwm_set_duty](#) (uint32 duty, uint8 channel)
Set the duty cycle of a PWM channel.
- uint32 [pwm_get_duty](#) (uint8 channel)
Get the duty cycle of a PWM channel.
- void [pwm_set_period](#) (uint32 period)
Set PWM period, unit : us.
- uint32 [pwm_get_period](#) (void)
Get PWM period, unit : us.
- void [pwm_start](#) (void)
Starts PWM.

4.21.1 Detailed Description

PWM driver APIs.

4.21.2 Function Documentation

4.21.2.1 uint32 pwm_get_duty (uint8 channel)

Get the duty cycle of a PWM channel.

Parameters

<i>uint8</i>	channel : PWM channel number
--------------	------------------------------

Returns

Duty cycle of PWM output.

4.21.2.2 uint32 pwm_get_period (void)

Get PWM period, unit : us.

Parameters

<i>null</i>

Returns

PWM period, unit : us.

4.21.2.3 void `pwm_init (uint32 period, uint32 * duty, uint32 pwm_channel_num, uint32(*) pin_info_list[3])`

PWM function initialization, including GPIO, frequency and duty cycle.

Attention

This API can be called only once.

Parameters

<i>uint32</i>	<code>period</code> : pwm frequency
<i>uint32</i>	<code>*duty</code> : duty cycle
<i>uint32</i>	<code>pwm_channel_num</code> : PWM channel number
<i>uint32</i>	<code>(*pin_info_list)[3]</code> : GPIO parameter of PWM channel, it is a pointer of n x 3 array which defines GPIO register, IO reuse of corresponding pin and GPIO number.

Returns

null

4.21.2.4 void `pwm_set_duty (uint32 duty, uint8 channel)`

Set the duty cycle of a PWM channel.

Set the time that high level signal will last, duty depends on period, the maximum value can be 1023.

Attention

After set configuration, `pwm_start` needs to be called to take effect.

Parameters

<i>uint32</i>	<code>duty</code> : duty cycle
<i>uint8</i>	<code>channel</code> : PWM channel number

Returns

null

4.21.2.5 void `pwm_set_period (uint32 period)`

Set PWM period, unit : us.

For example, for 1KHz PWM, period is 1000 us.

Attention

After set configuration, `pwm_start` needs to be called to take effect.

Parameters

<i>uint32</i>	period : PWM period, unit : us.
---------------	---------------------------------

Returns

null

4.21.2.6 void pwm_start (void)

Starts PWM.

Attention

This function needs to be called after PWM configuration is changed.

Parameters

<i>null</i>	
-------------	--

Returns

null

4.22 Smartconfig APIs

SmartConfig APIs.

Typedefs

- typedef void(* [sc_callback_t](#)) ([sc_status](#) status, void *pdata)

The callback of SmartConfig, executed when smart-config status changed.

Enumerations

- enum [sc_status](#) {
[SC_STATUS_WAIT](#) = 0, [SC_STATUS_FIND_CHANNEL](#), [SC_STATUS_GETTING_SSID_PSWD](#), [SC_STATUS_LINK](#),
[SC_STATUS_LINK_OVER](#) }
- enum [sc_type](#) { [SC_TYPE_ESPTOUCH](#) = 0, [SC_TYPE_AIRKISS](#), [SC_TYPE_ESPTOUCH_AIRKISS](#) }

Functions

- const char * [smartconfig_get_version](#) (void)
Get the version of SmartConfig.
- bool [smartconfig_start](#) ([sc_callback_t](#) cb,...)
Start SmartConfig mode.
- bool [smartconfig_stop](#) (void)
Stop SmartConfig, free the buffer taken by smartconfig_start.
- bool [esptouch_set_timeout](#) (uint8 time_s)
Set timeout of SmartConfig.
- bool [smartconfig_set_type](#) ([sc_type](#) type)
Set protocol type of SmartConfig.

4.22.1 Detailed Description

SmartConfig APIs.

SmartConfig can only be enabled in station only mode. Please make sure the target AP is enabled before enable SmartConfig.

4.22.2 Typedef Documentation

4.22.2.1 typedef void(* [sc_callback_t](#)) ([sc_status](#) status, void *pdata)

The callback of SmartConfig, executed when smart-config status changed.

Parameters

<i>sc_status</i>	<p>status : status of SmartConfig:</p> <ul style="list-style-type: none"> • if status == SC_STATUS_GETTING_SSID_PSWD, parameter void *pdata is a pointer of sc_type, means SmartConfig type: AirKiss or ESP-TOUCH. • if status == SC_STATUS_LINK, parameter void *pdata is a pointer of struct station↔_config; • if status == SC_STATUS_LINK_OVER, parameter void *pdata is a pointer of mobile phone's IP address, 4 bytes. This is only available in ESPTOUCH, otherwise, it is NULL. • otherwise, parameter void *pdata is NULL.
<i>void</i>	*pdata : data of SmartConfig

Returns

null

4.22.3 Enumeration Type Documentation

4.22.3.1 enum sc_status

Enumerator

- SC_STATUS_WAIT** waiting, do not start connection in this phase
- SC_STATUS_FIND_CHANNEL** find target channel, start connection by APP in this phase
- SC_STATUS_GETTING_SSID_PSWD** getting SSID and password of target AP
- SC_STATUS_LINK** connecting to target AP
- SC_STATUS_LINK_OVER** got IP, connect to AP successfully

4.22.3.2 enum sc_type

Enumerator

- SC_TYPE_ESPTOUCH** protocol: ESPTouch
- SC_TYPE_AIRKISS** protocol: AirKiss
- SC_TYPE_ESPTOUCH_AIRKISS** protocol: ESPTouch and AirKiss

4.22.4 Function Documentation

4.22.4.1 bool esptouch_set_timeout (uint8 time_s)

Set timeout of SmartConfig.

Attention

SmartConfig timeout start at SC_STATUS_FIND_CHANNEL, SmartConfig will restart if timeout.

Parameters

<i>uint8</i>	time_s : range 15s~255s, offset:45s.
--------------	--------------------------------------

Returns

true : succeed
false : fail

4.22.4.2 `const char* smartconfig_get_version (void)`

Get the version of SmartConfig.

Parameters

<i>null</i>	
-------------	--

Returns

SmartConfig version

4.22.4.3 `bool smartconfig_set_type (sc_type type)`

Set protocol type of SmartConfig.

Attention

If users need to set the SmartConfig type, please set it before calling `smartconfig_start`.

Parameters

<i>sc_type</i>	type : AirKiss, ESP-TOUCH or both.
----------------	------------------------------------

Returns

true : succeed
false : fail

4.22.4.4 `bool smartconfig_start (sc_callback_t cb, ...)`

Start SmartConfig mode.

Start SmartConfig mode, to connect ESP8266 station to AP, by sniffing for special packets from the air, containing SSID and password of desired AP. You need to broadcast the SSID and password (e.g. from mobile device or computer) with the SSID and password encoded.

Attention

1. This api can only be called in station mode.
2. During SmartConfig, ESP8266 station and soft-AP are disabled.
3. Can not call `smartconfig_start` twice before it finish, please call `smartconfig_stop` first.
4. Don't call any other APIs during SmartConfig, please call `smartconfig_stop` first.

Parameters

<i>sc_callback_t</i>	cb : SmartConfig callback; executed when SmartConfig status changed;
<i>uint8</i>	log : 1, UART output logs; otherwise, UART only outputs the result.

Returns

true : succeed
false : fail

4.22.4.5 bool smartconfig_stop (void)

Stop SmartConfig, free the buffer taken by smartconfig_start.

Attention

Whether connect to AP succeed or not, this API should be called to free memory taken by smartconfig_start.

Parameters

<i>null</i>

Returns

true : succeed
false : fail

4.23 SPI Driver APIs

SPI Flash APIs.

Data Structures

- struct [SpiFlashChip](#)

Macros

- #define [SPI_FLASH_SEC_SIZE](#) 4096

Typedefs

- typedef [SpiFlashOpResult](#)(* [user_spi_flash_read](#)) ([SpiFlashChip](#) *spi, uint32 src_addr, uint32 *des_addr, uint32 size)

Registered function for spi_flash_set_read_func.

Enumerations

- enum [SpiFlashOpResult](#) { [SPI_FLASH_RESULT_OK](#), [SPI_FLASH_RESULT_ERR](#), [SPI_FLASH_RESULT_TIMEOUT](#) }

Functions

- uint32 [spi_flash_get_id](#) (void)
Get ID info of SPI Flash.
- [SpiFlashOpResult](#) [spi_flash_read_status](#) (uint32 *status)
Read state register of SPI Flash.
- [SpiFlashOpResult](#) [spi_flash_write_status](#) (uint32 status_value)
Write state register of SPI Flash.
- [SpiFlashOpResult](#) [spi_flash_erase_sector](#) (uint16 sec)
Erase the Flash sector.
- [SpiFlashOpResult](#) [spi_flash_write](#) (uint32 des_addr, uint32 *src_addr, uint32 size)
Write data to Flash.
- [SpiFlashOpResult](#) [spi_flash_read](#) (uint32 src_addr, uint32 *des_addr, uint32 size)
Read data from Flash.
- void [spi_flash_set_read_func](#) ([user_spi_flash_read](#) read)
Register user-define SPI flash read API.

4.23.1 Detailed Description

SPI Flash APIs.

4.23.2 Macro Definition Documentation

4.23.2.1 #define SPI_FLASH_SEC_SIZE 4096

SPI Flash sector size

4.23.3 Typedef Documentation

4.23.3.1 typedef SpiFlashOpResult(* user_spi_flash_read) (SpiFlashChip *spi, uint32 src_addr, uint32 *des_addr, uint32 size)

Registered function for spi_flash_set_read_func.

Attention

used for sdk internal, don't need to care about params

Parameters

<i>SpiFlashChip</i>	*spi : spi flash struct pointer.
<i>uint32</i>	src_addr : source address of the data.
<i>uint32</i>	*des_addr : destination address in Flash.
<i>uint32</i>	size : length of data

Returns

SpiFlashOpResult

4.23.4 Enumeration Type Documentation

4.23.4.1 enum SpiFlashOpResult

Enumerator

- SPI_FLASH_RESULT_OK*** SPI Flash operating OK
- SPI_FLASH_RESULT_ERR*** SPI Flash operating fail
- SPI_FLASH_RESULT_TIMEOUT*** SPI Flash operating time out

4.23.5 Function Documentation

4.23.5.1 SpiFlashOpResult spi_flash_erase_sector (uint16 sec)

Erase the Flash sector.

Parameters

<i>uint16</i>	sec : Sector number, the count starts at sector 0, 4KB per sector.
---------------	--

Returns

SpiFlashOpResult

4.23.5.2 uint32 spi_flash_get_id (void)

Get ID info of SPI Flash.

Parameters

<i>null</i>	
-------------	--

Returns

SPI Flash ID

4.23.5.3 SpiFlashOpResult spi_flash_read (uint32 *src_addr*, uint32 * *des_addr*, uint32 *size*)

Read data from Flash.

Parameters

<i>uint32</i>	<i>src_addr</i> : source address of the data.
<i>uint32</i>	<i>*des_addr</i> : destination address in Flash.
<i>uint32</i>	<i>size</i> : length of data

Returns

SpiFlashOpResult

4.23.5.4 SpiFlashOpResult spi_flash_read_status (uint32 * status)

Read state register of SPI Flash.

Parameters

<i>uint32</i>	<i>*status</i> : the read value (pointer) of state register.
---------------	--

Returns

SpiFlashOpResult

4.23.5.5 void spi_flash_set_read_func (user_spi_flash_read read)

Register user-define SPI flash read API.

Attention

This API can be only used in SPI overlap mode, please refer to ESP8266_RTOS_SDK .c

Parameters

<i>user_spi_flash_read</i>	<i>read</i> : user-define SPI flash read API .
----------------------------	--

Returns

none

4.23.5.6 SpiFlashOpResult spi_flash_write (uint32 des_addr, uint32 * src_addr, uint32 size)

Write data to Flash.

Parameters

<i>uint32</i>	<i>des_addr</i> : destination address in Flash.
<i>uint32</i>	<i>*src_addr</i> : source address of the data.
<i>uint32</i>	<i>size</i> : length of data

Returns

SpiFlashOpResult

4.23.5.7 SpiFlashOpResult spi_flash_write_status (uint32 status_value)

Write state register of SPI Flash.

Parameters

<i>uint32</i>	status_value : Write state register value.
---------------	--

Returns

SpiFlashOpResult

4.24 Upgrade APIs

Firmware upgrade (FOTA) APIs.

Data Structures

- struct [upgrade_server_info](#)

Macros

- #define [SPI_FLASH_SEC_SIZE](#) 4096
- #define [USER_BIN1](#) 0x00
- #define [USER_BIN2](#) 0x01
- #define [UPGRADE_FLAG_IDLE](#) 0x00
- #define [UPGRADE_FLAG_START](#) 0x01
- #define [UPGRADE_FLAG_FINISH](#) 0x02
- #define [UPGRADE_FW_BIN1](#) 0x00
- #define [UPGRADE_FW_BIN2](#) 0x01

Typedefs

- typedef void(* [upgrade_states_check_callback](#)) (void *arg)
Callback of upgrading firmware through WiFi.

Functions

- uint8 [system_upgrade_userbin_check](#) (void)
Check the user bin.
- void [system_upgrade_reboot](#) (void)
Reboot system to use the new software.
- uint8 [system_upgrade_flag_check](#) ()
Check the upgrade status flag.
- void [system_upgrade_flag_set](#) (uint8 flag)
Set the upgrade status flag.
- void [system_upgrade_init](#) ()
Upgrade function initialization.
- void [system_upgrade_deinit](#) ()
Upgrade function de-initialization.
- bool [system_upgrade](#) (uint8 *data, uint32 len)
Upgrade function de-initialization.
- bool [system_upgrade_start](#) (struct [upgrade_server_info](#) *server)
Start upgrade firmware through WiFi with normal connection.

4.24.1 Detailed Description

Firmware upgrade (FOTA) APIs.

4.24.2 Macro Definition Documentation

4.24.2.1 #define SPI_FLASH_SEC_SIZE 4096

SPI Flash sector size

4.24.2.2 #define UPGRADE_FLAG_FINISH 0x02

flag of upgrading firmware, finish upgrading

4.24.2.3 #define UPGRADE_FLAG_IDLE 0x00

flag of upgrading firmware, idle

4.24.2.4 #define UPGRADE_FLAG_START 0x01

flag of upgrading firmware, start upgrade

4.24.2.5 #define UPGRADE_FW_BIN1 0x00

firmware, user1.bin

4.24.2.6 #define UPGRADE_FW_BIN2 0x01

firmware, user2.bin

4.24.2.7 #define USER_BIN1 0x00

firmware, user1.bin

4.24.2.8 #define USER_BIN2 0x01

firmware, user2.bin

4.24.3 Typedef Documentation

4.24.3.1 typedef void(* upgrade_states_check_callback) (void *arg)

Callback of upgrading firmware through WiFi.

Parameters

<i>void</i>	* arg : information about upgrading server
-------------	--

Returns

null

4.24.4 Function Documentation

4.24.4.1 `bool system_upgrade (uint8 * data, uint32 len)`

Upgrade function de-initialization.

Parameters

<i>uint8</i>	*data : segment of the firmware bin data
<i>uint32</i>	len : length of the segment bin data

Returns

null

4.24.4.2 void system_upgrade_deinit ()

Upgrade function de-initialization.

Parameters

<i>null</i>

Returns

null

4.24.4.3 uint8 system_upgrade_flag_check ()

Check the upgrade status flag.

Parameters

<i>null</i>

Returns

```
#define UPGRADE_FLAG_IDLE 0x00
#define UPGRADE_FLAG_START 0x01
#define UPGRADE_FLAG_FINISH 0x02
```

4.24.4.4 void system_upgrade_flag_set (uint8 flag)

Set the upgrade status flag.

Attention

After downloading new softwares, set the flag to UPGRADE_FLAG_FINISH and call system_upgrade_reboot to reboot the system in order to run the new software.

Parameters

<i>uint8</i>	flag: <ul style="list-style-type: none"> • UPGRADE_FLAG_IDLE 0x00 • UPGRADE_FLAG_START 0x01 • UPGRADE_FLAG_FINISH 0x02
--------------	---

Returns

null

4.24.4.5 void system_upgrade_init ()

Upgrade function initialization.

Parameters

<i>null</i>

Returns

null

4.24.4.6 void system_upgrade_reboot (void)

Reboot system to use the new software.

Parameters

<i>null</i>

Returns

null

4.24.4.7 bool system_upgrade_start (struct upgrade_server_info * server)

Start upgrade firmware through WiFi with normal connection.

Parameters

<i>struct</i>	upgrade_server_info *server : the firmware upgrade server info
---------------	--

Returns

true : succeed
false : fail

4.24.4.8 uint8 system_upgrade_userbin_check (void)

Check the user bin.

Parameters

<i>null</i>

Returns

0x00 : UPGRADE_FW_BIN1, i.e. user1.bin
0x01 : UPGRADE_FW_BIN2, i.e. user2.bin

Chapter 5

Data Structure Documentation

5.1 `_esp_event` Struct Reference

Data Fields

- [SYSTEM_EVENT event_id](#)
- [Event_Info_u event_info](#)

5.1.1 Field Documentation

5.1.1.1 `SYSTEM_EVENT event_id`

even ID

5.1.1.2 `Event_Info_u event_info`

event information

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

- `include/espressif/esp_wifi.h`

5.2 `_esp_tcp` Struct Reference

Data Fields

- `int remote_port`
- `int local_port`
- `uint8 local_ip [4]`
- `uint8 remote_ip [4]`
- `espconn_connect_callback connect_callback`
- `espconn_reconnect_callback reconnect_callback`
- `espconn_connect_callback disconnect_callback`
- `espconn_connect_callback write_finish_fn`

5.2.1 Field Documentation

5.2.1.1 `espconn_connect_callback` `connect_callback`

connected callback

5.2.1.2 `espconn_connect_callback` `disconnect_callback`

disconnected callback

5.2.1.3 `uint8` `local_ip[4]`

local IP of ESP8266

5.2.1.4 `int` `local_port`

ESP8266's local port of TCP connection

5.2.1.5 `espconn_reconnect_callback` `reconnect_callback`

as error handler, the TCP connection broke unexpectedly

5.2.1.6 `uint8` `remote_ip[4]`

remote IP of TCP connection

5.2.1.7 `int` `remote_port`

remote port of TCP connection

5.2.1.8 `espconn_connect_callback` `write_finish_fn`

data send by `espconn_send` has wrote into buffer waiting for sending, or has sent successfully

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

- `include/espressif/espconn.h`

5.3 `_esp_udp` Struct Reference

Data Fields

- `int` `remote_port`
- `int` `local_port`
- `uint8` `local_ip` [4]
- `uint8` `remote_ip` [4]

5.3.1 Field Documentation

5.3.1.1 `uint8 local_ip[4]`

local IP of ESP8266

5.3.1.2 `int local_port`

ESP8266's local port for UDP transmission

5.3.1.3 `uint8 remote_ip[4]`

remote IP of UDP transmission

5.3.1.4 `int remote_port`

remote port of UDP transmission

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

- `include/espressif/espconn.h`

5.4 `_os_timer_t` Struct Reference

Data Fields

- `struct _os_timer_t * timer_next`
- `void * timer_handle`
- `uint32 timer_expire`
- `uint32 timer_period`
- `os_timer_func_t * timer_func`
- `bool timer_repeat_flag`
- `void * timer_arg`

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

- `include/espressif/esp_timer.h`

5.5 `_remot_info` Struct Reference

Data Fields

- `enum espconn_state state`
- `int remote_port`
- `uint8 remote_ip [4]`

5.5.1 Field Documentation

5.5.1.1 `uint8 remote_ip[4]`

remote IP address

5.5.1.2 int remote_port

remote port

5.5.1.3 enum espconn_state state

state of espconn

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

- include/espressif/espconn.h

5.6 airkiss_config_t Struct Reference

Data Fields

- airkiss_memset_fn **memset**
- airkiss_memcpy_fn **memcpy**
- airkiss_memcmp_fn **memcmp**
- airkiss_printf_fn **printf**

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

- include/espressif/airkiss.h

5.7 bss_info Struct Reference

Public Member Functions

- [STAILQ_ENTRY](#) ([bss_info](#)) next

Data Fields

- uint8 [bssid](#) [6]
- uint8 [ssid](#) [32]
- uint8 [ssid_len](#)
- uint8 [channel](#)
- sint8 [rssi](#)
- [AUTH_MODE](#) [authmode](#)
- uint8 [is_hidden](#)
- sint16 [freq_offset](#)
- sint16 [freqcal_val](#)
- uint8 * [esp_mesh_ie](#)
- [CIPHER_TYPE](#) [pairwise_cipher](#)
- [CIPHER_TYPE](#) [group_cipher](#)
- uint32_t [phy_11b](#):1
- uint32_t [phy_11g](#):1
- uint32_t [phy_11n](#):1
- uint32_t [wps](#):1
- uint32_t [reserved](#):28

5.7.1 Member Function Documentation

5.7.1.1 STAILQ_ENTRY (bss_info)

information of next AP

5.7.2 Field Documentation

5.7.2.1 AUTH_MODE authmode

authmode of AP

5.7.2.2 uint8 bssid[6]

MAC address of AP

5.7.2.3 uint8 channel

channel of AP

5.7.2.4 sint16 freq_offset

frequency offset

5.7.2.5 CIPHER_TYPE group_cipher

group cipher of AP

5.7.2.6 uint8 is_hidden

SSID of current AP is hidden or not.

5.7.2.7 CIPHER_TYPE pairwise_cipher

pairwise cipher of AP

5.7.2.8 uint32_t phy_11b

bit: 0 flag to identify if 11b mode is enabled or not

5.7.2.9 uint32_t phy_11g

bit: 1 flag to identify if 11g mode is enabled or not

5.7.2.10 uint32_t phy_11n

bit: 2 flag to identify if 11n mode is enabled or not

5.7.2.11 uint32_t reserved

bit: 4..31 reserved

5.7.2.12 sint8 rssi

single strength of AP

5.7.2.13 uint8 ssid[32]

SSID of AP

5.7.2.14 uint8 ssid_len

SSID length

5.7.2.15 uint32_t wps

bit: 3 flag to identify if WPS is supported or not

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

- include/espressif/esp_sta.h

5.8 cmd_s Struct Reference

Data Fields

- char * **cmd_str**
- uint8 **flag**
- uint8 **id**
- void(* **cmd_func**)(void)
- void(* **cmd_callback**)(void *arg)

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

- include/espressif/esp_ssc.h

5.9 dhcps_lease Struct Reference

Data Fields

- bool [enable](#)
- struct ip_addr [start_ip](#)
- struct ip_addr [end_ip](#)

5.9.1 Field Documentation

5.9.1.1 bool enable

enable DHCP lease or not

5.9.1.2 struct ip_addr end_ip

end IP of IP range

5.9.1.3 struct ip_addr start_ip

start IP of IP range

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

- include/espressif/esp_misc.h

5.10 esp_spiffs_config Struct Reference

Data Fields

- uint32 [phys_size](#)
- uint32 [phys_addr](#)
- uint32 [phys_erase_block](#)
- uint32 [log_block_size](#)
- uint32 [log_page_size](#)
- uint32 [fd_buf_size](#)
- uint32 [cache_buf_size](#)

5.10.1 Field Documentation

5.10.1.1 uint32 cache_buf_size

cache buffer size

5.10.1.2 uint32 fd_buf_size

file descriptor memory area size

5.10.1.3 uint32 log_block_size

logical size of a block, must be on physical block size boundary and must never be less than a physical block

5.10.1.4 uint32 log_page_size

logical size of a page, at least `log_block_size/8`

5.10.1.5 uint32 phys_addr

physical offset in spi flash used for spiffs, must be on block boundary

5.10.1.6 uint32 phys_erase_block

physical size when erasing a block

5.10.1.7 uint32 phys_size

physical size of the SPI Flash

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

- include/espressif/esp_spiffs.h

5.11 espconn Struct Reference

```
#include <espconn.h>
```

Data Fields

- enum [espconn_type](#) type
- enum [espconn_state](#) state
- union {
 - [esp_tcp](#) * tcp
 - [esp_udp](#) * udp
- } proto
- [espconn_recv_callback](#) recv_callback
- [espconn_sent_callback](#) sent_callback
- uint8 [link_cnt](#)
- void * [reserve](#)

5.11.1 Detailed Description

A espconn descriptor

5.11.2 Field Documentation

5.11.2.1 uint8 link_cnt

link count

5.11.2.2 espconn_recv_callback recv_callback

data received callback

5.11.2.3 void* reserve

reserved for user data

5.11.2.4 espconn_sent_callback sent_callback

data sent callback

5.11.2.5 enum espconn_state state

current state of the espconn

5.11.2.6 enum espconn_type type

type of the espconn (TCP or UDP)

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

- include/espressif/espconn.h

5.12 Event_Info_u Union Reference

Data Fields

- [Event_StaMode_ScanDone_t scan_done](#)
- [Event_StaMode_Connected_t connected](#)
- [Event_StaMode_Disconnected_t disconnected](#)
- [Event_StaMode_AuthMode_Change_t auth_change](#)
- [Event_StaMode_Got_IP_t got_ip](#)
- [Event_SoftAPMode_StaConnected_t sta_connected](#)
- [Event_SoftAPMode_StaDisconnected_t sta_disconnected](#)
- [Event_SoftAPMode_ProbeReqRecved_t ap_probereqrecved](#)

5.12.1 Field Documentation

5.12.1.1 Event_SoftAPMode_ProbeReqRecved_t ap_probereqrecved

ESP8266 softAP receive probe request packet

5.12.1.2 Event_StaMode_AuthMode_Change_t auth_change

the auth mode of AP ESP8266 station connected to changed

5.12.1.3 Event_StaMode_Connected_t connected

ESP8266 station connected to AP

5.12.1.4 Event_StaMode_Disconnected_t disconnected

ESP8266 station disconnected to AP

5.12.1.5 Event_StaMode_Got_IP_t got_ip

ESP8266 station got IP

5.12.1.6 Event_StaMode_ScanDone_t scan_done

ESP8266 station scan (APs) done

5.12.1.7 Event_SoftAPMode_StaConnected_t sta_connected

a station connected to ESP8266 soft-AP

5.12.1.8 Event_SoftAPMode_StaDisconnected_t sta_disconnected

a station disconnected to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.13 Event_SoftAPMode_ProbeReqRecved_t Struct Reference

Data Fields

- int [rssi](#)
- uint8 [mac](#) [6]

5.13.1 Field Documentation

5.13.1.1 uint8 mac[6]

MAC address of the station which send probe request

5.13.1.2 int rssi

Received probe request signal strength

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

- include/espressif/esp_wifi.h

5.14 Event_SoftAPMode_StaConnected_t Struct Reference

Data Fields

- uint8 [mac](#) [6]
- uint8 [aid](#)

5.14.1 Field Documentation

5.14.1.1 uint8 aid

the aid that ESP8266 soft-AP gives to the station connected to

5.14.1.2 uint8 mac[6]

MAC address of the station connected to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.15 Event_SoftAPMode_StaDisconnected_t Struct Reference

Data Fields

- uint8 [mac](#) [6]
- uint8 [aid](#)

5.15.1 Field Documentation

5.15.1.1 uint8 aid

the aid that ESP8266 soft-AP gave to the station disconnects to

5.15.1.2 uint8 mac[6]

MAC address of the station disconnects to ESP8266 soft-AP

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

- include/espressif/esp_wifi.h

5.16 Event_StaMode_AuthMode_Change_t Struct Reference

Data Fields

- uint8 [old_mode](#)
- uint8 [new_mode](#)

5.16.1 Field Documentation

5.16.1.1 uint8 new_mode

the new auth mode of AP

5.16.1.2 uint8 old_mode

the old auth mode of AP

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

- include/espressif/esp_wifi.h

5.17 Event_StaMode_Connected_t Struct Reference

Data Fields

- uint8 [ssid](#) [32]
- uint8 [ssid_len](#)
- uint8 [bssid](#) [6]
- uint8 [channel](#)

5.17.1 Field Documentation

5.17.1.1 uint8 bssid[6]

BSSID of connected AP

5.17.1.2 uint8 channel

channel of connected AP

5.17.1.3 uint8 ssid[32]

SSID of connected AP

5.17.1.4 uint8 ssid_len

SSID length of connected AP

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

- include/espressif/esp_wifi.h

5.18 Event_StaMode_Disconnected_t Struct Reference

Data Fields

- uint8 [ssid](#) [32]
- uint8 [ssid_len](#)
- uint8 [bssid](#) [6]
- uint8 [reason](#)

5.18.1 Field Documentation

5.18.1.1 uint8 bssid[6]

BSSID of disconnected AP

5.18.1.2 uint8 reason

reason of disconnection

5.18.1.3 uint8 ssid[32]

SSID of disconnected AP

5.18.1.4 uint8 ssid_len

SSID length of disconnected AP

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

- include/espressif/esp_wifi.h

5.19 Event_StaMode_Got_IP_t Struct Reference

Data Fields

- struct ip_addr [ip](#)
- struct ip_addr [mask](#)
- struct ip_addr [gw](#)

5.19.1 Field Documentation

5.19.1.1 struct ip_addr gw

gateway that ESP8266 station got from connected AP

5.19.1.2 struct ip_addr ip

IP address that ESP8266 station got from connected AP

5.19.1.3 struct ip_addr mask

netmask that ESP8266 station got from connected AP

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

- include/espressif/esp_wifi.h

5.20 Event_StaMode_ScanDone_t Struct Reference

Data Fields

- uint32 [status](#)
- struct [bss_info](#) * [bss](#)

5.20.1 Field Documentation

5.20.1.1 struct bss_info* bss

list of APs found

5.20.1.2 uint32 status

status of scanning APs

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

- include/espressif/esp_wifi.h

5.21 ip_info Struct Reference

Data Fields

- struct ip_addr [ip](#)

- struct ip_addr [netmask](#)
- struct ip_addr [gw](#)

5.21.1 Field Documentation

5.21.1.1 struct ip_addr gw

gateway

5.21.1.2 struct ip_addr ip

IP address

5.21.1.3 struct ip_addr netmask

netmask

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

- include/espressif/esp_wifi.h

5.22 pwm_param Struct Reference

Data Fields

- uint32 [period](#)
- uint32 [freq](#)
- uint32 [duty](#) [8]

5.22.1 Field Documentation

5.22.1.1 uint32 duty[8]

PWM duty

5.22.1.2 uint32 freq

PWM frequency

5.22.1.3 uint32 period

PWM period

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

- include/espressif/pwm.h

5.23 rst_info Struct Reference

Data Fields

- [rst_reason](#) reason
- uint32 **exccause**
- uint32 **epc1**
- uint32 **epc2**
- uint32 **epc3**
- uint32 **excvaddr**
- uint32 **depc**
- uint32 **rtn_addr**

5.23.1 Field Documentation

5.23.1.1 [rst_reason](#) reason

enum [rst_reason](#)

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

- include/espressif/esp_system.h

5.24 scan_config Struct Reference

Data Fields

- uint8 * [ssid](#)
- uint8 * [bssid](#)
- uint8 [channel](#)
- uint8 [show_hidden](#)
- [wifi_scan_type_t](#) [scan_type](#)
- [wifi_scan_time_t](#) [scan_time](#)

5.24.1 Field Documentation

5.24.1.1 [uint8*](#) [bssid](#)

MAC address of AP

5.24.1.2 [uint8](#) [channel](#)

channel, scan the specific channel

5.24.1.3 [wifi_scan_time_t](#) [scan_time](#)

scan time per channel

5.24.1.4 [wifi_scan_type_t](#) [scan_type](#)

scan type, active or passive

5.24.1.5 uint8 show_hidden

enable to scan AP whose SSID is hidden

5.24.1.6 uint8* ssid

SSID of AP

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

- include/espressif/esp_sta.h

5.25 softap_config Struct Reference

Data Fields

- uint8 [ssid](#) [32]
- uint8 [password](#) [64]
- uint8 [ssid_len](#)
- uint8 [channel](#)
- [AUTH_MODE](#) [authmode](#)
- uint8 [ssid_hidden](#)
- uint8 [max_connection](#)
- uint16 [beacon_interval](#)

5.25.1 Field Documentation

5.25.1.1 AUTH_MODE authmode

Auth mode of ESP8266 soft-AP. Do not support AUTH_WEP in soft-AP mode

5.25.1.2 uint16 beacon_interval

Beacon interval, 100 ~ 60000 ms, default 100

5.25.1.3 uint8 channel

Channel of ESP8266 soft-AP

5.25.1.4 uint8 max_connection

Max number of stations allowed to connect in, default 4, max 4

5.25.1.5 uint8 password[64]

Password of ESP8266 soft-AP

5.25.1.6 uint8 ssid[32]

SSID of ESP8266 soft-AP

5.25.1.7 uint8 ssid_hidden

Broadcast SSID or not, default 0, broadcast the SSID

5.25.1.8 uint8 ssid_len

Length of SSID. If `softap_config.ssid_len==0`, check the SSID until there is a termination character; otherwise, set the SSID length according to `softap_config.ssid_len`.

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

- `include/espressif/esp_softap.h`

5.26 SpiFlashChip Struct Reference

Data Fields

- uint32 **deviceld**
- uint32 **chip_size**
- uint32 **block_size**
- uint32 **sector_size**
- uint32 **page_size**
- uint32 **status_mask**

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

- `include/espressif/spi_flash.h`

5.27 station_config Struct Reference

Data Fields

- uint8 `ssid` [32]
- uint8 `password` [64]
- uint8 `bssid_set`
- uint8 `bssid` [6]

5.27.1 Field Documentation

5.27.1.1 uint8 bssid[6]

MAC address of target AP

5.27.1.2 uint8 bssid_set

whether set MAC address of target AP or not. Generally, `station_config.bssid_set` needs to be 0; and it needs to be 1 only when users need to check the MAC address of the AP.

5.27.1.3 uint8 password[64]

password of target AP

5.27.1.4 uint8 ssid[32]

SSID of target AP

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

- include/espressif/esp_sta.h

5.28 station_info Struct Reference

Public Member Functions

- [STAILQ_ENTRY](#) ([station_info](#)) next

Data Fields

- uint8 [bssid](#) [6]
- struct ip_addr [ip](#)

5.28.1 Member Function Documentation

5.28.1.1 STAILQ_ENTRY (station_info)

Information of next AP

5.28.2 Field Documentation

5.28.2.1 uint8 bssid[6]

BSSID of AP

5.28.2.2 struct ip_addr ip

IP address of AP

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

- include/espressif/esp_softap.h

5.29 upgrade_server_info Struct Reference

Data Fields

- struct sockaddr_in [sockaddrin](#)
- [upgrade_states_check_callback](#) [check_cb](#)
- uint32 [check_times](#)
- uint8 [pre_version](#) [16]
- uint8 [upgrade_version](#) [16]
- uint8 * [url](#)
- void * [pclient_param](#)
- uint8 [upgrade_flag](#)

5.29.1 Field Documentation

5.29.1.1 upgrade_states_check_callback check_cb

callback of upgrading

5.29.1.2 uint32 check_times

time out of upgrading, unit : ms

5.29.1.3 uint8 pre_version[16]

previous version of firmware

5.29.1.4 struct sockaddr_in sockaddrin

socket of upgrading

5.29.1.5 uint8 upgrade_flag

true, upgrade succeed; false, upgrade fail

5.29.1.6 uint8 upgrade_version[16]

the new version of firmware

5.29.1.7 uint8* url

the url of upgrading server

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

- include/espressif/upgrade.h

5.30 wifi_active_scan_time_t Struct Reference

Range of active scan times per channel.

```
#include <esp_sta.h>
```

Data Fields

- [uint32_t min](#)
- [uint32_t max](#)

5.30.1 Detailed Description

Range of active scan times per channel.

5.30.2 Field Documentation

5.30.2.1 uint32_t max

maximum active scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

5.30.2.2 uint32_t min

minimum active scan time per channel, units: millisecond

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

- include/espressif/esp_sta.h

5.31 wifi_country_t Struct Reference

Data Fields

- char [cc](#) [3]
- uint8_t [schan](#)
- uint8_t [nchan](#)
- uint8_t [policy](#)

5.31.1 Field Documentation

5.31.1.1 char cc[3]

country code string

5.31.1.2 uint8_t nchan

total channel number

5.31.1.3 uint8_t policy

country policy

5.31.1.4 uint8_t schan

start channel

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

- include/espressif/esp_wifi.h

5.32 wifi_scan_time_t Union Reference

Aggregate of active & passive scan time per channel.

```
#include <esp_sta.h>
```

Data Fields

- [wifi_active_scan_time_t active](#)
- [uint32_t passive](#)

5.32.1 Detailed Description

Aggregate of active & passive scan time per channel.

5.32.2 Field Documentation

5.32.2.1 wifi_active_scan_time_t active

active scan time per channel, units: millisecond.

5.32.2.2 uint32_t passive

passive scan time per channel, units: millisecond, values above 1500ms may cause station to disconnect from AP and are not recommended.

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

- `include/espressif/esp_sta.h`

Index

- [_esp_event, 129](#)
 - [event_id, 129](#)
 - [event_info, 129](#)
 - [_esp_tcp, 129](#)
 - [connect_callback, 130](#)
 - [disconnect_callback, 130](#)
 - [local_ip, 130](#)
 - [local_port, 130](#)
 - [reconnect_callback, 130](#)
 - [remote_ip, 130](#)
 - [remote_port, 130](#)
 - [write_finish_fn, 130](#)
 - [_esp_udp, 130](#)
 - [local_ip, 131](#)
 - [local_port, 131](#)
 - [remote_ip, 131](#)
 - [remote_port, 131](#)
 - [_os_timer_t, 131](#)
 - [_remot_info, 131](#)
 - [remote_ip, 131](#)
 - [remote_port, 131](#)
 - [state, 132](#)
 - [AIRKISS_LAN_CONTINUE](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_ERR_CMD](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_ERR_OVERFLOW](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_ERR_PAKE](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_ERR_PARA](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_ERR_PKG](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_PAKE_READY](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_SSDP_REQ](#)
 - [AirKiss APIs, 8](#)
 - [AUTH_MODE](#)
 - [Common APIs, 52](#)
 - [AUTH_OPEN](#)
 - [Common APIs, 52](#)
 - [AUTH_WEP](#)
 - [Common APIs, 52](#)
 - [AUTH_WPA2_PSK](#)
 - [Common APIs, 52](#)
 - [AUTH_WPA_PSK](#)
 - [Common APIs, 52](#)
 - [AUTH_WPA_WPA2_PSK](#)
 - [Common APIs, 52](#)
- [Common APIs, 52](#)
 - [active](#)
 - [wifi_scan_time_t, 149](#)
 - [aid](#)
 - [Event_SoftAPMode_StaConnected_t, 138](#)
 - [Event_SoftAPMode_StaDisconnected_t, 139](#)
 - [AirKiss APIs, 8](#)
 - [AIRKISS_LAN_CONTINUE, 8](#)
 - [AIRKISS_LAN_ERR_CMD, 8](#)
 - [AIRKISS_LAN_ERR_OVERFLOW, 8](#)
 - [AIRKISS_LAN_ERR_PAKE, 8](#)
 - [AIRKISS_LAN_ERR_PARA, 8](#)
 - [AIRKISS_LAN_ERR_PKG, 8](#)
 - [AIRKISS_LAN_PAKE_READY, 8](#)
 - [AIRKISS_LAN_SSDP_REQ, 8](#)
 - [airkiss_lan_pack, 9](#)
 - [airkiss_lan_recv, 10](#)
 - [airkiss_lan_ret_t, 8](#)
 - [airkiss_version, 10](#)
 - [airkiss_config_t, 132](#)
 - [airkiss_lan_pack](#)
 - [AirKiss APIs, 9](#)
 - [airkiss_lan_recv](#)
 - [AirKiss APIs, 10](#)
 - [airkiss_lan_ret_t](#)
 - [AirKiss APIs, 8](#)
 - [airkiss_version](#)
 - [AirKiss APIs, 10](#)
 - [ap_probereqrecvd](#)
 - [Event_Info_u, 137](#)
 - [auth_change](#)
 - [Event_Info_u, 137](#)
 - [authmode](#)
 - [bss_info, 133](#)
 - [softap_config, 144](#)
 - [beacon_interval](#)
 - [softap_config, 144](#)
 - [Boot APIs, 43](#)
 - [FLASH_SIZE_128M_MAP_1024_1024, 44](#)
 - [FLASH_SIZE_16M_MAP_1024_1024, 44](#)
 - [FLASH_SIZE_16M_MAP_512_512, 44](#)
 - [FLASH_SIZE_2M, 44](#)
 - [FLASH_SIZE_32M_MAP_1024_1024, 44](#)
 - [FLASH_SIZE_32M_MAP_2048_2048, 44](#)
 - [FLASH_SIZE_32M_MAP_512_512, 44](#)
 - [FLASH_SIZE_4M_MAP_256_256, 44](#)
 - [FLASH_SIZE_64M_MAP_1024_1024, 44](#)
 - [FLASH_SIZE_8M_MAP_512_512, 44](#)
 - [flash_size_map, 44](#)

- SYS_BOOT_ENHANCE_MODE, 43
- SYS_BOOT_NORMAL_BIN, 43
- SYS_BOOT_NORMAL_MODE, 44
- SYS_BOOT_TEST_BIN, 44
- system_get_boot_mode, 44
- system_get_boot_version, 44
- system_get_cpu_freq, 45
- system_get_flash_size_map, 45
- system_get_userbin_addr, 45
- system_restart_enhance, 45
- system_update_cpu_freq, 46
- bss
 - Event_StaMode_ScanDone_t, 141
- bss_info, 132
 - authmode, 133
 - bssid, 133
 - channel, 133
 - freq_offset, 133
 - group_cipher, 133
 - is_hidden, 133
 - pairwise_cipher, 133
 - phy_11b, 133
 - phy_11g, 133
 - phy_11n, 133
 - reserved, 133
 - rsi, 134
 - STAILQ_ENTRY, 133
 - ssid, 134
 - ssid_len, 134
 - wps, 134
- bssid
 - bss_info, 133
 - Event_StaMode_Connected_t, 140
 - Event_StaMode_Disconnected_t, 140
 - scan_config, 143
 - station_config, 145
 - station_info, 146
- bssid_set
 - station_config, 145
- CIPHER_CCMP
 - Station APIs, 24
- CIPHER_NONE
 - Station APIs, 24
- CIPHER_TKIP
 - Station APIs, 24
- CIPHER_TKIP_CCMP
 - Station APIs, 24
- CIPHER_TYPE
 - Station APIs, 24
- CIPHER_UNKNOWN
 - Station APIs, 24
- CIPHER_WEP104
 - Station APIs, 24
- CIPHER_WEP40
 - Station APIs, 24
- cache_buf_size
 - esp_spiffs_config, 135
- cc
 - wifi_country_t, 148
- channel
 - bss_info, 133
 - Event_StaMode_Connected_t, 140
 - scan_config, 143
 - softap_config, 144
- check_cb
 - upgrade_server_info, 147
- check_times
 - upgrade_server_info, 147
- cmd_s, 134
- Common APIs, 49
 - AUTH_MODE, 52
 - AUTH_OPEN, 52
 - AUTH_WEP, 52
 - AUTH_WPA2_PSK, 52
 - AUTH_WPA_PSK, 52
 - AUTH_WPA_WPA2_PSK, 52
 - EVENT_SOFTAPMODE_PROBEREQRECVED, 52
 - EVENT_SOFTAPMODE_STACONNECTED, 52
 - EVENT_SOFTAPMODE_STADISCONNECTED, 52
 - EVENT_STAMODE_AUTHMODE_CHANGE, 52
 - EVENT_STAMODE_CONNECTED, 52
 - EVENT_STAMODE_DHCP_TIMEOUT, 52
 - EVENT_STAMODE_DISCONNECTED, 52
 - EVENT_STAMODE_GOT_IP, 52
 - EVENT_STAMODE_SCAN_DONE, 52
 - freedom_outside_cb_t, 51
 - NULL_MODE, 53
 - PHY_MODE_11B, 53
 - PHY_MODE_11G, 53
 - PHY_MODE_11N, 53
 - rfid_locp_cb_t, 51
 - SOFTAP_IF, 52
 - SOFTAP_MODE, 53
 - STATION_IF, 52
 - STATION_MODE, 53
 - STATIONAP_MODE, 53
 - SYSTEM_EVENT, 52
 - WIFI_COUNTRY_POLICY, 52
 - WIFI_COUNTRY_POLICY_AUTO, 52
 - WIFI_COUNTRY_POLICY_MANUAL, 52
 - WIFI_INTERFACE, 52
 - WIFI_MODE, 52
 - WIFI_PHY_MODE, 53
 - wifi_event_handler_cb_t, 51
 - wifi_get_ip_info, 53
 - wifi_get_macaddr, 53
 - wifi_get_opmode, 53
 - wifi_get_opmode_default, 54
 - wifi_get_phy_mode, 54
 - wifi_get_sleep_type, 54
 - wifi_register_rfid_locp_rcv_cb, 54
 - wifi_register_send_pkt_freedom_cb, 55
 - wifi_rfid_locp_rcv_close, 55
 - wifi_rfid_locp_rcv_open, 55

- wifi_send_pkt_freedom, 55
- wifi_set_event_handler_cb, 56
- wifi_set_ip_info, 56
- wifi_set_macaddr, 57
- wifi_set_opmode, 57
- wifi_set_opmode_current, 57
- wifi_set_phy_mode, 58
- wifi_set_sleep_type, 58
- wifi_status_led_install, 58
- wifi_status_led_uninstall, 60
- wifi_unregister_rfid_locp_rcv_cb, 60
- wifi_unregister_send_pkt_freedom_cb, 60
- connect_callback
 - _esp_tcp, 130
- connected
 - Event_Info_u, 137
- DHCP_STARTED
 - Misc APIs, 11
- DHCP_STOPPED
 - Misc APIs, 11
- dhcp_status
 - Misc APIs, 11
- dhcps_lease, 134
 - enable, 134
 - end_ip, 134
 - start_ip, 135
- dhcps_offer_option
 - Misc APIs, 11
- disconnect_callback
 - _esp_tcp, 130
- disconnected
 - Event_Info_u, 137
- dns_found_callback
 - Network Espconn APIs, 84
- Driver APIs, 109
- duty
 - pwm_param, 142
- ESP-NOW APIs, 99
 - esp_now_add_peer, 102
 - esp_now_deinit, 102
 - esp_now_del_peer, 102
 - esp_now_fetch_peer, 102
 - esp_now_get_cnt_info, 103
 - esp_now_get_peer_channel, 103
 - esp_now_get_peer_key, 103
 - esp_now_get_peer_role, 104
 - esp_now_get_self_role, 104
 - esp_now_init, 104
 - esp_now_is_peer_exist, 104
 - esp_now_rcv_cb_t, 100
 - esp_now_register_rcv_cb, 105
 - esp_now_register_send_cb, 105
 - esp_now_send, 105
 - esp_now_send_cb_t, 100
 - esp_now_set_kok, 105
 - esp_now_set_peer_channel, 107
 - esp_now_set_peer_key, 107
 - esp_now_set_peer_role, 107
 - esp_now_set_self_role, 108
 - esp_now_unregister_rcv_cb, 108
 - esp_now_unregister_send_cb, 108
- ESPCONN_ABRT
 - Network Espconn APIs, 83
- ESPCONN_ARG
 - Network Espconn APIs, 83
- ESPCONN_CLOSE
 - Network Espconn APIs, 86
- ESPCONN_CLSD
 - Network Espconn APIs, 83
- ESPCONN_CONN
 - Network Espconn APIs, 83
- ESPCONN_CONNECT
 - Network Espconn APIs, 86
- ESPCONN_COPY
 - Network Espconn APIs, 86
- ESPCONN_END
 - Network Espconn APIs, 86
- ESPCONN_IF
 - Network Espconn APIs, 83
- ESPCONN_INPROGRESS
 - Network Espconn APIs, 83
- ESPCONN_INVALID
 - Network Espconn APIs, 86
- ESPCONN_ISCONN
 - Network Espconn APIs, 83
- ESPCONN_KEEPALIVE
 - Network Espconn APIs, 86
- ESPCONN_KEEPCNT
 - Network Espconn APIs, 85
- ESPCONN_KEEPIDLE
 - Network Espconn APIs, 85
- ESPCONN_KEEPINTVL
 - Network Espconn APIs, 85
- ESPCONN_LISTEN
 - Network Espconn APIs, 86
- ESPCONN_MAXNUM
 - Network Espconn APIs, 83
- ESPCONN_MEM
 - Network Espconn APIs, 84
- ESPCONN_NODELAY
 - Network Espconn APIs, 86
- ESPCONN_NONE
 - Network Espconn APIs, 86
- ESPCONN_OK
 - Network Espconn APIs, 84
- ESPCONN_READ
 - Network Espconn APIs, 86
- ESPCONN_REUSEADDR
 - Network Espconn APIs, 86
- ESPCONN_RST
 - Network Espconn APIs, 84
- ESPCONN_RTE
 - Network Espconn APIs, 84
- ESPCONN_START
 - Network Espconn APIs, 86

- ESPCONN_TCP
 - Network Espconn APIs, [86](#)
- ESPCONN_TIMEOUT
 - Network Espconn APIs, [84](#)
- ESPCONN_UDP
 - Network Espconn APIs, [86](#)
- ESPCONN_WAIT
 - Network Espconn APIs, [86](#)
- ESPCONN_WRITE
 - Network Espconn APIs, [86](#)
- EVENT_SOFTAPMODE_PROBEREQRECVD
 - Common APIs, [52](#)
- EVENT_SOFTAPMODE_STACONNECTED
 - Common APIs, [52](#)
- EVENT_SOFTAPMODE_STADISCONNECTED
 - Common APIs, [52](#)
- EVENT_STAMODE_AUTHMODE_CHANGE
 - Common APIs, [52](#)
- EVENT_STAMODE_CONNECTED
 - Common APIs, [52](#)
- EVENT_STAMODE_DHCP_TIMEOUT
 - Common APIs, [52](#)
- EVENT_STAMODE_DISCONNECTED
 - Common APIs, [52](#)
- EVENT_STAMODE_GOT_IP
 - Common APIs, [52](#)
- EVENT_STAMODE_SCAN_DONE
 - Common APIs, [52](#)
- enable
 - dhcps_lease, [134](#)
- end_ip
 - dhcps_lease, [134](#)
- esp_now_add_peer
 - ESP-NOW APIs, [102](#)
- esp_now_deinit
 - ESP-NOW APIs, [102](#)
- esp_now_del_peer
 - ESP-NOW APIs, [102](#)
- esp_now_fetch_peer
 - ESP-NOW APIs, [102](#)
- esp_now_get_cnt_info
 - ESP-NOW APIs, [103](#)
- esp_now_get_peer_channel
 - ESP-NOW APIs, [103](#)
- esp_now_get_peer_key
 - ESP-NOW APIs, [103](#)
- esp_now_get_peer_role
 - ESP-NOW APIs, [104](#)
- esp_now_get_self_role
 - ESP-NOW APIs, [104](#)
- esp_now_init
 - ESP-NOW APIs, [104](#)
- esp_now_is_peer_exist
 - ESP-NOW APIs, [104](#)
- esp_now_rcv_cb_t
 - ESP-NOW APIs, [100](#)
- esp_now_register_rcv_cb
 - ESP-NOW APIs, [105](#)
- esp_now_register_send_cb
 - ESP-NOW APIs, [105](#)
- esp_now_send
 - ESP-NOW APIs, [105](#)
- esp_now_send_cb_t
 - ESP-NOW APIs, [100](#)
- esp_now_set_kok
 - ESP-NOW APIs, [105](#)
- esp_now_set_peer_channel
 - ESP-NOW APIs, [107](#)
- esp_now_set_peer_key
 - ESP-NOW APIs, [107](#)
- esp_now_set_peer_role
 - ESP-NOW APIs, [107](#)
- esp_now_set_self_role
 - ESP-NOW APIs, [108](#)
- esp_now_unregister_rcv_cb
 - ESP-NOW APIs, [108](#)
- esp_now_unregister_send_cb
 - ESP-NOW APIs, [108](#)
- esp_spiffs_config, [135](#)
 - cache_buf_size, [135](#)
 - fd_buf_size, [135](#)
 - log_block_size, [135](#)
 - log_page_size, [135](#)
 - phys_addr, [135](#)
 - phys_erase_block, [135](#)
 - phys_size, [135](#)
- esp_spiffs_deinit
 - Spiffs APIs, [19](#)
- esp_spiffs_init
 - Spiffs APIs, [19](#)
- espconn, [136](#)
 - link_cnt, [136](#)
 - rcv_callback, [136](#)
 - reserve, [136](#)
 - sent_callback, [136](#)
 - state, [136](#)
 - type, [136](#)
- espconn_accept
 - Network Espconn APIs, [86](#)
- espconn_clear_opt
 - Network Espconn APIs, [86](#)
- espconn_connect
 - Network Espconn APIs, [87](#)
- espconn_connect_callback
 - Network Espconn APIs, [84](#)
- espconn_create
 - Network Espconn APIs, [87](#)
- espconn_delete
 - Network Espconn APIs, [87](#)
- espconn_disconnect
 - Network Espconn APIs, [88](#)
- espconn_dns_setserver
 - Network Espconn APIs, [88](#)
- espconn_get_connection_info
 - Network Espconn APIs, [89](#)
- espconn_get_keepalive

- Network Espconn APIs, [89](#)
- espconn_gethostbyname
 - Network Espconn APIs, [89](#)
- espconn_igmp_join
 - Network Espconn APIs, [90](#)
- espconn_igmp_leave
 - Network Espconn APIs, [90](#)
- espconn_init
 - Network Espconn APIs, [90](#)
- espconn_level
 - Network Espconn APIs, [85](#)
- espconn_option
 - Network Espconn APIs, [85](#)
- espconn_port
 - Network Espconn APIs, [91](#)
- espconn_reconnect_callback
 - Network Espconn APIs, [85](#)
- espconn_rcv_callback
 - Network Espconn APIs, [85](#)
- espconn_rcv_hold
 - Network Espconn APIs, [91](#)
- espconn_rcv_unhold
 - Network Espconn APIs, [91](#)
- espconn_regist_connectcb
 - Network Espconn APIs, [92](#)
- espconn_regist_disconcb
 - Network Espconn APIs, [92](#)
- espconn_regist_reconcb
 - Network Espconn APIs, [92](#)
- espconn_regist_rcvcb
 - Network Espconn APIs, [93](#)
- espconn_regist_sentcb
 - Network Espconn APIs, [93](#)
- espconn_regist_time
 - Network Espconn APIs, [93](#)
- espconn_regist_write_finish
 - Network Espconn APIs, [94](#)
- espconn_send
 - Network Espconn APIs, [94](#)
- espconn_sendto
 - Network Espconn APIs, [95](#)
- espconn_sent
 - Network Espconn APIs, [95](#)
- espconn_set_keepalive
 - Network Espconn APIs, [96](#)
- espconn_set_opt
 - Network Espconn APIs, [96](#)
- espconn_state
 - Network Espconn APIs, [86](#)
- espconn_tcp_get_max_con
 - Network Espconn APIs, [97](#)
- espconn_tcp_get_max_con_allow
 - Network Espconn APIs, [97](#)
- espconn_tcp_set_max_con
 - Network Espconn APIs, [97](#)
- espconn_tcp_set_max_con_allow
 - Network Espconn APIs, [98](#)
- espconn_type
 - Network Espconn APIs, [86](#)
- esptouch_set_timeout
 - Smartconfig APIs, [114](#)
- Event_Info_u, [137](#)
 - ap_probereqrecved, [137](#)
 - auth_change, [137](#)
 - connected, [137](#)
 - disconnected, [137](#)
 - got_ip, [137](#)
 - scan_done, [137](#)
 - sta_connected, [137](#)
 - sta_disconnected, [137](#)
- Event_SoftAPMode_ProbeReqRecved_t, [138](#)
 - mac, [138](#)
 - rsi, [138](#)
- Event_SoftAPMode_StaConnected_t, [138](#)
 - aid, [138](#)
 - mac, [138](#)
- Event_SoftAPMode_StaDisconnected_t, [139](#)
 - aid, [139](#)
 - mac, [139](#)
- Event_StaMode_AuthMode_Change_t, [139](#)
 - new_mode, [139](#)
 - old_mode, [139](#)
- Event_StaMode_Connected_t, [139](#)
 - bssid, [140](#)
 - channel, [140](#)
 - ssid, [140](#)
 - ssid_len, [140](#)
- Event_StaMode_Disconnected_t, [140](#)
 - bssid, [140](#)
 - reason, [140](#)
 - ssid, [140](#)
 - ssid_len, [140](#)
- Event_StaMode_Got_IP_t, [141](#)
 - gw, [141](#)
 - ip, [141](#)
 - mask, [141](#)
- Event_StaMode_ScanDone_t, [141](#)
 - bss, [141](#)
 - status, [141](#)
- event_id
 - _esp_event, [129](#)
- event_info
 - _esp_event, [129](#)
- FLASH_SIZE_128M_MAP_1024_1024
 - Boot APIs, [44](#)
- FLASH_SIZE_16M_MAP_1024_1024
 - Boot APIs, [44](#)
- FLASH_SIZE_16M_MAP_512_512
 - Boot APIs, [44](#)
- FLASH_SIZE_2M
 - Boot APIs, [44](#)
- FLASH_SIZE_32M_MAP_1024_1024
 - Boot APIs, [44](#)
- FLASH_SIZE_32M_MAP_2048_2048
 - Boot APIs, [44](#)
- FLASH_SIZE_32M_MAP_512_512

- Boot APIs, 44
- FLASH_SIZE_4M_MAP_256_256
 - Boot APIs, 44
- FLASH_SIZE_64M_MAP_1024_1024
 - Boot APIs, 44
- FLASH_SIZE_8M_MAP_512_512
 - Boot APIs, 44
- fd_buf_size
 - esp_spiffs_config, 135
- flash_size_map
 - Boot APIs, 44
- Force Sleep APIs, 61
 - wifi_fpm_close, 61
 - wifi_fpm_do_sleep, 61
 - wifi_fpm_do_wakeup, 62
 - wifi_fpm_get_sleep_type, 62
 - wifi_fpm_open, 62
 - wifi_fpm_set_sleep_type, 63
 - wifi_fpm_set_wakeup_cb, 63
- freedom_outside_cb_t
 - Common APIs, 51
- freq
 - pwm_param, 142
- freq_offset
 - bss_info, 133
- got_ip
 - Event_Info_u, 137
- group_cipher
 - bss_info, 133
- gw
 - Event_StaMode_Got_IP_t, 141
 - ip_info, 142
- IP2STR
 - Misc APIs, 11
- ip
 - Event_StaMode_Got_IP_t, 141
 - ip_info, 142
 - station_info, 146
- ip_info, 141
 - gw, 142
 - ip, 142
 - netmask, 142
- is_hidden
 - bss_info, 133
- link_cnt
 - espconn, 136
- local_ip
 - _esp_tcp, 130
 - _esp_udp, 131
- local_port
 - _esp_tcp, 130
 - _esp_udp, 131
- log_block_size
 - esp_spiffs_config, 135
- log_page_size
 - esp_spiffs_config, 135
- mac
 - Event_SoftAPMode_ProbeReqRecved_t, 138
 - Event_SoftAPMode_StaConnected_t, 138
 - Event_SoftAPMode_StaDisconnected_t, 139
- mask
 - Event_StaMode_Got_IP_t, 141
- max
 - wifi_active_scan_time_t, 148
- max_connection
 - softap_config, 144
- min
 - wifi_active_scan_time_t, 148
- Misc APIs, 11
 - DHCP_STARTED, 11
 - DHCP_STOPPED, 11
 - dhcp_status, 11
 - dhcps_offer_option, 11
 - IP2STR, 11
 - OFFER_END, 12
 - OFFER_ROUTER, 12
 - OFFER_START, 12
 - os_delay_us, 12
 - os_install_putc1, 12
 - os_putc, 12
- NULL_MODE
 - Common APIs, 53
- nchan
 - wifi_country_t, 148
- netmask
 - ip_info, 142
- Network Espconn APIs, 81
 - dns_found_callback, 84
 - ESPCONN_ABRT, 83
 - ESPCONN_ARG, 83
 - ESPCONN_CLOSE, 86
 - ESPCONN_CLSD, 83
 - ESPCONN_CONN, 83
 - ESPCONN_CONNECT, 86
 - ESPCONN_COPY, 86
 - ESPCONN_END, 86
 - ESPCONN_IF, 83
 - ESPCONN_INPROGRESS, 83
 - ESPCONN_INVALID, 86
 - ESPCONN_ISCONN, 83
 - ESPCONN_KEEPAKIVE, 86
 - ESPCONN_KEEPCNT, 85
 - ESPCONN_KEEPIBLE, 85
 - ESPCONN_KEEPINTVL, 85
 - ESPCONN_LISTEN, 86
 - ESPCONN_MAXNUM, 83
 - ESPCONN_MEM, 84
 - ESPCONN_NODELAY, 86
 - ESPCONN_NONE, 86
 - ESPCONN_OK, 84
 - ESPCONN_READ, 86
 - ESPCONN_REUSEADDR, 86
 - ESPCONN_RST, 84
 - ESPCONN_RTE, 84

- ESPCONN_START, 86
- ESPCONN_TCP, 86
- ESPCONN_TIMEOUT, 84
- ESPCONN_UDP, 86
- ESPCONN_WAIT, 86
- ESPCONN_WRITE, 86
- espconn_accept, 86
- espconn_clear_opt, 86
- espconn_connect, 87
- espconn_connect_callback, 84
- espconn_create, 87
- espconn_delete, 87
- espconn_disconnect, 88
- espconn_dns_setserver, 88
- espconn_get_connection_info, 89
- espconn_get_keepalive, 89
- espconn_gethostbyname, 89
- espconn_igmp_join, 90
- espconn_igmp_leave, 90
- espconn_init, 90
- espconn_level, 85
- espconn_option, 85
- espconn_port, 91
- espconn_reconnect_callback, 85
- espconn_rcv_callback, 85
- espconn_rcv_hold, 91
- espconn_rcv_unhold, 91
- espconn_regist_connectcb, 92
- espconn_regist_disconcb, 92
- espconn_regist_reconcb, 92
- espconn_regist_rcvcb, 93
- espconn_regist_sentcb, 93
- espconn_regist_time, 93
- espconn_regist_write_finish, 94
- espconn_send, 94
- espconn_sendto, 95
- espconn_sent, 95
- espconn_set_keepalive, 96
- espconn_set_opt, 96
- espconn_state, 86
- espconn_tcp_get_max_con, 97
- espconn_tcp_get_max_con_allow, 97
- espconn_tcp_set_max_con, 97
- espconn_tcp_set_max_con_allow, 98
- espconn_type, 86
- new_mode
 - Event_StaMode_AuthMode_Change_t, 139
- OFFER_END
 - Misc APIs, 12
- OFFER_ROUTER
 - Misc APIs, 12
- OFFER_START
 - Misc APIs, 12
- old_mode
 - Event_StaMode_AuthMode_Change_t, 139
- os_delay_us
 - Misc APIs, 12
- os_install_putc1
 - Misc APIs, 12
- os_putc
 - Misc APIs, 12
- os_timer_arm
 - Software timer APIs, 47
- os_timer_disarm
 - Software timer APIs, 47
- os_timer_setfn
 - Software timer APIs, 47
- PHY_MODE_11B
 - Common APIs, 53
- PHY_MODE_11G
 - Common APIs, 53
- PHY_MODE_11N
 - Common APIs, 53
- PWM Driver APIs, 110
 - pwm_get_duty, 110
 - pwm_get_period, 110
 - pwm_init, 111
 - pwm_set_duty, 111
 - pwm_set_period, 111
 - pwm_start, 112
- pairwise_cipher
 - bss_info, 133
- passive
 - wifi_scan_time_t, 149
- password
 - softap_config, 144
 - station_config, 145
- period
 - pwm_param, 142
- phy_11b
 - bss_info, 133
- phy_11g
 - bss_info, 133
- phy_11n
 - bss_info, 133
- phys_addr
 - esp_spiffs_config, 135
- phys_erase_block
 - esp_spiffs_config, 135
- phys_size
 - esp_spiffs_config, 135
- policy
 - wifi_country_t, 148
- pre_version
 - upgrade_server_info, 147
- pwm_get_duty
 - PWM Driver APIs, 110
- pwm_get_period
 - PWM Driver APIs, 110
- pwm_init
 - PWM Driver APIs, 111
- pwm_param, 142
 - duty, 142
 - freq, 142
 - period, 142
- pwm_set_duty

- PWM Driver APIs, 111
- pwm_set_period
 - PWM Driver APIs, 111
- pwm_start
 - PWM Driver APIs, 112
- REASON_DEEP_SLEEP_AWAKE
 - System APIs, 33
- REASON_DEFAULT_RST
 - System APIs, 33
- REASON_EXCEPTION_RST
 - System APIs, 33
- REASON_EXT_SYS_RST
 - System APIs, 33
- REASON_SOFT_RESTART
 - System APIs, 33
- REASON_SOFT_WDT_RST
 - System APIs, 33
- REASON_WDT_RST
 - System APIs, 33
- Rate Control APIs, 64
 - wifi_get_user_fixed_rate, 65
 - wifi_get_user_limit_rate_mask, 65
 - wifi_set_user_fixed_rate, 65
 - wifi_set_user_limit_rate_mask, 66
 - wifi_set_user_rate_limit, 66
 - wifi_set_user_sup_rate, 67
- reason
 - Event_StaMode_Disconnected_t, 140
 - rst_info, 143
- reconnect_callback
 - _esp_tcp, 130
- recv_callback
 - espconn, 136
- remote_ip
 - _esp_tcp, 130
 - _esp_udp, 131
 - _remot_info, 131
- remote_port
 - _esp_tcp, 130
 - _esp_udp, 131
 - _remot_info, 131
- reserve
 - espconn, 136
- reserved
 - bss_info, 133
- rfid_locp_cb_t
 - Common APIs, 51
- rssi
 - bss_info, 134
 - Event_SoftAPMode_ProbeReqRecved_t, 138
- rst_info, 143
 - reason, 143
- rst_reason
 - System APIs, 33
- SC_STATUS_FIND_CHANNEL
 - Smartconfig APIs, 114
- SC_STATUS_GETTING_SSID_PSWD
 - Smartconfig APIs, 114
- SC_STATUS_LINK
 - Smartconfig APIs, 114
- SC_STATUS_LINK_OVER
 - Smartconfig APIs, 114
- SC_STATUS_WAIT
 - Smartconfig APIs, 114
- SC_TYPE_AIRKISS
 - Smartconfig APIs, 114
- SC_TYPE_ESPTOUCH
 - Smartconfig APIs, 114
- SC_TYPE_ESPTOUCH_AIRKISS
 - Smartconfig APIs, 114
- SOFTAP_IF
 - Common APIs, 52
- SOFTAP_MODE
 - Common APIs, 53
- SPI Driver APIs, 117
 - SPI_FLASH_RESULT_ERR, 118
 - SPI_FLASH_RESULT_OK, 118
 - SPI_FLASH_RESULT_TIMEOUT, 118
 - SPI_FLASH_SEC_SIZE, 117
 - spi_flash_erase_sector, 118
 - spi_flash_get_id, 118
 - spi_flash_read, 118
 - spi_flash_read_status, 120
 - spi_flash_set_read_func, 120
 - spi_flash_write, 120
 - spi_flash_write_status, 120
 - SpiFlashOpResult, 118
 - user_spi_flash_read, 118
- SPI_FLASH_RESULT_ERR
 - SPI Driver APIs, 118
- SPI_FLASH_RESULT_OK
 - SPI Driver APIs, 118
- SPI_FLASH_RESULT_TIMEOUT
 - SPI Driver APIs, 118
- SPI_FLASH_SEC_SIZE
 - SPI Driver APIs, 117
 - Upgrade APIs, 123
- SSC APIs, 20
 - ssc_attach, 20
 - ssc_param_len, 20
 - ssc_param_str, 20
 - ssc_parse_param, 21
 - ssc_register, 21
- STAILQ_ENTRY
 - bss_info, 133
 - station_info, 146
- STATION_CONNECT_FAIL
 - Station APIs, 24
- STATION_CONNECTING
 - Station APIs, 24
- STATION_GOT_IP
 - Station APIs, 24
- STATION_IDLE
 - Station APIs, 24
- STATION_IF

- Common APIs, 52
- STATION_MODE
 - Common APIs, 53
- STATION_NO_AP_FOUND
 - Station APIs, 24
- STATION_STATUS
 - Station APIs, 24
- STATION_WRONG_PASSWORD
 - Station APIs, 24
- STATIONAP_MODE
 - Common APIs, 53
- SYS_BOOT_ENHANCE_MODE
 - Boot APIs, 43
- SYS_BOOT_NORMAL_BIN
 - Boot APIs, 43
- SYS_BOOT_NORMAL_MODE
 - Boot APIs, 44
- SYS_BOOT_TEST_BIN
 - Boot APIs, 44
- SYSTEM_EVENT
 - Common APIs, 52
- sc_callback_t
 - Smartconfig APIs, 113
- sc_status
 - Smartconfig APIs, 114
- sc_type
 - Smartconfig APIs, 114
- scan_config, 143
 - bssid, 143
 - channel, 143
 - scan_time, 143
 - scan_type, 143
 - show_hidden, 143
 - ssid, 144
- scan_done
 - Event_Info_u, 137
- scan_done_cb_t
 - Station APIs, 23
- scan_time
 - scan_config, 143
- scan_type
 - scan_config, 143
- schan
 - wifi_country_t, 148
- sent_callback
 - espcn, 136
- show_hidden
 - scan_config, 143
- Smartconfig APIs, 113
 - esptouch_set_timeout, 114
 - SC_STATUS_FIND_CHANNEL, 114
 - SC_STATUS_GETTING_SSID_PSWD, 114
 - SC_STATUS_LINK, 114
 - SC_STATUS_LINK_OVER, 114
 - SC_STATUS_WAIT, 114
 - SC_TYPE_AIRKISS, 114
 - SC_TYPE_ESPTOUCH, 114
 - SC_TYPE_ESPTOUCH_AIRKISS, 114
- sc_callback_t, 113
- sc_status, 114
- sc_type, 114
- smartconfig_get_version, 115
- smartconfig_set_type, 115
- smartconfig_start, 115
- smartconfig_stop, 116
- smartconfig_get_version
 - Smartconfig APIs, 115
- smartconfig_set_type
 - Smartconfig APIs, 115
- smartconfig_start
 - Smartconfig APIs, 115
- smartconfig_stop
 - Smartconfig APIs, 116
- Sniffer APIs, 74
 - wifi_get_channel, 74
 - wifi_get_country, 75
 - wifi_promiscuous_cb_t, 74
 - wifi_promiscuous_enable, 75
 - wifi_promiscuous_set_mac, 75
 - wifi_set_channel, 76
 - wifi_set_country, 76
 - wifi_set_promiscuous_rx_cb, 76
- sockaddrin
 - upgrade_server_info, 147
- SoftAP APIs, 13
 - wifi_softap_dhcps_start, 14
 - wifi_softap_dhcps_status, 14
 - wifi_softap_dhcps_stop, 14
 - wifi_softap_free_station_info, 14
 - wifi_softap_get_config, 15
 - wifi_softap_get_config_default, 15
 - wifi_softap_get_dhcps_lease, 15
 - wifi_softap_get_dhcps_lease_time, 15
 - wifi_softap_get_station_info, 16
 - wifi_softap_get_station_num, 16
 - wifi_softap_reset_dhcps_lease_time, 16
 - wifi_softap_set_config, 17
 - wifi_softap_set_config_current, 17
 - wifi_softap_set_dhcps_lease, 17
 - wifi_softap_set_dhcps_lease_time, 18
 - wifi_softap_set_dhcps_offer_option, 18
- softap_config, 144
 - authmode, 144
 - beacon_interval, 144
 - channel, 144
 - max_connection, 144
 - password, 144
 - ssid, 144
 - ssid_hidden, 144
 - ssid_len, 145
- Software timer APIs, 47
 - os_timer_arm, 47
 - os_timer_disarm, 47
 - os_timer_setfn, 47
- spl_flash_erase_sector
 - SPI Driver APIs, 118

- spi_flash_get_id
 - SPI Driver APIs, 118
- spi_flash_read
 - SPI Driver APIs, 118
- spi_flash_read_status
 - SPI Driver APIs, 120
- spi_flash_set_read_func
 - SPI Driver APIs, 120
- spi_flash_write
 - SPI Driver APIs, 120
- spi_flash_write_status
 - SPI Driver APIs, 120
- SpiFlashChip, 145
- SpiFlashOpResult
 - SPI Driver APIs, 118
- Spiffs APIs, 19
 - esp_spiffs_deinit, 19
 - esp_spiffs_init, 19
- ssc_attach
 - SSC APIs, 20
- ssc_param_len
 - SSC APIs, 20
- ssc_param_str
 - SSC APIs, 20
- ssc_parse_param
 - SSC APIs, 21
- ssc_register
 - SSC APIs, 21
- ssid
 - bss_info, 134
 - Event_StaMode_Connected_t, 140
 - Event_StaMode_Disconnected_t, 140
 - scan_config, 144
 - softap_config, 144
 - station_config, 145
- ssid_hidden
 - softap_config, 144
- ssid_len
 - bss_info, 134
 - Event_StaMode_Connected_t, 140
 - Event_StaMode_Disconnected_t, 140
 - softap_config, 145
- sta_connected
 - Event_Info_u, 137
- sta_disconnected
 - Event_Info_u, 137
- start_ip
 - dhcps_lease, 135
- state
 - _remot_info, 132
 - espconn, 136
- Station APIs, 22
 - CIPHER_CCMP, 24
 - CIPHER_NONE, 24
 - CIPHER_TKIP, 24
 - CIPHER_TKIP_CCMP, 24
 - CIPHER_TYPE, 24
 - CIPHER_UNKNOWN, 24
 - CIPHER_WEP104, 24
 - CIPHER_WEP40, 24
 - STATION_CONNECT_FAIL, 24
 - STATION_CONNECTING, 24
 - STATION_GOT_IP, 24
 - STATION_IDLE, 24
 - STATION_NO_AP_FOUND, 24
 - STATION_STATUS, 24
 - STATION_WRONG_PASSWORD, 24
 - scan_done_cb_t, 23
 - WIFI_SCAN_TYPE_ACTIVE, 24
 - WIFI_SCAN_TYPE_PASSIVE, 24
 - wifi_scan_type_t, 24
 - wifi_station_ap_change, 24
 - wifi_station_ap_number_set, 24
 - wifi_station_connect, 25
 - wifi_station_dhcp_start, 25
 - wifi_station_dhcp_status, 25
 - wifi_station_dhcp_stop, 26
 - wifi_station_disconnect, 26
 - wifi_station_get_ap_info, 26
 - wifi_station_get_auto_connect, 27
 - wifi_station_get_config, 27
 - wifi_station_get_config_default, 27
 - wifi_station_get_connect_status, 27
 - wifi_station_get_current_ap_id, 28
 - wifi_station_get_hostname, 28
 - wifi_station_get_reconnect_policy, 28
 - wifi_station_get_rssi, 28
 - wifi_station_scan, 28
 - wifi_station_set_auto_connect, 29
 - wifi_station_set_config, 29
 - wifi_station_set_config_current, 30
 - wifi_station_set_hostname, 30
 - wifi_station_set_reconnect_policy, 30
- station_config, 145
 - bssid, 145
 - bssid_set, 145
 - password, 145
 - ssid, 145
- station_info, 146
 - bssid, 146
 - ip, 146
 - STAILQ_ENTRY, 146
- status
 - Event_StaMode_ScanDone_t, 141
- System APIs, 32
 - REASON_DEEP_SLEEP_AWAKE, 33
 - REASON_DEFAULT_RST, 33
 - REASON_EXCEPTION_RST, 33
 - REASON_EXT_SYS_RST, 33
 - REASON_SOFT_RESTART, 33
 - REASON_SOFT_WDT_RST, 33
 - REASON_WDT_RST, 33
 - rst_reason, 33
 - system_adc_read, 33
 - system_deep_sleep, 34
 - system_deep_sleep_set_option, 34

- system_get_chip_id, 34
- system_get_free_heap_size, 35
- system_get_rst_info, 35
- system_get_rtc_time, 35
- system_get_sdk_version, 35
- system_get_time, 37
- system_get_vdd33, 37
- system_param_load, 37
- system_param_save_with_protect, 38
- system_phy_set_max_tpw, 38
- system_phy_set_rfoption, 38
- system_phy_set_tpw_via_vdd33, 39
- system_print_meminfo, 39
- system_restart, 39
- system_restore, 40
- system_rtc_clock_cali_proc, 40
- system_rtc_mem_read, 40
- system_rtc_mem_write, 41
- system_uart_de_swap, 41
- system_uart_swap, 41
- system_adc_read
 - System APIs, 33
- system_deep_sleep
 - System APIs, 34
- system_deep_sleep_set_option
 - System APIs, 34
- system_get_boot_mode
 - Boot APIs, 44
- system_get_boot_version
 - Boot APIs, 44
- system_get_chip_id
 - System APIs, 34
- system_get_cpu_freq
 - Boot APIs, 45
- system_get_flash_size_map
 - Boot APIs, 45
- system_get_free_heap_size
 - System APIs, 35
- system_get_rst_info
 - System APIs, 35
- system_get_rtc_time
 - System APIs, 35
- system_get_sdk_version
 - System APIs, 35
- system_get_time
 - System APIs, 37
- system_get_userbin_addr
 - Boot APIs, 45
- system_get_vdd33
 - System APIs, 37
- system_param_load
 - System APIs, 37
- system_param_save_with_protect
 - System APIs, 38
- system_phy_set_max_tpw
 - System APIs, 38
- system_phy_set_rfoption
 - System APIs, 38
- system_phy_set_tpw_via_vdd33
 - System APIs, 39
- system_print_meminfo
 - System APIs, 39
- system_restart
 - System APIs, 39
- system_restart_enhance
 - Boot APIs, 45
- system_restore
 - System APIs, 40
- system_rtc_clock_cali_proc
 - System APIs, 40
- system_rtc_mem_read
 - System APIs, 40
- system_rtc_mem_write
 - System APIs, 41
- system_uart_de_swap
 - System APIs, 41
- system_uart_swap
 - System APIs, 41
- system_update_cpu_freq
 - Boot APIs, 46
- system_upgrade
 - Upgrade APIs, 124
- system_upgrade_deinit
 - Upgrade APIs, 125
- system_upgrade_flag_check
 - Upgrade APIs, 125
- system_upgrade_flag_set
 - Upgrade APIs, 125
- system_upgrade_init
 - Upgrade APIs, 125
- system_upgrade_reboot
 - Upgrade APIs, 127
- system_upgrade_start
 - Upgrade APIs, 127
- system_upgrade_userbin_check
 - Upgrade APIs, 127
- type
 - espconn, 136
- UPGRADE_FLAG_FINISH
 - Upgrade APIs, 123
- UPGRADE_FLAG_IDLE
 - Upgrade APIs, 123
- UPGRADE_FLAG_START
 - Upgrade APIs, 123
- UPGRADE_FW_BIN1
 - Upgrade APIs, 123
- UPGRADE_FW_BIN2
 - Upgrade APIs, 123
- USER_BIN1
 - Upgrade APIs, 123
- USER_BIN2
 - Upgrade APIs, 123
- Upgrade APIs, 122
 - SPI_FLASH_SEC_SIZE, 123
 - system_upgrade, 124

- system_upgrade_deinit, 125
- system_upgrade_flag_check, 125
- system_upgrade_flag_set, 125
- system_upgrade_init, 125
- system_upgrade_reboot, 127
- system_upgrade_start, 127
- system_upgrade_userbin_check, 127
- UPGRADE_FLAG_FINISH, 123
- UPGRADE_FLAG_IDLE, 123
- UPGRADE_FLAG_START, 123
- UPGRADE_FW_BIN1, 123
- UPGRADE_FW_BIN2, 123
- USER_BIN1, 123
- USER_BIN2, 123
- upgrade_states_check_callback, 123
- upgrade_flag
 - upgrade_server_info, 147
- upgrade_server_info, 146
 - check_cb, 147
 - check_times, 147
 - pre_version, 147
 - sockaddrin, 147
 - upgrade_flag, 147
 - upgrade_version, 147
 - url, 147
- upgrade_states_check_callback
 - Upgrade APIs, 123
- upgrade_version
 - upgrade_server_info, 147
- url
 - upgrade_server_info, 147
- User IE APIs, 71
 - user_ie_manufacturer_rcv_cb_t, 71
 - wifi_register_user_ie_manufacturer_rcv_cb, 72
 - wifi_set_user_ie, 73
 - wifi_unregister_user_ie_manufacturer_rcv_cb, 73
- user_ie_manufacturer_rcv_cb_t
 - User IE APIs, 71
- user_spi_flash_read
 - SPI Driver APIs, 118
- VND_IE_TYPE_ASSOC_REQ
 - Vendor IE APIs, 69
- VND_IE_TYPE_ASSOC_RESP
 - Vendor IE APIs, 69
- VND_IE_TYPE_BEACON
 - Vendor IE APIs, 68
- VND_IE_TYPE_PROBE_REQ
 - Vendor IE APIs, 68
- VND_IE_TYPE_PROBE_RESP
 - Vendor IE APIs, 69
- Vendor IE APIs, 68
 - VND_IE_TYPE_ASSOC_REQ, 69
 - VND_IE_TYPE_ASSOC_RESP, 69
 - VND_IE_TYPE_BEACON, 68
 - VND_IE_TYPE_PROBE_REQ, 68
 - VND_IE_TYPE_PROBE_RESP, 69
 - vendor_ie_rcv_cb_t, 68
 - vendor_ie_type, 68
 - wifi_register_vnd_ie_rcv_cb, 69
 - wifi_set_vnd_ie, 69
 - wifi_unregister_vnd_ie_rcv_cb, 69
- vendor_ie_rcv_cb_t
 - Vendor IE APIs, 68
- vendor_ie_type
 - Vendor IE APIs, 68
- WIFI_COUNTRY_POLICY
 - Common APIs, 52
- WIFI_COUNTRY_POLICY_AUTO
 - Common APIs, 52
- WIFI_COUNTRY_POLICY_MANUAL
 - Common APIs, 52
- WIFI_INTERFACE
 - Common APIs, 52
- WIFI_MODE
 - Common APIs, 52
- WIFI_PHY_MODE
 - Common APIs, 53
- WIFI_SCAN_TYPE_ACTIVE
 - Station APIs, 24
- WIFI_SCAN_TYPE_PASSIVE
 - Station APIs, 24
- WPS APIs, 78
 - WPS_CB_ST_FAILED, 79
 - WPS_CB_ST_SCAN_ERR, 79
 - WPS_CB_ST_SUCCESS, 79
 - WPS_CB_ST_TIMEOUT, 79
 - WPS_CB_ST_WEP, 79
 - wifi_set_wps_cb, 79
 - wifi_wps_disable, 79
 - wifi_wps_enable, 79
 - wifi_wps_start, 80
 - wps_cb_status, 79
 - wps_st_cb_t, 78
- WPS_CB_ST_FAILED
 - WPS APIs, 79
- WPS_CB_ST_SCAN_ERR
 - WPS APIs, 79
- WPS_CB_ST_SUCCESS
 - WPS APIs, 79
- WPS_CB_ST_TIMEOUT
 - WPS APIs, 79
- WPS_CB_ST_WEP
 - WPS APIs, 79
- WiFi Related APIs, 7
 - wifi_active_scan_time_t, 147
 - max, 148
 - min, 148
 - wifi_country_t, 148
 - cc, 148
 - nchan, 148
 - policy, 148
 - schan, 148
 - wifi_event_handler_cb_t
 - Common APIs, 51
 - wifi_fpm_close
 - Force Sleep APIs, 61

- wifi_fpm_do_sleep
 - Force Sleep APIs, 61
- wifi_fpm_do_wakeup
 - Force Sleep APIs, 62
- wifi_fpm_get_sleep_type
 - Force Sleep APIs, 62
- wifi_fpm_open
 - Force Sleep APIs, 62
- wifi_fpm_set_sleep_type
 - Force Sleep APIs, 63
- wifi_fpm_set_wakeup_cb
 - Force Sleep APIs, 63
- wifi_get_channel
 - Sniffer APIs, 74
- wifi_get_country
 - Sniffer APIs, 75
- wifi_get_ip_info
 - Common APIs, 53
- wifi_get_macaddr
 - Common APIs, 53
- wifi_get_opmode
 - Common APIs, 53
- wifi_get_opmode_default
 - Common APIs, 54
- wifi_get_phy_mode
 - Common APIs, 54
- wifi_get_sleep_type
 - Common APIs, 54
- wifi_get_user_fixed_rate
 - Rate Control APIs, 65
- wifi_get_user_limit_rate_mask
 - Rate Control APIs, 65
- wifi_promiscuous_cb_t
 - Sniffer APIs, 74
- wifi_promiscuous_enable
 - Sniffer APIs, 75
- wifi_promiscuous_set_mac
 - Sniffer APIs, 75
- wifi_register_rfid_locp_rcv_cb
 - Common APIs, 54
- wifi_register_send_pkt_freedom_cb
 - Common APIs, 55
- wifi_register_user_ie_manufacturer_rcv_cb
 - User IE APIs, 72
- wifi_register_vnd_ie_rcv_cb
 - Vendor IE APIs, 69
- wifi_rfid_locp_rcv_close
 - Common APIs, 55
- wifi_rfid_locp_rcv_open
 - Common APIs, 55
- wifi_scan_time_t, 148
 - active, 149
 - passive, 149
- wifi_scan_type_t
 - Station APIs, 24
- wifi_send_pkt_freedom
 - Common APIs, 55
- wifi_set_channel
 - Sniffer APIs, 76
- wifi_set_country
 - Sniffer APIs, 76
- wifi_set_event_handler_cb
 - Common APIs, 56
- wifi_set_ip_info
 - Common APIs, 56
- wifi_set_macaddr
 - Common APIs, 57
- wifi_set_opmode
 - Common APIs, 57
- wifi_set_opmode_current
 - Common APIs, 57
- wifi_set_phy_mode
 - Common APIs, 58
- wifi_set_promiscuous_rx_cb
 - Sniffer APIs, 76
- wifi_set_sleep_type
 - Common APIs, 58
- wifi_set_user_fixed_rate
 - Rate Control APIs, 65
- wifi_set_user_ie
 - User IE APIs, 73
- wifi_set_user_limit_rate_mask
 - Rate Control APIs, 66
- wifi_set_user_rate_limit
 - Rate Control APIs, 66
- wifi_set_user_sup_rate
 - Rate Control APIs, 67
- wifi_set_vnd_ie
 - Vendor IE APIs, 69
- wifi_set_wps_cb
 - WPS APIs, 79
- wifi_softap_dhcps_start
 - SoftAP APIs, 14
- wifi_softap_dhcps_status
 - SoftAP APIs, 14
- wifi_softap_dhcps_stop
 - SoftAP APIs, 14
- wifi_softap_free_station_info
 - SoftAP APIs, 14
- wifi_softap_get_config
 - SoftAP APIs, 15
- wifi_softap_get_config_default
 - SoftAP APIs, 15
- wifi_softap_get_dhcps_lease
 - SoftAP APIs, 15
- wifi_softap_get_dhcps_lease_time
 - SoftAP APIs, 15
- wifi_softap_get_station_info
 - SoftAP APIs, 16
- wifi_softap_get_station_num
 - SoftAP APIs, 16
- wifi_softap_reset_dhcps_lease_time
 - SoftAP APIs, 16
- wifi_softap_set_config
 - SoftAP APIs, 17
- wifi_softap_set_config_current

- SoftAP APIs, [17](#)
- wifi_softap_set_dhcp_lease
 - SoftAP APIs, [17](#)
- wifi_softap_set_dhcp_lease_time
 - SoftAP APIs, [18](#)
- wifi_softap_set_dhcp_offer_option
 - SoftAP APIs, [18](#)
- wifi_station_ap_change
 - Station APIs, [24](#)
- wifi_station_ap_number_set
 - Station APIs, [24](#)
- wifi_station_connect
 - Station APIs, [25](#)
- wifi_station_dhcp_start
 - Station APIs, [25](#)
- wifi_station_dhcp_status
 - Station APIs, [25](#)
- wifi_station_dhcp_stop
 - Station APIs, [26](#)
- wifi_station_disconnect
 - Station APIs, [26](#)
- wifi_station_get_ap_info
 - Station APIs, [26](#)
- wifi_station_get_auto_connect
 - Station APIs, [27](#)
- wifi_station_get_config
 - Station APIs, [27](#)
- wifi_station_get_config_default
 - Station APIs, [27](#)
- wifi_station_get_connect_status
 - Station APIs, [27](#)
- wifi_station_get_current_ap_id
 - Station APIs, [28](#)
- wifi_station_get_hostname
 - Station APIs, [28](#)
- wifi_station_get_reconnect_policy
 - Station APIs, [28](#)
- wifi_station_get_rssi
 - Station APIs, [28](#)
- wifi_station_scan
 - Station APIs, [28](#)
- wifi_station_set_auto_connect
 - Station APIs, [29](#)
- wifi_station_set_config
 - Station APIs, [29](#)
- wifi_station_set_config_current
 - Station APIs, [30](#)
- wifi_station_set_hostname
 - Station APIs, [30](#)
- wifi_station_set_reconnect_policy
 - Station APIs, [30](#)
- wifi_status_led_install
 - Common APIs, [58](#)
- wifi_status_led_uninstall
 - Common APIs, [60](#)
- wifi_unregister_rfid_locp_rcv_cb
 - Common APIs, [60](#)
- wifi_unregister_send_pkt_freedom_cb
 - Common APIs, [60](#)
- wifi_unregister_user_ie_manufacturer_rcv_cb
 - User IE APIs, [73](#)
- wifi_unregister_vnd_ie_rcv_cb
 - Vendor IE APIs, [69](#)
- wifi_wps_disable
 - WPS APIs, [79](#)
- wifi_wps_enable
 - WPS APIs, [79](#)
- wifi_wps_start
 - WPS APIs, [80](#)
- wps
 - bss_info, [134](#)
 - wps_cb_status
 - WPS APIs, [79](#)
 - wps_st_cb_t
 - WPS APIs, [78](#)
 - write_finish_fn
 - _esp_tcp, [130](#)