

# ESP8266\_RTOS\_SDK

## v1.4.0

Generated by Doxygen 1.8.10

Tue Mar 1 2016 10:19:27



# Contents

<b>1</b>	<b>ESP8266_RTOS_SDK</b>	<b>1</b>
<b>2</b>	<b>Module Index</b>	<b>3</b>
2.1	Modules . . . . .	3
<b>3</b>	<b>Data Structure Index</b>	<b>5</b>
3.1	Data Structures . . . . .	5
<b>4</b>	<b>Module Documentation</b>	<b>7</b>
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_recv(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 *lease)	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 *lease)	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	23
4.7.3.1	STATION_STATUS	23
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)	24
4.7.4.4	wifi_station_dhcpc_start(void)	24
4.7.4.5	wifi_station_dhcpc_status(void)	25
4.7.4.6	wifi_station_dhcpc_stop(void)	25
4.7.4.7	wifi_station_disconnect(void)	25
4.7.4.8	wifi_station_get_ap_info(struct station_config config[])	26
4.7.4.9	wifi_station_get_auto_connect(void)	26
4.7.4.10	wifi_station_get_config(struct station_config *config)	26
4.7.4.11	wifi_station_get_config_default(struct station_config *config)	26
4.7.4.12	wifi_station_get_connect_status(void)	27
4.7.4.13	wifi_station_get_current_ap_id(void)	27
4.7.4.14	wifi_station_get_hostname(void)	27
4.7.4.15	wifi_station_get_reconnect_policy(void)	27
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)	28
4.7.4.18	wifi_station_set_auto_connect(bool set)	28
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)	29
4.7.4.21	wifi_station_set_hostname(char *name)	30
4.7.4.22	wifi_station_set_reconnect_policy(bool set)	31
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)	36
4.8.3.10	system_get_vdd33(void)	36
4.8.3.11	system_param_load(uint16 start_sec, uint16 offset, void *param, uint16 len)	36
4.8.3.12	system_param_save_with_protect(uint16 start_sec, void *param, uint16 len)	37
4.8.3.13	system_phy_set_max_tpw(uint8 max_tpw)	37
4.8.3.14	system_phy_set_rfoption(uint8 option)	38

4.8.3.15	system_phy_set_tpw_via_vdd33(uint16 vdd33)	38
4.8.3.16	system_print_meminfo(void)	38
4.8.3.17	system_restart(void)	39
4.8.3.18	system_restore(void)	39
4.8.3.19	system_rtc_clock_calib_proc(void)	39
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)	40
4.8.3.22	system_uart_de_swap(void)	40
4.8.3.23	system_uart_swap(void)	41
4.9	Boot APIs	42
4.9.1	Detailed Description	42
4.9.2	Macro Definition Documentation	42
4.9.2.1	SYS_BOOT_ENHANCE_MODE	42
4.9.2.2	SYS_BOOT_NORMAL_BIN	42
4.9.2.3	SYS_BOOT_NORMAL_MODE	43
4.9.2.4	SYS_BOOT_TEST_BIN	43
4.9.3	Enumeration Type Documentation	43
4.9.3.1	flash_size_map	43
4.9.4	Function Documentation	43
4.9.4.1	system_get_boot_mode(void)	43
4.9.4.2	system_get_boot_version(void)	43
4.9.4.3	system_get_cpu_freq(void)	43
4.9.4.4	system_get_flash_size_map(void)	44
4.9.4.5	system_get_userbin_addr(void)	44
4.9.4.6	system_restart_enhance(uint8 bin_type, uint32 bin_addr)	44
4.9.4.7	system_update_cpu_freq(uint8 freq)	45
4.10	Software timer APIs	46
4.10.1	Detailed Description	46
4.10.2	Function Documentation	46
4.10.2.1	os_timer_arm(os_timer_t *ptimer, uint32 msec, bool repeat_flag)	46
4.10.2.2	os_timer_disarm(os_timer_t *ptimer)	46
4.10.2.3	os_timer_setfn(os_timer_t *ptimer, os_timer_func_t *pfunction, void *parg)	47
4.11	Common APIs	48
4.11.1	Detailed Description	50
4.11.2	Typedef Documentation	50
4.11.2.1	freedom_outside_cb_t	50
4.11.2.2	rfid_locp_cb_t	50
4.11.2.3	wifi_event_handler_cb_t	50
4.11.3	Enumeration Type Documentation	50
4.11.3.1	AUTH_MODE	50

4.11.3.2	SYSTEM_EVENT	51
4.11.3.3	WIFI_INTERFACE	51
4.11.3.4	WIFI_MODE	51
4.11.3.5	WIFI_PHY_MODE	51
4.11.4	Function Documentation	51
4.11.4.1	wifi_get_ip_info(WIFI_INTERFACE if_index, struct ip_info *info)	51
4.11.4.2	wifi_get_macaddr(WIFI_INTERFACE if_index, uint8 *macaddr)	52
4.11.4.3	wifi_get_opmode(void)	52
4.11.4.4	wifi_get_opmode_default(void)	52
4.11.4.5	wifi_get_phy_mode(void)	53
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)	54
4.11.4.9	wifi_rfid_locp_rcv_close(void)	54
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)	55
4.11.4.12	wifi_set_event_handler_cb(wifi_event_handler_cb_t cb)	55
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)	56
4.11.4.15	wifi_set_opmode(WIFI_MODE opmode)	57
4.11.4.16	wifi_set_opmode_current(WIFI_MODE opmode)	57
4.11.4.17	wifi_set_phy_mode(WIFI_PHY_MODE mode)	57
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)	58
4.11.4.20	wifi_status_led_uninstall(void)	58
4.11.4.21	wifi_unregister_rfid_locp_rcv_cb(void)	58
4.11.4.22	wifi_unregister_send_pkt_freedom_cb(void)	59
4.12	Force Sleep APIs	60
4.12.1	Detailed Description	60
4.12.2	Function Documentation	60
4.12.2.1	wifi_fpm_close(void)	60
4.12.2.2	wifi_fpm_do_sleep(uint32 sleep_time_in_us)	60
4.12.2.3	wifi_fpm_do_wakeup(void)	61
4.12.2.4	wifi_fpm_get_sleep_type(void)	61
4.12.2.5	wifi_fpm_open(void)	61
4.12.2.6	wifi_fpm_set_sleep_type(sleep_type type)	62
4.12.2.7	wifi_fpm_set_wakeup_cb(fpm_wakeup_cb cb)	62
4.13	Rate Control APIs	63
4.13.1	Detailed Description	64
4.13.2	Function Documentation	64

4.13.2.1	wifi_get_user_fixed_rate(uint8 *enable_mask, uint8 *rate)	64
4.13.2.2	wifi_get_user_limit_rate_mask(void)	64
4.13.2.3	wifi_set_user_fixed_rate(uint8 enable_mask, uint8 rate)	64
4.13.2.4	wifi_set_user_limit_rate_mask(uint8 enable_mask)	65
4.13.2.5	wifi_set_user_rate_limit(uint8 mode, uint8 ifidx, uint8 max, uint8 min)	65
4.13.2.6	wifi_set_user_sup_rate(uint8 min, uint8 max)	66
4.14	User IE APIs	67
4.14.1	Detailed Description	67
4.14.2	Typedef Documentation	67
4.14.2.1	user_ie_manufacturer_rcv_cb_t	67
4.14.3	Function Documentation	68
4.14.3.1	wifi_register_user_ie_manufacturer_rcv_cb(user_ie_manufacturer_rcv_cb_t cb)	68
4.14.3.2	wifi_set_user_ie(bool enable, uint8 *m_oui, user_ie_type type, uint8 *user_ie, uint8 len)	69
4.14.3.3	wifi_unregister_user_ie_manufacturer_rcv_cb(void)	69
4.15	Sniffer APIs	70
4.15.1	Detailed Description	70
4.15.2	Typedef Documentation	70
4.15.2.1	wifi_promiscuous_cb_t	70
4.15.3	Function Documentation	70
4.15.3.1	wifi_get_channel(void)	70
4.15.3.2	wifi_promiscuous_enable(uint8 promiscuous)	71
4.15.3.3	wifi_promiscuous_set_mac(const uint8_t *address)	71
4.15.3.4	wifi_set_channel(uint8 channel)	71
4.15.3.5	wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb)	72
4.16	WPS APIs	73
4.16.1	Detailed Description	73
4.16.2	Typedef Documentation	73
4.16.2.1	wps_st_cb_t	73
4.16.3	Enumeration Type Documentation	74
4.16.3.1	wps_cb_status	74
4.16.4	Function Documentation	74
4.16.4.1	wifi_set_wps_cb(wps_st_cb_t cb)	74
4.16.4.2	wifi_wps_disable(void)	74
4.16.4.3	wifi_wps_enable(WPS_TYPE_t wps_type)	74
4.16.4.4	wifi_wps_start(void)	75
4.17	Network Espconn APIs	76
4.17.1	Detailed Description	78
4.17.2	Macro Definition Documentation	78
4.17.2.1	ESPCONN_ABRT	78



4.17.2.2	ESPCONN_ARG . . . . .	78
4.17.2.3	ESPCONN_CLSD . . . . .	78
4.17.2.4	ESPCONN_CONN . . . . .	78
4.17.2.5	ESPCONN_IF . . . . .	78
4.17.2.6	ESPCONN_INPROGRESS . . . . .	78
4.17.2.7	ESPCONN_ISCONN . . . . .	78
4.17.2.8	ESPCONN_MAXNUM . . . . .	79
4.17.2.9	ESPCONN_MEM . . . . .	79
4.17.2.10	ESPCONN_OK . . . . .	79
4.17.2.11	ESPCONN_RST . . . . .	79
4.17.2.12	ESPCONN_RTE . . . . .	79
4.17.2.13	ESPCONN_TIMEOUT . . . . .	79
4.17.3	Typedef Documentation . . . . .	79
4.17.3.1	dns_found_callback . . . . .	79
4.17.3.2	espconn_connect_callback . . . . .	79
4.17.3.3	espconn_reconnect_callback . . . . .	80
4.17.3.4	espconn_rcv_callback . . . . .	80
4.17.4	Enumeration Type Documentation . . . . .	80
4.17.4.1	espconn_level . . . . .	80
4.17.4.2	espconn_option . . . . .	81
4.17.4.3	espconn_state . . . . .	81
4.17.4.4	espconn_type . . . . .	81
4.17.5	Function Documentation . . . . .	81
4.17.5.1	espconn_accept(struct espconn *espconn) . . . . .	81
4.17.5.2	espconn_clear_opt(struct espconn *espconn, uint8 opt) . . . . .	82
4.17.5.3	espconn_connect(struct espconn *espconn) . . . . .	82
4.17.5.4	espconn_create(struct espconn *espconn) . . . . .	82
4.17.5.5	espconn_delete(struct espconn *espconn) . . . . .	83
4.17.5.6	espconn_disconnect(struct espconn *espconn) . . . . .	83
4.17.5.7	espconn_dns_setserver(char numdns, ip_addr_t *dnsserver) . . . . .	83
4.17.5.8	espconn_get_connection_info(struct espconn *pespconn, remot_info **pcon_↔ info, uint8 typeflags) . . . . .	84
4.17.5.9	espconn_get_keepalive(struct espconn *espconn, uint8 level, void *optarg) . . . . .	84
4.17.5.10	espconn_gethostbyname(struct espconn *pespconn, const char *hostname, ip_↔ _addr_t *addr, dns_found_callback found) . . . . .	84
4.17.5.11	espconn_igmp_join(ip_addr_t *host_ip, ip_addr_t *multicast_ip) . . . . .	85
4.17.5.12	espconn_igmp_leave(ip_addr_t *host_ip, ip_addr_t *multicast_ip) . . . . .	85
4.17.5.13	espconn_init(void) . . . . .	86
4.17.5.14	espconn_port(void) . . . . .	86
4.17.5.15	espconn_rcv_hold(struct espconn *pespconn) . . . . .	86

4.17.5.16	espconn_rcv_unhold(struct espconn *pespconn)	86
4.17.5.17	espconn_regist_connectcb(struct espconn *espconn, espconn_connect↵ callback connect_cb)	87
4.17.5.18	espconn_regist_disconcb(struct espconn *espconn, espconn_connect_callback discon_cb)	87
4.17.5.19	espconn_regist_reconcb(struct espconn *espconn, espconn_reconnect_callback recon_cb)	87
4.17.5.20	espconn_regist_recvcb(struct espconn *espconn, espconn_rcv_callback recv↵ _cb)	88
4.17.5.21	espconn_regist_sentcb(struct espconn *espconn, espconn_sent_callback sent↵ _cb)	88
4.17.5.22	espconn_regist_time(struct espconn *espconn, uint32 interval, uint8 type_flag)	88
4.17.5.23	espconn_regist_write_finish(struct espconn *espconn, espconn_connect↵ callback write_finish_fn)	89
4.17.5.24	espconn_send(struct espconn *espconn, uint8 *psent, uint16 length)	89
4.17.5.25	espconn_sendto(struct espconn *espconn, uint8 *psent, uint16 length)	90
4.17.5.26	espconn_sent(struct espconn *espconn, uint8 *psent, uint16 length)	90
4.17.5.27	espconn_set_keeplive(struct espconn *espconn, uint8 level, void *optarg)	91
4.17.5.28	espconn_set_opt(struct espconn *espconn, uint8 opt)	91
4.17.5.29	espconn_tcp_get_max_con(void)	92
4.17.5.30	espconn_tcp_get_max_con_allow(struct espconn *espconn)	92
4.17.5.31	espconn_tcp_set_max_con(uint8 num)	92
4.17.5.32	espconn_tcp_set_max_con_allow(struct espconn *espconn, uint8 num)	93
4.18	ESP-NOW APIs	94
4.18.1	Detailed Description	95
4.18.2	Typedef Documentation	95
4.18.2.1	esp_now_rcv_cb_t	95
4.18.2.2	esp_now_send_cb_t	95
4.18.3	Function Documentation	97
4.18.3.1	esp_now_add_peer(uint8 *mac_addr, uint8 role, uint8 channel, uint8 *key, uint8 key_len)	97
4.18.3.2	esp_now_deinit(void)	97
4.18.3.3	esp_now_del_peer(uint8 *mac_addr)	97
4.18.3.4	esp_now_fetch_peer(bool restart)	98
4.18.3.5	esp_now_get_cnt_info(uint8 *all_cnt, uint8 *encrypt_cnt)	98
4.18.3.6	esp_now_get_peer_channel(uint8 *mac_addr)	98
4.18.3.7	esp_now_get_peer_key(uint8 *mac_addr, uint8 *key, uint8 *key_len)	98
4.18.3.8	esp_now_get_peer_role(uint8 *mac_addr)	99
4.18.3.9	esp_now_get_self_role(void)	99
4.18.3.10	esp_now_init(void)	99
4.18.3.11	esp_now_is_peer_exist(uint8 *mac_addr)	99
4.18.3.12	esp_now_register_rcv_cb(esp_now_rcv_cb_t cb)	100

4.18.3.13	esp_now_register_send_cb(esp_now_send_cb_t cb)	100
4.18.3.14	esp_now_send(uint8 *da, uint8 *data, uint8 len)	100
4.18.3.15	esp_now_set_kok(uint8 *key, uint8 len)	101
4.18.3.16	esp_now_set_peer_channel(uint8 *mac_addr, uint8 channel)	102
4.18.3.17	esp_now_set_peer_key(uint8 *mac_addr, uint8 *key, uint8 key_len)	102
4.18.3.18	esp_now_set_peer_role(uint8 *mac_addr, uint8 role)	102
4.18.3.19	esp_now_set_self_role(uint8 role)	103
4.18.3.20	esp_now_unregister_rcv_cb(void)	103
4.18.3.21	esp_now_unregister_send_cb(void)	103
4.19	Mesh APIs	104
4.19.1	Detailed Description	104
4.19.2	Enumeration Type Documentation	105
4.19.2.1	mesh_node_type	105
4.19.2.2	mesh_status	105
4.19.3	Function Documentation	105
4.19.3.1	espconn_mesh_connect(struct espconn *usr_esp)	105
4.19.3.2	espconn_mesh_disable(espconn_mesh_callback disable_cb)	105
4.19.3.3	espconn_mesh_disconnect(struct espconn *usr_esp)	106
4.19.3.4	espconn_mesh_enable(espconn_mesh_callback enable_cb, enum mesh_type type)	106
4.19.3.5	espconn_mesh_encrypt_init(AUTH_MODE mode, uint8_t *passwd, uint8_t passwd_len)	106
4.19.3.6	espconn_mesh_get_max_hops()	107
4.19.3.7	espconn_mesh_get_node_info(enum mesh_node_type type, uint8_t **info, uint8_t *count)	107
4.19.3.8	espconn_mesh_get_status()	107
4.19.3.9	espconn_mesh_group_id_init(uint8_t *grp_id, uint16_t gid_len)	107
4.19.3.10	espconn_mesh_init()	108
4.19.3.11	espconn_mesh_local_addr(struct ip_addr *ip)	108
4.19.3.12	espconn_mesh_sent(struct espconn *usr_esp, uint8 *pdata, uint16 len)	108
4.19.3.13	espconn_mesh_set_dev_type(uint8_t dev_type)	109
4.19.3.14	espconn_mesh_set_max_hops(uint8_t max_hops)	109
4.19.3.15	espconn_mesh_set_ssid_prefix(uint8_t *prefix, uint8_t prefix_len)	109
4.20	Driver APIs	111
4.20.1	Detailed Description	111
4.21	PWM Driver APIs	112
4.21.1	Detailed Description	112
4.21.2	Function Documentation	112
4.21.2.1	pwm_get_duty(uint8 channel)	112
4.21.2.2	pwm_get_period(void)	112

4.21.2.3	<a href="#">pwm_init(uint32 period, uint32 *duty, uint32 pwm_channel_num, uint32(*pin_info_list)[3])</a>	113
4.21.2.4	<a href="#">pwm_set_duty(uint32 duty, uint8 channel)</a>	113
4.21.2.5	<a href="#">pwm_set_period(uint32 period)</a>	113
4.21.2.6	<a href="#">pwm_start(void)</a>	114
4.22	<a href="#">Smartconfig APIs</a>	115
4.22.1	<a href="#">Detailed Description</a>	115
4.22.2	<a href="#">Typedef Documentation</a>	115
4.22.2.1	<a href="#">sc_callback_t</a>	115
4.22.3	<a href="#">Enumeration Type Documentation</a>	116
4.22.3.1	<a href="#">sc_status</a>	116
4.22.3.2	<a href="#">sc_type</a>	116
4.22.4	<a href="#">Function Documentation</a>	116
4.22.4.1	<a href="#">esptouch_set_timeout(uint8 time_s)</a>	116
4.22.4.2	<a href="#">smartconfig_get_version(void)</a>	117
4.22.4.3	<a href="#">smartconfig_set_type(sc_type type)</a>	117
4.22.4.4	<a href="#">smartconfig_start(sc_callback_t cb,...)</a>	117
4.22.4.5	<a href="#">smartconfig_stop(void)</a>	118
4.23	<a href="#">SPI Driver APIs</a>	119
4.23.1	<a href="#">Detailed Description</a>	119
4.23.2	<a href="#">Macro Definition Documentation</a>	119
4.23.2.1	<a href="#">SPI_FLASH_SEC_SIZE</a>	119
4.23.3	<a href="#">Typedef Documentation</a>	120
4.23.3.1	<a href="#">user_spi_flash_read</a>	120
4.23.4	<a href="#">Enumeration Type Documentation</a>	120
4.23.4.1	<a href="#">SpiFlashOpResult</a>	120
4.23.5	<a href="#">Function Documentation</a>	120
4.23.5.1	<a href="#">spi_flash_erase_sector(uint16 sec)</a>	120
4.23.5.2	<a href="#">spi_flash_get_id(void)</a>	120
4.23.5.3	<a href="#">spi_flash_read(uint32 src_addr, uint32 *des_addr, uint32 size)</a>	121
4.23.5.4	<a href="#">spi_flash_read_status(uint32 *status)</a>	121
4.23.5.5	<a href="#">spi_flash_set_read_func(user_spi_flash_read read)</a>	121
4.23.5.6	<a href="#">spi_flash_write(uint32 des_addr, uint32 *src_addr, uint32 size)</a>	121
4.23.5.7	<a href="#">spi_flash_write_status(uint32 status_value)</a>	122
4.24	<a href="#">Upgrade APIs</a>	123
4.24.1	<a href="#">Detailed Description</a>	123
4.24.2	<a href="#">Macro Definition Documentation</a>	124
4.24.2.1	<a href="#">SPI_FLASH_SEC_SIZE</a>	124
4.24.2.2	<a href="#">UPGRADE_FLAG_FINISH</a>	124
4.24.2.3	<a href="#">UPGRADE_FLAG_IDLE</a>	124

4.24.2.4	UPGRADE_FLAG_START	124
4.24.2.5	UPGRADE_FW_BIN1	124
4.24.2.6	UPGRADE_FW_BIN2	124
4.24.2.7	USER_BIN1	124
4.24.2.8	USER_BIN2	124
4.24.3	Typedef Documentation	124
4.24.3.1	upgrade_states_check_callback	124
4.24.4	Function Documentation	125
4.24.4.1	system_upgrade(uint8 *data, uint32 len)	125
4.24.4.2	system_upgrade_deinit()	126
4.24.4.3	system_upgrade_flag_check()	126
4.24.4.4	system_upgrade_flag_set(uint8 flag)	126
4.24.4.5	system_upgrade_init()	127
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
4.25	GPIO Driver APIs	128
4.25.1	Detailed Description	128
4.25.2	Macro Definition Documentation	128
4.25.2.1	GPIO_AS_INPUT	128
4.25.2.2	GPIO_AS_OUTPUT	129
4.25.2.3	GPIO_DIS_OUTPUT	129
4.25.2.4	GPIO_INPUT_GET	129
4.25.2.5	GPIO_OUTPUT	129
4.25.2.6	GPIO_OUTPUT_SET	130
4.25.3	Function Documentation	130
4.25.3.1	gpio16_input_conf(void)	130
4.25.3.2	gpio16_input_get(void)	130
4.25.3.3	gpio16_output_conf(void)	130
4.25.3.4	gpio16_output_set(uint8 value)	131
4.25.3.5	gpio_input_get(void)	131
4.25.3.6	gpio_intr_handler_register(void *fn, void *arg)	131
4.25.3.7	gpio_output_conf(uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)	131
4.25.3.8	gpio_pin_intr_state_set(uint32 i, GPIO_INT_TYPE intr_state)	132
4.25.3.9	gpio_pin_wakeup_disable()	132
4.25.3.10	gpio_pin_wakeup_enable(uint32 i, GPIO_INT_TYPE intr_state)	132
4.26	Hardware timer APIs	133
4.26.1	Detailed Description	133
4.26.2	Function Documentation	133

4.26.2.1	hw_timer_arm(uint32 val)	133
4.26.2.2	hw_timer_init(uint8 req)	133
4.26.2.3	hw_timer_set_func(void(*user_hw_timer_cb_set)(void))	133
4.27	UART Driver APIs	135
4.27.1	Detailed Description	135
4.27.2	Function Documentation	135
4.27.2.1	UART_ClearIntrStatus(UART_Port uart_no, uint32 clr_mask)	135
4.27.2.2	uart_init_new(void)	136
4.27.2.3	UART_intr_handler_register(void *fn, void *arg)	136
4.27.2.4	UART_IntrConfig(UART_Port uart_no, UART_IntrConfTypeDef *pUARTIntrConf)	136
4.27.2.5	UART_ParamConfig(UART_Port uart_no, UART_ConfigTypeDef *pUARTConfig)	136
4.27.2.6	UART_ResetFifo(UART_Port uart_no)	137
4.27.2.7	UART_SetBaudrate(UART_Port uart_no, uint32 baud_rate)	137
4.27.2.8	UART_SetFlowCtrl(UART_Port uart_no, UART_HwFlowCtrl flow_ctrl, uint8 rx_↔ thresh)	137
4.27.2.9	UART_SetIntrEna(UART_Port uart_no, uint32 ena_mask)	137
4.27.2.10	UART_SetLineInverse(UART_Port uart_no, UART_LineLevelInverse inverse_↔ mask)	138
4.27.2.11	UART_SetParity(UART_Port uart_no, UART_ParityMode Parity_mode)	138
4.27.2.12	UART_SetPrintPort(UART_Port uart_no)	138
4.27.2.13	UART_SetStopBits(UART_Port uart_no, UART_StopBits bit_num)	138
4.27.2.14	UART_SetWordLength(UART_Port uart_no, UART_WordLength len)	139
4.27.2.15	UART_WaitTxFifoEmpty(UART_Port uart_no)	139
<b>5</b>	<b>Data Structure Documentation</b>	<b>141</b>
5.1	_esp_event Struct Reference	141
5.1.1	Field Documentation	141
5.1.1.1	event_id	141
5.1.1.2	event_info	141
5.2	_esp_tcp Struct Reference	141
5.2.1	Field Documentation	142
5.2.1.1	connect_callback	142
5.2.1.2	disconnect_callback	142
5.2.1.3	local_ip	142
5.2.1.4	local_port	142
5.2.1.5	reconnect_callback	142
5.2.1.6	remote_ip	142
5.2.1.7	remote_port	142
5.2.1.8	write_finish_fn	142
5.3	_esp_udp Struct Reference	142
5.3.1	Field Documentation	143

5.3.1.1	<a href="#">local_ip</a>	143
5.3.1.2	<a href="#">local_port</a>	143
5.3.1.3	<a href="#">remote_ip</a>	143
5.3.1.4	<a href="#">remote_port</a>	143
5.4	<a href="#">_os_timer_t Struct Reference</a>	143
5.5	<a href="#">_remot_info Struct Reference</a>	143
5.5.1	<a href="#">Field Documentation</a>	143
5.5.1.1	<a href="#">remote_ip</a>	143
5.5.1.2	<a href="#">remote_port</a>	144
5.5.1.3	<a href="#">state</a>	144
5.6	<a href="#">airkiss_config_t Struct Reference</a>	144
5.7	<a href="#">bss_info Struct Reference</a>	144
5.7.1	<a href="#">Member Function Documentation</a>	144
5.7.1.1	<a href="#">STAILQ_ENTRY(bss_info) next</a>	144
5.7.2	<a href="#">Field Documentation</a>	145
5.7.2.1	<a href="#">authmode</a>	145
5.7.2.2	<a href="#">bssid</a>	145
5.7.2.3	<a href="#">channel</a>	145
5.7.2.4	<a href="#">freq_offset</a>	145
5.7.2.5	<a href="#">is_hidden</a>	145
5.7.2.6	<a href="#">rssi</a>	145
5.7.2.7	<a href="#">ssid</a>	145
5.7.2.8	<a href="#">ssid_len</a>	145
5.8	<a href="#">cmd_s Struct Reference</a>	145
5.9	<a href="#">dhcps_lease Struct Reference</a>	146
5.9.1	<a href="#">Field Documentation</a>	146
5.9.1.1	<a href="#">enable</a>	146
5.9.1.2	<a href="#">end_ip</a>	146
5.9.1.3	<a href="#">start_ip</a>	146
5.10	<a href="#">esp_spiffs_config Struct Reference</a>	146
5.10.1	<a href="#">Field Documentation</a>	146
5.10.1.1	<a href="#">cache_buf_size</a>	146
5.10.1.2	<a href="#">fd_buf_size</a>	146
5.10.1.3	<a href="#">log_block_size</a>	146
5.10.1.4	<a href="#">log_page_size</a>	147
5.10.1.5	<a href="#">phys_addr</a>	147
5.10.1.6	<a href="#">phys_erase_block</a>	147
5.10.1.7	<a href="#">phys_size</a>	147
5.11	<a href="#">espconn Struct Reference</a>	147
5.11.1	<a href="#">Detailed Description</a>	147

5.11.2	Field Documentation	147
5.11.2.1	link_cnt	147
5.11.2.2	recv_callback	147
5.11.2.3	reserve	148
5.11.2.4	sent_callback	148
5.11.2.5	state	148
5.11.2.6	type	148
5.12	Event_Info_u Union Reference	148
5.12.1	Field Documentation	148
5.12.1.1	ap_probereqrecved	148
5.12.1.2	auth_change	148
5.12.1.3	connected	148
5.12.1.4	disconnected	148
5.12.1.5	got_ip	149
5.12.1.6	scan_done	149
5.12.1.7	sta_connected	149
5.12.1.8	sta_disconnected	149
5.13	Event_SoftAPMode_ProbeReqRecved_t Struct Reference	149
5.13.1	Field Documentation	149
5.13.1.1	mac	149
5.13.1.2	rss	149
5.14	Event_SoftAPMode_StaConnected_t Struct Reference	149
5.14.1	Field Documentation	150
5.14.1.1	aid	150
5.14.1.2	mac	150
5.15	Event_SoftAPMode_StaDisconnected_t Struct Reference	150
5.15.1	Field Documentation	150
5.15.1.1	aid	150
5.15.1.2	mac	150
5.16	Event_StaMode_AuthMode_Change_t Struct Reference	150
5.16.1	Field Documentation	150
5.16.1.1	new_mode	150
5.16.1.2	old_mode	151
5.17	Event_StaMode_Connected_t Struct Reference	151
5.17.1	Field Documentation	151
5.17.1.1	bssid	151
5.17.1.2	channel	151
5.17.1.3	ssid	151
5.17.1.4	ssid_len	151
5.18	Event_StaMode_Disconnected_t Struct Reference	151



5.18.1	Field Documentation	152
5.18.1.1	bssid	152
5.18.1.2	reason	152
5.18.1.3	ssid	152
5.18.1.4	ssid_len	152
5.19	Event_StaMode_Got_IP_t Struct Reference	152
5.19.1	Field Documentation	152
5.19.1.1	gw	152
5.19.1.2	ip	152
5.19.1.3	mask	152
5.20	Event_StaMode_ScanDone_t Struct Reference	153
5.20.1	Field Documentation	153
5.20.1.1	bss	153
5.20.1.2	status	153
5.21	GPIO_ConfigTypeDef Struct Reference	153
5.21.1	Field Documentation	153
5.21.1.1	GPIO_IntrType	153
5.21.1.2	GPIO_Mode	153
5.21.1.3	GPIO_Pin	153
5.21.1.4	GPIO_Pullup	153
5.22	ip_info Struct Reference	154
5.22.1	Field Documentation	154
5.22.1.1	gw	154
5.22.1.2	ip	154
5.22.1.3	netmask	154
5.23	pwm_param Struct Reference	154
5.23.1	Field Documentation	154
5.23.1.1	duty	154
5.23.1.2	freq	154
5.23.1.3	period	154
5.24	rst_info Struct Reference	155
5.24.1	Field Documentation	155
5.24.1.1	reason	155
5.25	scan_config Struct Reference	155
5.25.1	Field Documentation	155
5.25.1.1	bssid	155
5.25.1.2	channel	155
5.25.1.3	show_hidden	155
5.25.1.4	ssid	155
5.26	softap_config Struct Reference	156

5.26.1	Field Documentation	156
5.26.1.1	authmode	156
5.26.1.2	beacon_interval	156
5.26.1.3	channel	156
5.26.1.4	max_connection	156
5.26.1.5	password	156
5.26.1.6	ssid	156
5.26.1.7	ssid_hidden	156
5.26.1.8	ssid_len	156
5.27	SpiFlashChip Struct Reference	157
5.28	station_config Struct Reference	157
5.28.1	Field Documentation	157
5.28.1.1	bssid	157
5.28.1.2	bssid_set	157
5.28.1.3	password	157
5.28.1.4	ssid	157
5.29	station_info Struct Reference	157
5.29.1	Member Function Documentation	158
5.29.1.1	STAILQ_ENTRY(station_info) next	158
5.29.2	Field Documentation	158
5.29.2.1	bssid	158
5.29.2.2	ip	158
5.30	UART_ConfigTypeDef Struct Reference	158
5.31	UART_IntrConfTypeDef Struct Reference	158
5.32	upgrade_server_info Struct Reference	159
5.32.1	Field Documentation	159
5.32.1.1	check_cb	159
5.32.1.2	check_times	159
5.32.1.3	pre_version	159
5.32.1.4	sockaddrin	159
5.32.1.5	upgrade_flag	159
5.32.1.6	upgrade_version	159
5.32.1.7	url	159

# 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
- Mesh APIs : Mesh 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 . . . . .	48
Force Sleep APIs . . . . .	60
Rate Control APIs . . . . .	63
User IE APIs . . . . .	67
Sniffer APIs . . . . .	70
WPS APIs . . . . .	73
Smartconfig APIs . . . . .	115
Misc APIs . . . . .	11
Spiffs APIs . . . . .	19
SSC APIs . . . . .	20
System APIs . . . . .	32
Boot APIs . . . . .	42
Upgrade APIs . . . . .	123
Software timer APIs . . . . .	46
Network Espconn APIs . . . . .	76
ESP-NOW APIs . . . . .	94
Mesh APIs . . . . .	104
Driver APIs . . . . .	111
PWM Driver APIs . . . . .	112
SPI Driver APIs . . . . .	119
GPIO Driver APIs . . . . .	128
Hardware timer APIs . . . . .	133
UART Driver APIs . . . . .	135



## Chapter 3

# Data Structure Index

### 3.1 Data Structures

Here are the data structures with brief descriptions:

<a href="#">_esp_event</a>	141
<a href="#">_esp_tcp</a>	141
<a href="#">_esp_udp</a>	142
<a href="#">_os_timer_t</a>	143
<a href="#">_remot_info</a>	143
<a href="#">airkiss_config_t</a>	144
<a href="#">bss_info</a>	144
<a href="#">cmd_s</a>	145
<a href="#">dhcps_lease</a>	146
<a href="#">esp_spiffs_config</a>	146
<a href="#">espconn</a>	147
<a href="#">Event_Info_u</a>	148
<a href="#">Event_SoftAPMode_ProbeReqRecved_t</a>	149
<a href="#">Event_SoftAPMode_StaConnected_t</a>	149
<a href="#">Event_SoftAPMode_StaDisconnected_t</a>	150
<a href="#">Event_StaMode_AuthMode_Change_t</a>	150
<a href="#">Event_StaMode_Connected_t</a>	151
<a href="#">Event_StaMode_Disconnected_t</a>	151
<a href="#">Event_StaMode_Got_IP_t</a>	152
<a href="#">Event_StaMode_ScanDone_t</a>	153
<a href="#">GPIO_ConfigTypeDef</a>	153
<a href="#">ip_info</a>	154
<a href="#">pwm_param</a>	154
<a href="#">rst_info</a>	155
<a href="#">scan_config</a>	155
<a href="#">softap_config</a>	156
<a href="#">SpiFlashChip</a>	157
<a href="#">station_config</a>	157
<a href="#">station_info</a>	157
<a href="#">UART_ConfigTypeDef</a>	158
<a href="#">UART_IntrConfTypeDef</a>	158
<a href="#">upgrade_server_info</a>	159





# 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.*
- [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 <a href="#">airkiss_config_t</a>

## 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 <a href="#">airkiss_config_t</a>

## 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 [dhcp\\_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 [dhcp\\_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(\*p)(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>	<a href="#">softap_config</a> *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>	<a href="#">softap_config</a> *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>	<a href="#">dhcps_lease</a> *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>	<a href="#">softap_config</a> *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>	<a href="#">softap_config</a> *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>	<a href="#">dhcps_lease</a> *please : IP range of the ESP8266 soft-AP DHCP server.
---------------	--

## Returns

true : succeed  
false : fail

4.4.2.15 bool wifi\_softap\_set\_dhcps\_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\_dhcps\_offer\_option ( uint8 *level*, void \* *optarg* )

Set the ESP8266 soft-AP DHCP server option.

Example:

```
uint8 mode = 0;
wifi_softap_set_dhcps_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"> <li>• bit0, 0 disable the router information;</li> <li>• bit0, 1 enable the router information.</li> </ul>

## 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>	<a href="#">esp_spiffs_config</a> *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 [scan\\_config](#)
- struct [bss\\_info](#)

### Typedefs

- typedef void(\* [scan\\_done\\_cb\\_t](#)) (void \*arg, STATUS status)  
*Callback function for wifi\_station\_scan.*

### Enumerations

- 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 <a href="#">bss_info</a>
<i>STATUS</i>	status : status of scanning

##### Returns

null

#### 4.7.3 Enumeration Type Documentation

##### 4.7.3.1 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

#### 4.7.4 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>	<code>new_ap_id</code> : 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>	<code>ap_number</code> : 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_dhcpc_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\_dhcpc\_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];
int i = wifi_station_get_ap_info(config);
```

## Parameters

<i>struct</i>	<a href="#">station_config</a> 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>	<a href="#">station_config</a> *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>	<a href="#">station_config</a> *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>	<a href="#">station_config</a> config[] : 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>	<a href="#">scan_config</a> *config : configuration of scanning
<i>struct</i>	scan_done_cb_t 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>	set : If it will automatically connect to the AP when the power is on <ul style="list-style-type: none"> <li>• true : it will connect automatically</li> <li>• false: it will not connect automatically</li> </ul>
-------------	--

## 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>	<code>station_config</code> *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>	<code>station_config</code> *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.  <ul style="list-style-type: none"> <li>• 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.</li> </ul>
<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"> <li>• 0 : Radio calibration after deep-sleep wake up depends on esp_init_data_default.bin (0~127byte) byte 108.</li> <li>• 1 : Radio calibration is done after deep-sleep wake up; this increases the current consumption.</li> <li>• 2 : No radio calibration after deep-sleep wake up; this reduces the current consumption.</li> <li>• 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.</li> </ul>
--------------	--

## 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\_cal\_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 }`

### 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

### 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  $\geq 1.3$  , 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"> <li>• #define SYS_BOOT_NORMAL_BIN 0 // user1.bin or user2.bin</li> <li>• #define SYS_BOOT_TEST_BIN 1 // can only be Espressif test bin</li> </ul>
--------------	---

<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

<code>os_timer_t</code>	*ptimer : timer structure
<code>uint32_t</code>	milliseconds : Timing, unit: millisecond, range: 5 ~ 0x68DB8
<code>bool</code>	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

<code>os_timer_t</code>	*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 [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\\_ProbeReqRecvd\\_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\\_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\\_DISCON-](#)  
[NECTED](#), [EVENT\\_STAMODE\\_AUTHMODE\\_CHANGE](#),  
[EVENT\\_STAMODE\\_GOT\\_IP](#), [EVENT\\_STAMODE\\_DHCP\\_TIMEOUT](#), [EVENT\\_SOFTAPMODE\\_STACO-](#)  
[NECTED](#), [EVENT\\_SOFTAPMODE\\_STADISCONNECTED](#),  
[EVENT\\_SOFTAPMODE\\_PROBEREQRECVED](#), **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\_K**  
**EY\_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

<i>System_Event_t</i>	*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\_INTERFACE

##### Enumerator

**STATION\_IF** ESP8266 station interface  
**SOFTAP\_IF** ESP8266 soft-AP interface

#### 4.11.3.4 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

#### 4.11.3.5 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_INTERFACE</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>	<a href="#">ip_info</a> *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_INTERFACE</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_rcv_close ( void )`

Disable RFID LOCP (Location Control Protocol) .

## Parameters

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

## Returns

*null*

4.11.4.10 `sint32 wifi_rfid_locp_rcv_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>	<i>*buf</i> : pointer of packet
<i>uint16</i>	<i>len</i> : packet length
<i>bool</i>	<i>sys_seq</i> : 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↔ 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>	<a href="#">ip_info</a> *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↔ 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"> <li>• 0x01: station mode;</li> <li>• 0x02: soft-AP mode</li> <li>• 0x03: station+soft-AP mode</li> </ul>
--------------	--

##### 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"> <li>• 0x01: station mode;</li> <li>• 0x02: soft-AP mode</li> <li>• 0x03: station+soft-AP mode</li> </ul>
--------------	--

##### 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"> <li>• If sleep_time_in_us is 0xFFFFFFFF, the ESP8266 will sleep till</li> <li>• if wifi_fpm_set_sleep_type is set to be LIGHT_SLEEP_T, ESP8266 can wake up by GPIO.</li> <li>• if wifi_fpm_set_sleep_type is set to be MODEM_SLEEP_T, ESP8266 can wake up by wifi_fpm_do_wakeup.</li> </ul>
---------------	--

## 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"> <li>• 0x01 - use the fixed rate on ESP8266 station</li> <li>• 0x02 - use the fixed rate on ESP8266 soft-AP</li> <li>• 0x03 - use the fixed rate on ESP8266 station and soft-AP</li> </ul>
<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"> <li>LIMIT_RATE_MASK_NONE - disable the limitation on both ESP8266 station and soft-AP</li> <li>LIMIT_RATE_MASK_STA - enable the limitation on ESP8266 station</li> <li>LIMIT_RATE_MASK_AP - enable the limitation on ESP8266 soft-AP</li> <li>LIMIT_RATE_MASK_ALL - enable the limitation on both ESP8266 station and soft-AP</li> </ul>
--------------	---

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

<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 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.*

#### 4.14.1 Detailed Description

WiFi User IE APIs.

#### 4.14.2 Typedef Documentation

- 4.14.2.1 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

### 4.14.3 Function Documentation

#### 4.14.3.1 `uint32_t 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

#### 4.14.3.2 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"> <li>• true, enable the corresponding user IE function, all parameters below have to be set.</li> <li>• false, disable the corresponding user IE function and release the resource, only the parameter "type" below has to be set.</li> </ul>
<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

#### 4.14.3.3 void wifi\_unregister\_user\_ie\_manufacturer\_recv\_cb ( void )

Unregister user IE received callback.

## Parameters

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

## Returns

null



## 4.15 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.*

#### 4.15.1 Detailed Description

WiFi sniffer APIs.

#### 4.15.2 Typedef Documentation

##### 4.15.2.1 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

#### 4.15.3 Function Documentation

##### 4.15.3.1 uint8 [wifi\\_get\\_channel](#) ( void )

Get the channel number for sniffer functions.

**Parameters**

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

**Returns**

channel number

**4.15.3.2 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"> <li>• 0: to disable the promiscuous mode</li> <li>• 1: to enable the promiscuous mode</li> </ul>
--------------	---

**Returns**

null

**4.15.3.3 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

**4.15.3.4 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

## 4.15.3.5 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_↔ promiscuous_↔ cb_t</i>	cb : callback
--	---------------

## Returns

null

## 4.16 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.*

#### 4.16.1 Detailed Description

ESP8266 WPS APIs.

WPS can only be used when ESP8266 station is enabled.

#### 4.16.2 Typedef Documentation

##### 4.16.2.1 typedef void(\* [wps\\_st\\_cb\\_t](#)) (int status)

WPS callback.

Parameters

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

## Returns

null

## 4.16.3 Enumeration Type Documentation

## 4.16.3.1 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

## 4.16.4 Function Documentation

## 4.16.4.1 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

## 4.16.4.2 bool wifi\_wps\_disable ( void )

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

## Parameters

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

## Returns

true : succeed  
false : fail

## 4.16.4.3 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

**4.16.4.4 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

## 4.17 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\\_KEEPAVIVE](#) = 0x08, [ESPCONN\\_END](#) }
- enum [espconn\\_level](#) { [ESPCONN\\_KEEPIVIVE](#), [ESPCONN\\_KEEPIVIVL](#), [ESPCONN\\_KEEPCNT](#) }
- 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\\_sendcb](#) (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\\_rcv\\_callback](#) rcv\_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 \*`dnsserver`)

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

### 4.17.1 Detailed Description

Network espconn APIs.

### 4.17.2 Macro Definition Documentation

#### 4.17.2.1 #define ESPCONN\_ABRT -8

Connection aborted.

#### 4.17.2.2 #define ESPCONN\_ARG -12

Illegal argument.

#### 4.17.2.3 #define ESPCONN\_CLSD -10

Connection closed.

#### 4.17.2.4 #define ESPCONN\_CONN -11

Not connected.

#### 4.17.2.5 #define ESPCONN\_IF -14

UDP send error.

#### 4.17.2.6 #define ESPCONN\_INPROGRESS -5

Operation in progress.

#### 4.17.2.7 #define ESPCONN\_ISCONN -15

Already connected.

#### 4.17.2.8 #define ESPCONN\_MAXNUM -7

Total number exceeds the maximum limitation.

#### 4.17.2.9 #define ESPCONN\_MEM -1

Out of memory.

#### 4.17.2.10 #define ESPCONN\_OK 0

No error, everything OK.

#### 4.17.2.11 #define ESPCONN\_RST -9

Connection reset.

#### 4.17.2.12 #define ESPCONN\_RTE -4

Routing problem.

#### 4.17.2.13 #define ESPCONN\_TIMEOUT -3

Timeout.

### 4.17.3 Typedef Documentation

#### 4.17.3.1 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

#### 4.17.3.2 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

## 4.17.3.3 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"> <li>• ESPCONN_TIMEOUT - Timeout</li> <li>• ESPCONN_ABRT - TCP connection aborted</li> <li>• ESPCONN_RST - TCP connection abort</li> <li>• ESPCONN_CLSD - TCP connection closed</li> <li>• ESPCONN_CONN - TCP connection</li> <li>• ESPCONN_HANDSHAKE - TCP SSL handshake fail</li> <li>• ESPCONN_PROTO_MSG - SSL application invalid</li> </ul>

## Returns

null

## 4.17.3.4 typedef void(\* espconn\_recv\_callback) (void \*arg, char \*pdata, unsigned short len)

A callback prototype to inform about events for a espconn

## 4.17.4 Enumeration Type Documentation

## 4.17.4.1 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.

#### 4.17.4.2 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.

#### 4.17.4.3 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

#### 4.17.4.4 enum espconn\_type

Protocol family and type of the espconn

##### Enumerator

**ESPCONN\_INVALID** invalid type

**ESPCONN\_TCP** TCP

**ESPCONN\_UDP** UDP

### 4.17.5 Function Documentation

#### 4.17.5.1 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

4.17.5.2 `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

4.17.5.3 `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

4.17.5.4 `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

#### 4.17.5.5 `sint8 espconn_delete ( struct espconn * espconn )`

Delete a transmission.

##### Attention

Corresponding creation API :

- TCP: `espconn_accept`,
- UDP: `espconn_create`

##### Parameters

<i>struct</i>	<code>espconn *espconn</code> : 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.

#### 4.17.5.6 `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>	<code>espconn *espconn</code> : 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`

#### 4.17.5.7 `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_dhpc_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

#### 4.17.5.8 `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

#### 4.17.5.9 `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

#### 4.17.5.10 `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

4.17.5.11 `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

4.17.5.12 `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



## 4.17.5.13 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*

## 4.17.5.14 uint32 espconn\_port ( void )

Get an available port for network.

## Parameters

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

## Returns

Port number.

## 4.17.5.15 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.

## 4.17.5.16 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.

#### 4.17.5.17 `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

#### 4.17.5.18 `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

#### 4.17.5.19 `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_↔ reconnect_↔ 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

4.17.5.20 `sint8 espconn_regist_rcvcb ( struct espconn * espconn, espconn_rcv_callback rcv_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_rcv↔ _callback</i>	rcv_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

4.17.5.21 `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↔ _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

4.17.5.22 `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"> <li>If the type_flag set to be 0, please call this API after espconn_accept, before listened a TCP connection.</li> <li>If the type_flag set to be 1, the first parameter *espconn is the specific connection.</li> </ul>

## Returns

0 : succeed

Non-0 : error code

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

#### 4.17.5.23 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

#### 4.17.5.24 sint8 espconn\_send ( struct espconn \* espconn, uint8 \* psend, 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

#### 4.17.5.25 `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

#### 4.17.5.26 `sint8 espconn_send ( 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\_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

#### 4.17.5.27 `sint8 espconn_set_keepalive ( struct espconn * espconn, 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 espconn\_connect\_callback and call espconn↵\_set\_opt to enable keep alive first.

**Parameters**

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

#### 4.17.5.28 `sint8 espconn_set_opt ( struct espconn * espconn, 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>	<code>espconn *espconn</code> : the TCP connection structure
<i>uint8</i>	<code>opt</code> : option of TCP connection, refer to enum <code>espconn_option</code> <ul style="list-style-type: none"> <li>• bit 0: 1: free memory after TCP disconnection happen need not wait 2 minutes;</li> <li>• bit 1: 1: disable nagle algorithm during TCP data transmission, quiken the data transmission.</li> <li>• 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.</li> <li>• bit 3: 1: enable TCP keep alive</li> </ul>

**Returns**

0 : succeed

Non-0 : error code

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

#### 4.17.5.29 `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.

#### 4.17.5.30 `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>	<code>espconn *espconn</code> : 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`

#### 4.17.5.31 `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

**4.17.5.32 `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



## 4.18 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.*

### 4.18.1 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.

### 4.18.2 Typedef Documentation

#### 4.18.2.1 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

#### 4.18.2.2 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

## 4.18.3 Function Documentation

## 4.18.3.1 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

## 4.18.3.2 sint32 esp\_now\_deinit ( void )

Deinitialize ESP-NOW.

## Parameters

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

## Returns

0 : succeed  
Non-0 : fail

## 4.18.3.3 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

4.18.3.4 `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>	restart : 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

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

**Returns**

0 : succeed  
Non-0 : fail

4.18.3.6 `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>	*mac_addr : MAC address of target device.
--------------	---

**Returns**

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

4.18.3.7 `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

**4.18.3.8 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

**4.18.3.9 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

**4.18.3.10 sint32 esp\_now\_init ( void )**

ESP-NOW initialization.

**Parameters**

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

**Returns**

0 : succeed  
Non-0 : fail

**4.18.3.11 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

## 4.18.3.12 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

## 4.18.3.13 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

## 4.18.3.14 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

#### 4.18.3.15 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

4.18.3.16 `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

4.18.3.17 `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

4.18.3.18 `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

**4.18.3.19 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

**4.18.3.20 sint32 esp\_now\_unregister\_recv\_cb ( void )**

Unregister ESP-NOW receive callback.

**Parameters**

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

**Returns**

0 : succeed  
Non-0 : fail

**4.18.3.21 sint32 esp\_now\_unregister\_send\_cb ( void )**

Unregister ESP-NOW send callback.

**Parameters**

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

**Returns**

0 : succeed  
Non-0 : fail

## 4.19 Mesh APIs

Mesh APIs.

### Enumerations

- enum `mesh_status` {  
`MESH_DISABLE` = 0, `MESH_WIFI_CONN`, `MESH_NET_CONN`, `MESH_LOCAL_AVAIL`,  
`MESH_ONLINE_AVAIL` }
- enum `mesh_node_type` { `MESH_NODE_PARENT` = 0, `MESH_NODE_CHILD`, `MESH_NODE_ALL` }

### Functions

- bool `espconn_mesh_local_addr` (struct ip\_addr \*ip)  
*Check whether the IP address is mesh local IP address or not.*
- bool `espconn_mesh_get_node_info` (enum `mesh_node_type` type, uint8\_t \*\*info, uint8\_t \*count)  
*Get the information of mesh node.*
- bool `espconn_mesh_encrypt_init` (`AUTH_MODE` mode, uint8\_t \*passwd, uint8\_t passwd\_len)  
*Set WiFi crypton algorithm and password for mesh node.*
- bool `espconn_mesh_set_ssid_prefix` (uint8\_t \*prefix, uint8\_t prefix\_len)  
*Set prefix of SSID for mesh node.*
- bool `espconn_mesh_set_max_hops` (uint8\_t max\_hops)  
*Set max hop for mesh network.*
- bool `espconn_mesh_group_id_init` (uint8\_t \*grp\_id, uint16\_t gid\_len)  
*Set group ID of mesh node.*
- bool `espconn_mesh_set_dev_type` (uint8\_t dev\_type)  
*Set the curent device type.*
- sint8 `espconn_mesh_connect` (struct `espconn` \*usr\_esp)  
*Try to establish mesh connection to server.*
- sint8 `espconn_mesh_disconnect` (struct `espconn` \*usr\_esp)  
*Disconnect a mesh connection.*
- sint8 `espconn_mesh_get_status` ()  
*Get current mesh status.*
- sint8 `espconn_mesh_sent` (struct `espconn` \*usr\_esp, uint8 \*pdata, uint16 len)  
*Send data through mesh network.*
- uint8 `espconn_mesh_get_max_hops` ()  
*Get max hop of mesh network.*
- void `espconn_mesh_enable` (espconn\_mesh\_callback enable\_cb, enum `mesh_type` type)  
*To enable mesh network.*
- void `espconn_mesh_disable` (espconn\_mesh\_callback disable\_cb)  
*To disable mesh network.*
- void `espconn_mesh_init` ()  
*To print version of mesh.*

#### 4.19.1 Detailed Description

Mesh APIs.

## 4.19.2 Enumeration Type Documentation

### 4.19.2.1 enum mesh\_node\_type

Enumerator

**MESH\_NODE\_PARENT** get information of parent node

**MESH\_NODE\_CHILD** get information of child node(s)

**MESH\_NODE\_ALL** get information of all nodes

### 4.19.2.2 enum mesh\_status

Enumerator

**MESH\_DISABLE** mesh disabled

**MESH\_WIFI\_CONN** WiFi connected

**MESH\_NET\_CONN** TCP connection OK

**MESH\_LOCAL\_AVAIL** local mesh is available

**MESH\_ONLINE\_AVAIL** online mesh is available

## 4.19.3 Function Documentation

### 4.19.3.1 sint8 espconn\_mesh\_connect ( struct espconn \* usr\_esp )

Try to establish mesh connection to server.

Attention

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

Parameters

<i>struct</i>	espconn *usr_esp : the network connection structure, the usr_esp 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 connection according to structure espconn

### 4.19.3.2 void espconn\_mesh\_disable ( espconn\_mesh\_callback disable\_cb )

To disable mesh network.

Attention

When mesh network is disabled, the system will trigger disable\_cb.

## Parameters

<i>espconn_↔ mesh_callback</i>	disable_cb : callback function of mesh-disable
<i>enum</i>	mesh_type type : type of mesh, local or online.

## Returns

null

## 4.19.3.3 sint8 espconn\_mesh\_disconnect ( struct espconn \* usr\_esp )

Disconnect a mesh connection.

## Attention

Do not call this API in any espconn callback. If needed, please use system task to trigger espconn\_mesh\_↔ disconnect.

## Parameters

<i>struct</i>	espconn *usr_esp : 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

## 4.19.3.4 void espconn\_mesh\_enable ( espconn\_mesh\_callback enable\_cb, enum mesh\_type type )

To enable mesh network.

## Attention

1. the function should be called in user\_init.
2. if mesh node can not scan the mesh AP, it will be isolate node without trigger enable\_cb. user can use espconn\_mesh\_get\_status to get current status of node.
3. if user try to enable online mesh, but node fails to establish mesh connection the node will work with local mesh.

## Parameters

<i>espconn_↔ mesh_callback</i>	enable_cb : callback function of mesh-enable
<i>enum</i>	mesh_type type : type of mesh, local or online.

## Returns

null

## 4.19.3.5 bool espconn\_mesh\_encrypt\_init ( AUTH\_MODE mode, uint8\_t \* passwd, uint8\_t passwd\_len )

Set WiFi crypton algorithm and password for mesh node.

## Attention

The function must be called before espconn\_mesh\_enable.

## Parameters

<i>AUTH_MODE</i>	mode : crypton algrithm (WPA/WAP2/WPA_WPA2).
<i>uint8_t</i>	*passwd : password of WiFi.
<i>uint8_t</i>	passwd_len : length of password (8 <= passwd_len <= 64).

## Returns

true : succeed  
false : fail

4.19.3.6 `uint8 espconn_mesh_get_max_hops ( )`

Get max hop of mesh network.

## Parameters

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

## Returns

the current max hop of mesh

4.19.3.7 `bool espconn_mesh_get_node_info ( enum mesh_node_type type, uint8_t ** info, uint8_t * count )`

Get the information of mesh node.

## Parameters

<i>enum</i>	mesh_node_type typ : mesh node type.
<i>uint8_t</i>	**info : the information will be saved in *info.
<i>uint8_t</i>	*count : the node count in *info.

## Returns

true : succeed  
false : fail

4.19.3.8 `sint8 espconn_mesh_get_status ( )`

Get current mesh status.

## Parameters

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

## Returns

the current mesh status, please refer to enum mesh\_status.

4.19.3.9 `bool espconn_mesh_group_id_init ( uint8_t * grp_id, uint16_t gid_len )`

Set group ID of mesh node.

## Attention

The function must be called before espconn\_mesh\_enable.

## Parameters

<i>uint8_t</i>	*grp_id : group ID.
<i>uint16_t</i>	gid_len: length of group ID, now gid_len = 6.

## Returns

true : succeed  
false : fail

## 4.19.3.10 void espconn\_mesh\_init ( )

To print version of mesh.

## Parameters

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

## Returns

null

## 4.19.3.11 bool espconn\_mesh\_local\_addr ( struct ip\_addr \* ip )

Check whether the IP address is mesh local IP address or not.

## Attention

1. The range of mesh local IP address is 2.255.255.\* ~ max\_hop.255.255.\*.
2. IP pointer should not be NULL. If the IP pointer is NULL, it will return false.

## Parameters

<i>struct</i>	ip_addr *ip : IP address
---------------	--------------------------

## Returns

true : the IP address is mesh local IP address  
false : the IP address is not mesh local IP address

## 4.19.3.12 sint8 espconn\_mesh\_sent ( struct espconn \* usr\_esp, uint8 \* pdata, uint16 len )

Send data through mesh network.

## Attention

Please call espconn\_mesh\_sent after espconn\_sent\_callback of the pre-packet.

## Parameters

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

<i>uint8</i>	*pdata : pointer of data
<i>uint16</i>	len : 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

#### 4.19.3.13 bool espconn\_mesh\_set\_dev\_type ( uint8\_t dev\_type )

Set the current device type.

**Parameters**

<i>uint8_t</i>	dev_type : device type of mesh node
----------------	-------------------------------------

**Returns**

true : succeed

false : fail

#### 4.19.3.14 bool espconn\_mesh\_set\_max\_hops ( uint8\_t max\_hops )

Set max hop for mesh network.

**Attention**

The function must be called before espconn\_mesh\_enable.

**Parameters**

<i>uint8_t</i>	max_hops : max hop of mesh network (1 <= max_hops < 10, 4 is recommended).
----------------	--

**Returns**

true : succeed

false : fail

#### 4.19.3.15 bool espconn\_mesh\_set\_ssid\_prefix ( uint8\_t \* prefix, uint8\_t prefix\_len )

Set prefix of SSID for mesh node.

**Attention**

The function must be called before espconn\_mesh\_enable.



## Parameters

<i>uint8_t</i>	*prefix : prefix of SSID.
<i>uint8_t</i>	prefix_len : length of prefix (0 < passwd_len <= 22).

## Returns

true : succeed  
false : fail

## 4.20 Driver APIs

Driver APIs.

### Modules

- [PWM Driver APIs](#)  
*PWM driver APIs.*
- [SPI Driver APIs](#)  
*SPI Flash APIs.*
- [GPIO Driver APIs](#)  
*GPIO APIs.*
- [Hardware timer APIs](#)  
*Hardware timer APIs.*
- [UART Driver APIs](#)  
*UART driver 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>	<i>period</i> : pwm frequency
<i>uint32</i>	* <i>duty</i> : duty cycle
<i>uint32</i>	<i>pwm_channel_num</i> : PWM channel number
<i>uint32</i>	(* <i>pin_info_list</i> )[3] : 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>	<i>duty</i> : duty cycle
<i>uint8</i>	<i>channel</i> : 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\\_WAITING\\_FOR\\_CONFIG](#),  
[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>	status : status of SmartConfig: <ul style="list-style-type: none"> <li>• if status == SC_STATUS_GETTING_SSID_PSWD, parameter void *pdata is a pointer of sc_type, means SmartConfig type: AirKiss or ESP-TOUCH.</li> <li>• if status == SC_STATUS_LINK, parameter void *pdata is a pointer of struct <a href="#">station↔_config</a>;</li> <li>• 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.</li> <li>• otherwise, parameter void *pdata is NULL.</li> </ul>
<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>	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

## 4.23.5.4 SpiFlashOpResult spi\_flash\_read\_status ( uint32 \* status )

Read state register of SPI Flash.

## Parameters

<i>uint32</i>	*status : 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>	read : 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>	des_addr : destination address in Flash.
<i>uint32</i>	*src_addr : source address of the data.
<i>uint32</i>	size : 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"> <li>• UPGRADE_FLAG_IDLE 0x00</li> <li>• UPGRADE_FLAG_START 0x01</li> <li>• UPGRADE_FLAG_FINISH 0x02</li> </ul>
--------------	---

## 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>	<a href="#">upgrade_server_info</a> *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

## 4.25 GPIO Driver APIs

GPIO APIs.

### Macros

- `#define GPIO_OUTPUT_SET(gpio_no, bit_value) gpio_output_conf(bit_value < <gpio_no, ((~bit_value)&0x01) < <gpio_no, 1 < <gpio_no, 0)`  
*Set GPIO pin output level.*
- `#define GPIO_OUTPUT(gpio_bits, bit_value)`  
*Set GPIO pin output level.*
- `#define GPIO_DIS_OUTPUT(gpio_no) gpio_output_conf(0, 0, 0, 1 < <gpio_no)`  
*Disable GPIO pin output.*
- `#define GPIO_AS_INPUT(gpio_bits) gpio_output_conf(0, 0, 0, gpio_bits)`  
*Enable GPIO pin input.*
- `#define GPIO_AS_OUTPUT(gpio_bits) gpio_output_conf(0, 0, gpio_bits, 0)`  
*Enable GPIO pin output.*
- `#define GPIO_INPUT_GET(gpio_no) ((gpio_input_get() > >gpio_no)&BIT0)`  
*Sample the level of GPIO input.*

### Functions

- `void gpio16_output_conf (void)`  
*Enable GPIO16 output.*
- `void gpio16_output_set (uint8 value)`  
*Set GPIO16 output level.*
- `void gpio16_input_conf (void)`  
*Enable GPIO pin input.*
- `uint8 gpio16_input_get (void)`  
*Sample the value of GPIO16 input.*
- `void gpio_output_conf (uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)`  
*Configure Gpio pins out or input.*
- `void gpio_intr_handler_register (void *fn, void *arg)`  
*Register an application-specific interrupt handler for GPIO pin interrupts.*
- `void gpio_pin_wakeup_enable (uint32 i, GPIO_INT_TYPE intr_state)`  
*Configure GPIO wake up to light sleep, Only level way is effective.*
- `void gpio_pin_wakeup_disable ()`  
*Disable GPIO wake up to light sleep.*
- `void gpio_pin_intr_state_set (uint32 i, GPIO_INT_TYPE intr_state)`  
*Config interrupt types of GPIO pin.*
- `uint32 gpio_input_get (void)`  
*Sample the value of GPIO input pins and returns a bitmask.*

#### 4.25.1 Detailed Description

GPIO APIs.

#### 4.25.2 Macro Definition Documentation

##### 4.25.2.1 `#define GPIO_AS_INPUT( gpio_bits ) gpio_output_conf(0, 0, 0, gpio_bits)`

Enable GPIO pin input.

**Parameters**

<i>gpio_bits</i>	: The GPIO bit number.
------------------	------------------------

**Returns**

null

4.25.2.2 `#define GPIO_AS_OUTPUT( gpio_bits ) gpio_output_conf(0, 0, gpio_bits, 0)`

Enable GPIO pin output.

**Parameters**

<i>gpio_bits</i>	: The GPIO bit number.
------------------	------------------------

**Returns**

null

4.25.2.3 `#define GPIO_DIS_OUTPUT( gpio_no ) gpio_output_conf(0, 0, 1 << gpio_no)`

Disable GPIO pin output.

**Parameters**

<i>gpio_no</i>	: The GPIO sequence number.
----------------	-----------------------------

**Returns**

null

4.25.2.4 `#define GPIO_INPUT_GET( gpio_no ) ((gpio_input_get()) >> gpio_no) & BIT0`

Sample the level of GPIO input.

**Parameters**

<i>gpio_no</i>	: The GPIO sequence number.
----------------	-----------------------------

**Returns**

the level of GPIO input

4.25.2.5 `#define GPIO_OUTPUT( gpio_bits, bit_value )`

**Value:**

```
if(bit_value) gpio_output_conf(gpio_bits, 0, gpio_bits, 0);\nelse gpio_output_conf(0, gpio_bits, gpio_bits, 0)
```

Set GPIO pin output level.

## Parameters

<i>gpio_bits</i>	: The GPIO bit number.
<i>bit_value</i>	: GPIO pin output level.

## Returns

null

4.25.2.6 `#define GPIO_OUTPUT_SET( gpio_no, bit_value ) gpio_output_conf(bit_value < <gpio_no, ((~bit_value)&0x01) < <gpio_no, 1 < <gpio_no, 0)`

Set GPIO pin output level.

## Parameters

<i>gpio_no</i>	: The GPIO sequence number.
<i>bit_value</i>	: GPIO pin output level.

## Returns

null

## 4.25.3 Function Documentation

4.25.3.1 `void gpio16_input_conf ( void )`

Enable GPIO pin input.

## Parameters

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

## Returns

null

4.25.3.2 `uint8 gpio16_input_get ( void )`

Sample the value of GPIO16 input.

## Parameters

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

## Returns

the level of GPIO16 input.

4.25.3.3 `void gpio16_output_conf ( void )`

Enable GPIO16 output.

## Parameters

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

## Returns

*null*

**4.25.3.4 void gpio16\_output\_set ( uint8 value )**

Set GPIO16 output level.

## Parameters

<i>uint8</i>	value : GPIO16 output level.
--------------	------------------------------

## Returns

*null*

**4.25.3.5 uint32 gpio\_input\_get ( void )**

Sample the value of GPIO input pins and returns a bitmask.

## Parameters

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

## Returns

bitmask of GPIO pins input

**4.25.3.6 void gpio\_intr\_handler\_register ( void \* fn, void \* arg )**



<i>uint32</i>	set_mask : Set the output for the high bit, the corresponding bit is 1, the output of low, the corresponding bit is 0, do not change the state.
<i>uint32</i>	enable_mask : Enable Output
<i>uint32</i>	disable_mask : Enable Input

**Returns**

null

**4.25.3.8 void gpio\_pin\_intr\_state\_set ( uint32 i, GPIO\_INT\_TYPE intr\_state )**

Config interrupt types of GPIO pin.

**Parameters**

<i>uint32</i>	i : The GPIO sequence number.
<i>GPIO_INT_TYPE</i> <i>PE</i>	intr_state : GPIO interrupt types.

**Returns**

null

**4.25.3.9 void gpio\_pin\_wakeup\_disable ( )**

Disable GPIO wake up to light sleep.

**Parameters**

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

**Returns**

null

**4.25.3.10 void gpio\_pin\_wakeup\_enable ( uint32 i, GPIO\_INT\_TYPE intr\_state )**

Configure GPIO wake up to light sleep,Only level way is effective.

**Parameters**

<i>uint32</i>	i : Gpio sequence number
<i>GPIO_INT_TYPE</i> <i>PE</i>	intr_state : the level of wake up to light sleep

**Returns**

null

## 4.26 Hardware timer APIs

Hardware timer APIs.

### Functions

- void [hw\\_timer\\_init](#) (uint8 req)  
*Initialize the hardware ISR timer.*
- void [hw\\_timer\\_arm](#) (uint32 val)  
*Set a trigger timer delay to enable this timer.*
- void [hw\\_timer\\_set\\_func](#) (void(\*user\_hw\_timer\_cb\_set)(void))  
*Set timer callback function.*

### 4.26.1 Detailed Description

Hardware timer APIs.

Attention

u2 3

## Parameters

<i>uint32</i>	<div>val : Timing<ul style="list-style-type: none"><li>• In autoloading mode, range : 50 ~ 0x7fffff</li><li>• In non-autoloading mode, range : 10 ~ 0x7fffff</li></ul></div>
---------------	--

## Returns

null

## 4.27 UART Driver APIs

UART driver APIs.

### Functions

- void [UART\\_WaitTxFifoEmpty](#) (UART\_Port uart\_no)  
*Wait uart tx fifo empty, do not use it if tx flow control enabled.*
- void [UART\\_ResetFifo](#) (UART\_Port uart\_no)  
*Clear uart tx fifo and rx fifo.*
- void [UART\\_ClearIntrStatus](#) (UART\_Port uart\_no, uint32 clr\_mask)  
*Clear uart interrupt flags.*
- void [UART\\_SetIntrEna](#) (UART\_Port uart\_no, uint32 ena\_mask)  
*Enable uart interrupts .*
- void [UART\\_intr\\_handler\\_register](#) (void \*fn, void \*arg)  
*Register an application-specific interrupt handler for Uarts interrupts.*
- void [UART\\_SetPrintPort](#) (UART\_Port uart\_no)  
*Config from which serial output printf function.*
- void [UART\\_ParamConfig](#) (UART\_Port uart\_no, [UART\\_ConfigTypeDef](#) \*pUARTConfig)  
*Config Common parameters of serial ports.*
- void [UART\\_IntrConfig](#) (UART\_Port uart\_no, [UART\\_IntrConfTypeDef](#) \*pUARTIntrConf)  
*Config types of uarts.*
- void [UART\\_SetWordLength](#) (UART\_Port uart\_no, UART\_WordLength len)  
*Config the length of the uart communication data bits.*
- void [UART\\_SetStopBits](#) (UART\_Port uart\_no, UART\_StopBits bit\_num)  
*Config the length of the uart communication stop bits.*
- void [UART\\_SetParity](#) (UART\_Port uart\_no, UART\_ParityMode Parity\_mode)  
*Configure whether to open the parity.*
- void [UART\\_SetBaudrate](#) (UART\_Port uart\_no, uint32 baud\_rate)  
*Configure the Baud rate.*
- void [UART\\_SetFlowCtrl](#) (UART\_Port uart\_no, UART\_HwFlowCtrl flow\_ctrl, uint8 rx\_thresh)  
*Configure Hardware flow control.*
- void [UART\\_SetLineInverse](#) (UART\_Port uart\_no, UART\_LineLevelInverse inverse\_mask)  
*Configure triggering signal of uarts.*
- void [uart\\_init\\_new](#) (void)  
*An example illustrates how to configure the serial port.*

### 4.27.1 Detailed Description

UART driver APIs.

### 4.27.2 Function Documentation

#### 4.27.2.1 void [UART\\_ClearIntrStatus](#) ( UART\_Port *uart\_no*, uint32 *clr\_mask* )

Clear uart interrupt flags.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>uint32</i>	clr_mask : To clear the interrupt bits

## Returns

null

## 4.27.2.2 void uart\_init\_new ( void )

An example illustrates how to configure the serial port.

## Parameters

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

## Returns

null

## 4.27.2.3 void UART\_intr\_handler\_register ( void \* fn, void \* arg )

Register an application-specific interrupt handler for Uarts interrupts.

## Parameters

<i>void</i>	*fn : interrupt handler for Uart interrupts.
<i>void</i>	*arg : interrupt handler's arg.

## Returns

null

## 4.27.2.4 void UART\_IntrConfig ( UART\_Port uart\_no, UART\_IntrConfTypeDef \* pUARTIntrConf )

Config types of uarts.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<a href="#"><i>UART_IntrConfTypeDef</i></a>	*pUARTIntrConf : parameters structure

## Returns

null

## 4.27.2.5 void UART\_ParamConfig ( UART\_Port uart\_no, UART\_ConfigTypeDef \* pUARTConfig )

Config Common parameters of serial ports.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_ConfigTypeDef</i>	*pUARTConfig : parameters structure

## Returns

null

4.27.2.6 void UART\_ResetFifo ( UART\_Port *uart\_no* )

Clear uart tx fifo and rx fifo.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
------------------	--------------------------

## Returns

null

4.27.2.7 void UART\_SetBaudrate ( UART\_Port *uart\_no*, uint32 *baud\_rate* )

Configure the Baud rate.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>uint32</i>	baud_rate : the Baud rate

## Returns

null

4.27.2.8 void UART\_SetFlowCtrl ( UART\_Port *uart\_no*, UART\_HwFlowCtrl *flow\_ctrl*, uint8 *rx\_thresh* )

Configure Hardware flow control.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_HwFlowCtrl</i>	flow_ctrl : Hardware flow control mode
<i>uint8</i>	rx_thresh : threshold of Hardware flow control

## Returns

null

4.27.2.9 void UART\_SetIntrEna ( UART\_Port *uart\_no*, uint32 *ena\_mask* )

Enable uart interrupts .

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>uint32</i>	ena_mask : To enable the interrupt bits

## Returns

null

4.27.2.10 void UART\_SetLineInverse ( UART\_Port *uart\_no*, UART\_LineLevelInverse *inverse\_mask* )

Configure triggering signal of uarts.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_Line↔ LevelInverse</i>	inverse_mask : Choose need to flip the IO

## Returns

null

4.27.2.11 void UART\_SetParity ( UART\_Port *uart\_no*, UART\_ParityMode *Parity\_mode* )

Configure whether to open the parity.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_Parity↔ Mode</i>	Parity_mode : the enum of uart parity configuration

## Returns

null

4.27.2.12 void UART\_SetPrintPort ( UART\_Port *uart\_no* )

Config from which serial output printf function.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
------------------	--------------------------

## Returns

null

4.27.2.13 void UART\_SetStopBits ( UART\_Port *uart\_no*, UART\_StopBits *bit\_num* )

Config the length of the uart communication stop bits.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_StopBits</i>	bit_num : the length uart communication stop bits

## Returns

null

4.27.2.14 void UART\_SetWordLength ( UART\_Port *uart\_no*, UART\_WordLength *len* )

Config the length of the uart communication data bits.

## Parameters

<i>UART_Port</i>	uart_no : UART0 or UART1
<i>UART_WordLength</i>	len : the length of the uart communication data bits

## Returns

null

4.27.2.15 void UART\_WaitTxFifoEmpty ( UART\_Port *uart\_no* )

Wait uart tx fifo empty, do not use it if tx flow control enabled.

## Parameters

<i>UART_Port</i>	uart_no:UART0 or UART1
------------------	------------------------

## Returns

null



## 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](#)

### 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 uint8 is\_hidden

SSID of current AP is hidden or not.

### 5.7.2.6 sint8 rssi

single strength of AP

### 5.7.2.7 uint8 ssid[32]

SSID of AP

### 5.7.2.8 uint8 ssid\_len

SSID length

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 int78(rssi)]T 0 g 0 G F29 8.9663 Tf 0 -25.546 TeReeive(send)-278(probe)-278(requich)-2ignal(the)-27gtht-AP

The documentation for this(Struct)-278(w)15(as)-278(gener)10(ated)-278(from)-278(the)-278(f)30(ollo)15(wing)-278(file )]T 0 g 0 G 16.787 -2.481 Td [()])T

## Data Fields

- uint8 [mac](#)
- uint8

### 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 GPIO\_ConfigTypeDef Struct Reference

### Data Fields

- uint16 [GPIO\\_Pin](#)
- GPIOMode\_TypeDef [GPIO\\_Mode](#)
- GPIO\_Pullup\_IF [GPIO\\_Pullup](#)
- GPIO\_INT\_TYPE [GPIO\\_IntrType](#)

#### 5.21.1 Field Documentation

##### 5.21.1.1 GPIO\_INT\_TYPE [GPIO\\_IntrType](#)

GPIO interrupt type

##### 5.21.1.2 GPIOMode\_TypeDef [GPIO\\_Mode](#)

GPIO mode

##### 5.21.1.3 uint16 [GPIO\\_Pin](#)

GPIO pin

##### 5.21.1.4 [GPIO\\_Pullup\\_IF](#) [GPIO\\_Pullup](#)

GPIO pullup

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

- `examples/driver_lib/include/gpio.h`

## 5.22 ip\_info Struct Reference

### Data Fields

- struct ip\_addr [ip](#)
- struct ip\_addr [netmask](#)
- struct ip\_addr [gw](#)

### 5.22.1 Field Documentation

#### 5.22.1.1 struct ip\_addr gw

gateway

#### 5.22.1.2 struct ip\_addr ip

IP address

#### 5.22.1.3 struct ip\_addr netmask

netmask

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

- include/espressif/esp\_wifi.h

## 5.23 pwm\_param Struct Reference

### Data Fields

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

### 5.23.1 Field Documentation

#### 5.23.1.1 uint32 duty[8]

PWM duty

#### 5.23.1.2 uint32 freq

PWM frequency

#### 5.23.1.3 uint32 period

PWM period

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

- include/espressif/pwm.h

## 5.24 rst\_info Struct Reference

### Data Fields

- [rst\\_reason](#) reason
- uint32 **exccause**
- uint32 **epc1**
- uint32 **epc2**
- uint32 **epc3**
- uint32 **excvaddr**
- uint32 **depc**
- uint32 **rtn\_addr**

### 5.24.1 Field Documentation

#### 5.24.1.1 [rst\\_reason](#) reason

enum rst\_reason

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

- include/espressif/esp\_system.h

## 5.25 scan\_config Struct Reference

### Data Fields

- uint8 \* [ssid](#)
- uint8 \* [bssid](#)
- uint8 [channel](#)
- uint8 [show\\_hidden](#)

### 5.25.1 Field Documentation

#### 5.25.1.1 uint8\* [bssid](#)

MAC address of AP

#### 5.25.1.2 uint8 [channel](#)

channel, scan the specific channel

#### 5.25.1.3 uint8 [show\\_hidden](#)

enable to scan AP whose SSID is hidden

#### 5.25.1.4 uint8\* [ssid](#)

SSID of AP

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

- include/espressif/esp\_sta.h

## 5.26 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](#)

### 5.26.1 Field Documentation

#### 5.26.1.1 [AUTH\\_MODE](#) [authmode](#)

Auth mode of ESP8266 soft-AP. Do not support AUTH\_WEP in soft-AP mode

#### 5.26.1.2 [uint16](#) [beacon\\_interval](#)

Beacon interval, 100 ~ 60000 ms, default 100

#### 5.26.1.3 [uint8](#) [channel](#)

Channel of ESP8266 soft-AP

#### 5.26.1.4 [uint8](#) [max\\_connection](#)

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

#### 5.26.1.5 [uint8](#) [password](#)[64]

Password of ESP8266 soft-AP

#### 5.26.1.6 [uint8](#) [ssid](#)[32]

SSID of ESP8266 soft-AP

#### 5.26.1.7 [uint8](#) [ssid\\_hidden](#)

Broadcast SSID or not, default 0, broadcast the SSID

#### 5.26.1.8 [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](#)



## 5.27 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

## 5.28 station\_config Struct Reference

### Data Fields

- uint8 [ssid](#) [32]
- uint8 [password](#) [64]
- uint8 [bssid\\_set](#)
- uint8 [bssid](#) [6]

### 5.28.1 Field Documentation

#### 5.28.1.1 uint8 bssid[6]

MAC address of target AP

#### 5.28.1.2 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.

#### 5.28.1.3 uint8 password[64]

password of target AP

#### 5.28.1.4 uint8 ssid[32]

SSID of target AP

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

- include/espressif/esp\_sta.h

## 5.29 station\_info Struct Reference

### Public Member Functions

- [STAILQ\\_ENTRY](#) ([station\\_info](#)) next

## Data Fields

- uint8 `bssid` [6]
- struct ip\_addr `ip`

### 5.29.1 Member Function Documentation

#### 5.29.1.1 STAILQ\_ENTRY ( station\_info )

Information of next AP

### 5.29.2 Field Documentation

#### 5.29.2.1 uint8 bssid[6]

BSSID of AP

#### 5.29.2.2 struct ip\_addr ip

IP address of AP

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

- include/espressif/esp\_softap.h

## 5.30 UART\_ConfigTypeDef Struct Reference

### Data Fields

- UART\_BaudRate **baud\_rate**
- UART\_WordLength **data\_bits**
- UART\_ParityMode **parity**
- UART\_StopBits **stop\_bits**
- UART\_HwFlowCtrl **flow\_ctrl**
- uint8 **UART\_RxFlowThresh**
- uint32 **UART\_InverseMask**

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

- examples/driver\_lib/include/uart.h

## 5.31 UART\_IntrConfTypeDef Struct Reference

### Data Fields

- uint32 **UART\_IntrEnMask**
- uint8 **UART\_RX\_TimeOutIntrThresh**
- uint8 **UART\_TX\_FifoEmptyIntrThresh**
- uint8 **UART\_RX\_FifoFullIntrThresh**

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

- examples/driver\_lib/include/uart.h

## 5.32 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](#)

### 5.32.1 Field Documentation

#### 5.32.1.1 [upgrade\\_states\\_check\\_callback](#) [check\\_cb](#)

callback of upgrading

#### 5.32.1.2 uint32 [check\\_times](#)

time out of upgrading, unit : ms

#### 5.32.1.3 uint8 [pre\\_version](#)[16]

previous version of firmware

#### 5.32.1.4 struct sockaddr\_in [sockaddrin](#)

socket of upgrading

#### 5.32.1.5 uint8 [upgrade\\_flag](#)

true, upgrade succeed; false, upgrade fail

#### 5.32.1.6 uint8 [upgrade\\_version](#)[16]

the new version of firmware

#### 5.32.1.7 uint8\* [url](#)

the url of upgrading server

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

- [include/espressif/upgrade.h](#)

