# Espressif IoT SDK: Programming Guide



| | |
|---|---|
| **Status** | Released |
| **Current version** | V0.9.1 |
| **Author** | Fei Yu |
| **Completion Date** | 2014.9.23 |
| **Reviewer** | Jiangang Wu |
| **Completion Date** | 2014.9.23 |

**[ ] CONFIDENTIAL**

**[ ] INTERNAL**

**[√ ] PUBLIC**

# Version Info

| Date | Version | Author | Comments/Changes |
|---|---|---|---|
| 2013.12.25 | 0.1 | Jiangang Wu | Draft |
| 2013.12.25 | 0.1.1 | Han Liu | Revise Json APIs |
| 2014.1.15 | 0.2 | Jiangang Wu | Revise some APIs |
| 2014.1.29 | 0.3 | Han Liu | Add client/server APIs |
| 2014.3.20 | 0.4 | Jiangang Wu / Han Liu | 1.Add UART APIs;<br>2.Add i2c master APIs;<br>3.Revise client/server APIs;<br>4.Add secure espconn APIs;<br>5.Add upgrade APIs |
| 2014.4.17 | 0.5 | Jiangang Wu / Han Liu | 1.Revise espconn APIs;<br>2.Add gpio APIs instruction;<br>3.Add some other instructions; |
| 2014.5.14 | 0.6 | Jiangang Wu | Add some APIs |
| 2014.6.18 | 0.7 | Jiangang Wu | 1.Add dns APIs;<br>2.Revise save-param APIs;<br>3.Add task APIs;<br>4.Add API to get connected station's info when ESP8266 in softAP mode;<br>5. Add API to get free heap size.<br>6.Others |
| 2014.7.10 | 0.8 | Fei Yu | 1.Add OTA APIs;<br>2.Add API to ON/OFF printf;<br>3.Add wifi_station_get_connect_status<br>4.Add TCP Server(SSL) APIs |
| 2014.8.13 | 0.9 | Fei Yu | 1.Revise espconn APIs;<br>2.Add sniffer APIs;<br>3.Add system_get_chip_id;<br>4.Add APIs to read/ write mac&ip; |
| 2014.9.23 | 0.9.1 | Fei Yu | 1、Add system_deep_sleep;<br>2、Revise APIs to read/write flash;<br>3、Add APIs for AP_CHE<br>4、Revise UDP APIs |

**Disclaimer and Copyright Notice**

Information in this document, including URL references, is subject to change without notice.

THIS DOCUMENT IS PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE. All liability, including liability for infringement of any proprietary rights, relating to use of information in this document is disclaimed. No licenses express or implied, by estoppel or otherwise, to any intellectual property rights are granted herein.

The Wi-Fi Alliance Member Logo is a trademark of the Wi-Fi Alliance.

All trade names, trademarks and registered trademarks mentioned in this document are property of their respective owners, and are hereby acknowledged.

# Table of Contents

# 1. Foreword

The SDK based on ESP8266 IoT platform offers users an easy, fast and efficient way to develop IoT devices.

The programming guide provides overview of the SDK as well as details on the API. It is written for embedded software developers to help them program on ESP8266 IoT platform.

# 2. Overview

The SDK provides a set of interfaces for data receive and transmit functions over the Wi-Fi and TCP/IP layer so programmers can focus on application development on the high level. Users can easily make use of the corresponding interfaces to realize data receive and transmit.

All networking functions on the ESP8266 IoT platform are realized in the library, and are not transparent to users. Instead, users can initialize the interface in user_main.c

void usre_init(void) is the default method provided. Users can add functions like firmware initialization, network parameters setting, and timer initialization in the interface.

The SDK provides an API to handle json, and users can also use self-defined data types to handle the them.

# 3. Application Programming Interface (APIs)

## 3.1. Timer

Locate in "\esp_iot_sdk\include\osapi.h"

### 3.1.1. os_timer_arm

Function: arm timer

Prototype:

void os_timer_arm(ETSTimer *ptimer, uint32_t milliseconds, boolrepeat_flag)

Input parameters:

ETSTimer*ptimer——Timer structure

uint32_t milliseconds——Timing, Unit: milisecond

boolrepeat_flag——Whether to repeat the timing

Return:

null

### 3.1.2. os_timer_disarm

Function: Disarm timer

Prototype:

void os_timer_disarm (ETSTimer *ptimer)

Input parameters:

ETSTimer*ptimer——Timer structure

Return:

null

### 3.1.3. os_timer_setfn

Function: Set timer callback function

Prototype:

void os_timer_setfn (ETSTimer *ptimer, ETSTimerFunc *pfunction, void *parg)

Input parameters:

ETSTimer*ptimer——Timer structure

TESTimerFunc*pfunction——timer callback function

void*parg——callback function parameter

Return:

null

## 3.2. System APIs

Locate in "\esp_iot_sdk \include\user_interface.h"。

### 3.2.1. system_restore

Function: Reset to default settings

Prototype:

void system_restore(void)

Input parameters:

null

Return:

null

### 3.2.2. system_restart

Function: Restart

Prototype:

void system_restart(void)

Input parameters:

null

Return:

null

### 3.2.3. system_get_chip_id

Function: Get chip id

Prototype:

uint32 system_get_chip_id (void)

Input parameters:

null

Return:

Chip id

### 3.2.4. system_deep_sleep

Function: Set for deep-sleep mode. Device in deep-sleep mode automatically, every X us wake up once. Everytime device wakes up, it starts from user_init.

Prototype：

void system_deep_sleep(uint32 time_in_us)

parameters：

uint32 time_in_us – during the time (us) device is in deep-sleep

Return：

NULL

### 3.2.5. system_upgrade_userbin_check

Function: Check userbin

Function definition:

uint8 system_upgrade_userbin_check()

Input parameters:

null

Return:

0x00 : UPGRADE_FW_BIN1 , i.e., user1.bin

0x01 : UPGRADE_FW_BIN2 , i.e., user2.bin

### 3.2.6. system_upgrade_start

Function: Configure parameters and start upgrade

Function definition:

bool system_upgrade_start (struct upgrade_server_info *server)

Parameters:

struct upgrade_server_info *server – server related parameters

Return

true: start upgrade

false: upgrade can't be started.

### 3.2.7. system_upgrade_reboot

Function: reboot system and use new version

Function definition:

void system_upgrade_reboot (void)

Input parameters:

null

Return:

null

### 3.2.8. system_set_os_print

Function: Turn on/off print logFunction

Function definition:

void system_set_os_print (uint8onoff)

Input parameters:

uint8 onoff — turn on/off print function;

0x00 : print function off

0x01: print function on

Defaut: print function on

Return:

null

### 3.2.9. system_timer_reinit

Function: Reinitiate the timer when you need to use microsecond timer

Not es: 1. Define USE_US_TIMER;

2. Put system_timer_reinit at the beginning and user_init in the first sentence.

Function definition:

void system_timer_reinit (void)

Input parameters:

null

Return:

null

### 3.2.10. system_print_meminfo

Function: Print memory information, including data/rodata/bss/heap

Function definition:

void system_print_meminfo (void)

Input parameters:

null

Return:

null

### 3.2.11. system_get_free_heap_size

Function: Get free heap size

Function definition:

uint32 system_get_free_heap_size(void)

Input parameters:

null

Return:

uint32 ——available heap size

### 3.2.12. system_os_task

Function: Set up tasks

Function definition:

void system_os_task(os_task_t task, uint8 prio, os_event_t *queue, uint8 qlen)

Input parameters:

os_task_t task——task function

uint8 prio——task priority. 3 priorities are supported, 0/1/2, 0 is the lowest priority.

os_event_t *queue——message queue pointer

uint8 qlen——message queue depth

Return:

null

Example:

```
#define SIG_RX  0
#define TEST_QUEUE_LEN    4
os_event_t *testQueue;
void test_task (os_event_t *e)
{
```

```
    switch (e->sig) {
case SIG_RX:
    os_printf("sig_rx %c\n",    (char)e->par);
    break;
default:
break;
    }
}
void task_init(void)
{
lwipIf0EvtQueue = (os_event_t *)os_malloc(sizeof(os_event_t)*TEST_QUEUE_LEN);
system_os_task(test_task,    USER_TASK_PRIO_0,    testQueue,    TEST_QUEUE_LEN);
}
```

### 3.2.13. system_os_post

Function: send message to task

Function definition:

    void system_os_post (uint8 prio,    os_signal_t sig,    os_param_t par)

Input parameters:

    uint8 prio——task priority,    corresponding to that you set up

    os_signal_t sig——message type

    os_param_t par——message parameters

Return:

    null

Refer to the example above:

```
void task_post(void)
{
system_os_post(USER_TASK_PRIO_0,    SIG_RX,    'a');
}
```

Printout: sig_rx a

## 3.2.14. spi_flash_erase_sector

Function: erase sector in flash

Note: More details in document “”

Prototype：

SpiFlashOpResult spi_flash_erase_sector (uint16 sec)

Parameters：

uint16 sec – Sector number, the count starts at sector 0, 4KB per sector.

Return：

Typedef enum{

SPI_FLASH_RESULT_OK,

SPI_FLASH_RESULT_ERR,

SPI_FLASH_RESULT_TIMEOUT

}SpiFlashOpResult;

## 3.2.15. spi_flash_write

Function： Write data to flash。

Note: More details in document “”

Prototype：

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

Parameters：

uint32 des_addr - destination address in flash.

uint32 *src_addr – source address of the data.

Uint32 size - length of data

Return：

Typedef enum{

SPI_FLASH_RESULT_OK,

```
        SPI_FLASH_RESULT_ERR,
        SPI_FLASH_RESULT_TIMEOUT
    }SpiFlashOpResult;
```

### 3.2.16. spi_flash_read

Function： Read data from flash.

Note: More details in document “”

Prototype：

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

Parameters：

```
    uint32 src_addr  - source address in flash
    uint32 * des_addr – destination address to keep data.
    Uint32 size  - length of data
```

Return：

```
    Typedef enum{
        SPI_FLASH_RESULT_OK,
        SPI_FLASH_RESULT_ERR,
        SPI_FLASH_RESULT_TIMEOUT
    }SpiFlashOpResult;
```

### 3.2.17. wifi_get_opmode

Function: get wifi working mode

Function definition:

```
    uint8 wifi_get_opmode (void)
```

Input parameters:

```
    null
```

Return:

Wifi working modes:

0x01 means STATION_MODE,

0x02 means SOFTAP_MODE,

0x03 means STATIONAP_MODE.

### 3.2.18. wifi_set_opmode

Function: set wifi working mode as STATION, SOFTAP or STATION+SOFTAP

Note:

Versions before esp_iot_sdk_v0.9.2, need to call system_restart() after this api;

after esp_iot_sdk_v0.9.2, need not to restart.

Function definition:

bool wifi_set_opmode (uint8 opmode)

Input parameters:

uint8 opmode——Wifi working modes: 0x01 means STATION_MODE, 0x02 means SOFTAP_MODE, 0x03 means STATIONAP_MODE.

Return:

True - succeed;

False - fail.

### 3.2.19. wifi_station_get_config

Function: get wifi station configuration

Function definition:

bool wifi_station_get_config (struct station_config *config)

Input parameters:

struct station_config *config——wifi station configuration pointer

Return:

True - succeed;

False - fail.

### 3.2.20. wifi_station_set_config

Function: Set wifi station configuration

Note: If wifi_station_set_config is called in user_init , there is no need to call wifi_station_connect after that, ESP8266 will connect to router automatically; otherwise, need wifi_station_connect to connect.

Function definition:

bool wifi_station_set_config (struct station_config *config)

Input parameters:

struct station_config *config——wifi station configuration pointer

Return:

True - succeed；

False - fail.

### 3.2.21. wifi_station_connect

Function: wifi station connected AP

Note： if ESP8266 has already connected to a router， it‘s necessary to call wifi_station_disconnect first，then call wifi_station_connect to connect.

Function definition:

bool wifi_station_connect (void)

Input parameters:

null

Return:

True - succeed；

False - fail.

## 3.2.22. wifi_station_disconnect

Function: wifi station disconnected AP

Function definition:

bool wifi_station_disconnect (void)

Input parameters:

null

Return:

True - succeed；

False - fail.

## 3.2.23. wifi_station_get_connect_status

Function: get the connection status between wifi station and AP

Function definition:

uint8 wifi_station_get_connect_status (void)

Input parameters:

null

Return:

```
enum{
    STATION_IDLE = 0,
    STATION_CONNECTING,
    STATION_WRONG_PASSWORD,
    STATION_NO_AP_FOUND,
    STATION_CONNECT_FAIL,
    STATION_GOT_IP
};
```

### 3.2.24. wifi_station_scan

Function: Scan AP

Function definition:

bool wifi_station_scan (struct scan_config *config, scan_done_cb_t cb);

Structure：

```
struct scan_config{
    uint8 *ssid;          // AP's ssid
    uint8 *bssid;         // AP's bssid
    uint8 channel;        //scan a specific channel
};
```

Parameters:

struct scan_config *config – AP config for scan

if config = Null， scan all APs

if config.ssid、config.bssid are null，config.channel isn't null, ESP8266 will scan the specific channel.

scan_done_cb_t cb - callback function after scan

Return:

True - succeed；

False - fail.

### 3.2.25. scan_done_cb_t

Function: scan callback function

Function definition:

void scan_done_cb_t (void *arg, STATUS status);

Input parameters:

void *arg——get Ap's input parameters

STATUS status——get status

Return:

null

## 3.2.26. wifi_station_ap_number_set

Function: Set the number of APs that can be recorded for ESP8266 station. When ESP8266 station connects to an AP, ESP8266 keeps a record of this AP. Record id starts counting from 0.

Prototype:

bool wifi_station_ap_number_set (uint8 ap_number);

Parameters:

uint8 ap_number—— how many APs can be recorded（MAX: 5）

eg: if ap_number is 5, record id : 0 ~ 4

Return:

True - succeed;

False - fail.

## 3.2.27. wifi_station_ap_change

Function: ESP8266 station change to connect to the AP which is recorded in specific id.

Prototype:

bool wifi_station_ap_change (uint8 current_ap_id);

Parameters:

uint8 current_ap_id — AP's record id, start counting from 0.

Return:

True - succeed;

False - fail.

## 3.2.28. wifi_station_get_current_ap_id

Function: Get the current record id of AP.

Prototype：

Uint8 wifi_station_get_current_ap_id ();

Parameter：

Null

Return：

The record id of the AP which ESP8266 is connected with right now.

### 3.2.29. wifi_softap_get_config

Function: set wifi softap configuration

Function definition:

bool wifi_softap_get_config(struct softap_config *config)

Parameter：

struct softap_config *config——ESP8266 softap config

Return：

True - succeed；

False - fail.

### 3.2.30. wifi_softap_set_config

Function: set wifi softap configuration

Function definition:

bool wifi_softap_set_config (struct softap_config *config)

Parameter：

struct softap_config *config—— wifi softap configuration pointer

Return：

True - succeed；

False - fail.

### 3.2.31. wifi_softap_get_station_info

Function: get connected station devices under softap mode, including mac and ip

Function definition：

struct station_info * wifi_softap_get_station_info(void)

Input parameters:

null

Return:

struct station_info*——station information structure

### 3.2.32. wifi_softap_free_station_info

Function: free the struct station_info by calling the wifi_softap_get_station_info function

Function definition:

void wifi_softap_free_station_info(void)

Input parameters:

null

Return:

null

Examples of getting mac and ip information:

Method 1:

```
struct station_info * station = wifi_softap_get_station_info();
struct station_info * next_station;
while(station){
        os_printf("bssid : "MACSTR", ip : "IPSTR"\n", MAC2STR(station->bssid),
IP2STR(&station->ip));
        next_station = STAILQ_NEXT(station, next);
        os_free(station);      // Free it directly
        station = next_station;
}
```

Method 2：

```
struct station_info * station = wifi_softap_get_station_info();
```

```
while(station){
   os_printf("bssid : "MACSTR", ip : "IPSTR"\n", MAC2STR(station->bssid),
IP2STR(&station->ip));
   station = STAILQ_NEXT(station, next);
}
wifi_softap_free_station_info();  // Free it by calling functions
```

### 3.2.33. wifi_get_ip_info

Function: Get ip info of wifi station or softap interface

Function definition:

bool wifi_get_ip_info(uint8 if_index, struct ip_info *info)

Parameters:

uint8 if_index——the interface to get ip info: 0x00 for STATION_IF, 0x01 for SOFTAP_IF.

struct ip_info *info——pointer to get ip info of a certain interface

Return：

True - succeed；

False - fail.

### 3.2.34. wifi_set_ip_info

Function: Set ip address of ESP8266 station or softAP

Note: only can be used in user_init.

Function definition:

bool wifi_set_ip_info(uint8 if_index, struct ip_info *info)

Prototype：

uint8 if_index – set station ip or softAP ip

#define STATION_IF 0x00

#define SOFTAP_IF 0x01

struct ip_info *info – ip information

Example：

```
struct ip_info info;


IP4_ADDR(&info.ip, 192, 168, 3, 200);
IP4_ADDR(&info.gw, 192, 168, 3, 1);
IP4_ADDR(&info.netmask, 255, 255, 255, 0);
wifi_set_ip_info(STATION_IF, &info);


IP4_ADDR(&info.ip, 10, 10, 10, 1);
IP4_ADDR(&info.gw, 10, 10, 10, 1);
IP4_ADDR(&info.netmask, 255, 255, 255, 0);
wifi_set_ip_info(SOFTAP_IF, &info);
```

Return：

True - succeed；

False - fail.

## 3.2.35. wifi_set_macaddr

Function: set mac address

Note: only can be used in user_init

Function definition:

```
bool wifi_set_macaddr(uint8 if_index, uint8 *macaddr)
```

Parameter：

uint8 if_index – set station mac or softAP mac

```
#define STATION_IF      0x00
#define SOFTAP_IF       0x01
```

uint8 *macaddr – mac address

Example：

```
char sofap_mac[6] = {0x16, 0x34, 0x56, 0x78, 0x90, 0xab};
char sta_mac[6] = {0x12, 0x34, 0x56, 0x78, 0x90, 0xab};
```

wifi_set_macaddr(SOFTAP_IF, sofap_mac);

wifi_set_macaddr(STATION_IF, sta_mac);

Return：

True - succeed；

False - fail.

## 3.2.36. wifi_get_macaddr

Function: get mac address

Function definition:

Bool wifi_get_macaddr(uint8 if_index , uint8 *macaddr)

Parameter：

uint8 if_index —— set station mac or softAP mac

#define STATION_IF 0x00

#define SOFTAP_IF 0x01

uint8 *macaddr—— mac address

Return：

True - succeed;

False - fail.

## 3.2.37. wifi_status_led_install

Function: Install wifi status LED

Function definition:

Void wifi_status_led_install (uint8 gpio_id, uint32 gpio_name, uint8 gpio_func)

Parameter：

uint8 gpio_id——gpio number

uint8 gpio_name——gpio mux name

uint8 gpio_func——gpio function

Return：

无

Example：

Use GPIO0 as wifi status LED

```
#define HUMITURE_WIFI_LED_IO_MUX        PERIPHS_IO_MUX_GPIO0_U
#define HUMITURE_WIFI_LED_IO_NUM        0
#define HUMITURE_WIFI_LED_IO_FUNC       FUNC_GPIO0
wifi_status_led_install(HUMITURE_WIFI_LED_IO_NUM,
HUMITURE_WIFI_LED_IO_MUX, HUMITURE_WIFI_LED_IO_FUNC)
```

## 3.2.38. wifi_promiscuous_enable

Function: Enable promiscuous mode for sniffer

Function definition:

Void wifi_promiscuous_enable(uint8 promiscuous)

Parameter：

uint8 promiscuous —— 0, disable promiscuous
1, enable promiscuous

Return:

null

Example: apply for a demo of sniffer function from Espressif

## 3.2.39. wifi_set_promiscuous_rx_cb

Function: register a rx callback function in promiscuous mode, which will be called when data packet is received.

Function definition:

Void wifi_set_promiscuous_rx_cb(wifi_promiscuous_cb_t cb)

Parameter：

wifi_promiscuous_cb_t cb—— callback

Return:

null

### 3.2.40. wifi_get_channel

Function: get channel number, for sniffer

Function definition:

uint8 wifi_get_channel(void)

parameters:

null

Return:

Channel number

### 3.2.41. wifi_set_channel

Function: set channel number, for sniffer

Function definition:

bool wifi_set_channel (uint8 channel)

Parameters：

uint8 channel—— channel number

Return：

True - succeed；

False - fail.

## 3.3. Espconn APIs

Locate in “esp_iot_sdk\include\espconn.h”

General APIs： APIs can be used for both TCP and UDP .

TCP APIs： APIs that are only used for TCP.

UDP APIs： APIs that are only used for UDP.

## 3.3.1. General APIs

### 3.3.1.1. espconn_gethostbyname

Function: DNS

Function definition:

Err_t espconn_gethostbyname(struct espconn *pespconn, const char *hostname, ip_addr_t *addr, dns_found_callback found)

Parameters:

struct espconn *espconn——corresponding connected control block structure

const char *hostname——domain name string pointer

ip_addr_t *addr——ip address

dns_found_callback found——callback

Return:

Err_t——ESPCONN_OK

ESPCONN_INPROGRESS

ESPCONN_ARG

Example as follows. Pls refer to source code of IoT_Demo：

```
ip_addr_t esp_server_ip;
LOCAL void ICACHE_FLASH_ATTR
user_esp_platform_dns_found(const char *name, ip_addr_t *ipaddr, void *arg)
{
    struct espconn *pespconn = (struct espconn *)arg;

    os_printf("user_esp_platform_dns_found %d.%d.%d.%d\n",
              *((uint8 *)&ipaddr->addr), *((uint8 *)&ipaddr->addr + 1),
              *((uint8 *)&ipaddr->addr + 2), *((uint8 *)&ipaddr->addr + 3));
}
Void dns_test(void)
```

```
{
espconn_gethostbyname(pespconn,”iot.espressif.cn”,&esp_server_ip,user_esp_platf
orm_dns_found);
}
```

### 3.3.1.2. espconn_port

Function: get void ports

Function definition:

uint32 espconn_port(void);

Input parameters:

null

Return:

uint32——id of the port you get

### 3.3.1.3. espconn_regist_sentcb

Function: register data sent function which will be called back when data are successfully sent.

Function definition:

Sint8 espconn_regist_sentcb(struct espconn *espconn, espconn_sent_callback sent_cb)

Parameters：

struct espconn *espconn——corresponding connected control block structure

espconn_sent_callback sent_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.1.4. espconn_regist_recvcb

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

Function definition:

Sint8 espconn_regist_recvcb(struct espconn *espconn, espconn_recv_callback recv_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.1.5. espconn_sent_callback

Function: callback after the data are sent

Function definition:

void espconn_sent_callback (void *arg)

Input parameters:

void *arg——call back function parameters

Return:

null

### 3.3.1.6. espconn_recv_callback

Function: callback after data are received

Function definition:

void espconn_recv_callback (void *arg, char *pdata, unsigned short len)

Input parameters:

void *arg——callback function parameters

char *pdata——received data entry parameters

unsigned short len——received data length

Return:

null

#### 3.3.1.7. espconn_sent

Function: send data through wifi

Function definition:

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

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint8 *psent——sent data pointer

uint16 length——sent data length

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2. TCP APIs

#### 3.3.2.1. espconn_accept

Function: listening connection. This function is used when create a TCP server.

Function definition:

sint8 espconn_accept(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

| Not 0 - error, pls refer to espconn.h |
|---|

### 3.3.2.2. espconn_secure_accept

Function: encrypted listening connection. This function is used when create a TCP server which support SSL.

Function definition:

sint8 espconn_secure_accept(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.3. espconn_regist_time

Function: register timeout interval when ESP8266 is TCP server

Function definition:

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

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint32 interval ——timeout interval, unit: second, maximum: 7200 seconds

uint8 type_flag ——0, set all connections; 1, set a single connection

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.4. espconn_get_connection_info

Function: get a connection's info in TCP multi-connection case

Function definition:

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

Input parameters:

struct espconn *espconn——corresponding connected control block structure

remot_info **pcon_info——connect to client info

uint8 typeflags —— 0, regular server;1, ssl server

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.5. espconn_connect

Function: connect to a TCP server, and ESP8266 is the TCP client.

Function definition:

sint8 espconn_connect(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.6. espconn_connect_callback

Function: successful listening(ESP8266 as TCP server) or connection(ESP8266 as TCP client) callback

Function definition:

void espconn_connect_callback (void *arg)

Input parameters:

void *arg——callback function parameters

Return:

null

### 3.3.2.7. espconn_disconnect

Function: disconnect a TCP connection

Function definition:

sint8 espconn_disconnect(struct espconn *espconn);

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.8. espconn_regist_connectcb

Function: register connection function which will be called back under successful TCP connection

Function definition:

Sint8 espconn_regist_connectcb(struct espconn *espconn, espconn_connect_callback connect_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.9. espconn_regist_reconcb

Function: register reconnection function, which will be called back when TCP reconnection starts

Function definition:

sint8 espconn_regist_reconcb(struct espconn *espconn, espconn_connect_callback recon_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.10. espconn_regist_disconcb

Function: register disconnection function which will be called back under successful TCP disconnection

Function definition:

Sint8 espconn_regist_disconcb(struct espconn *espconn, espconn_connect_callback discon_cb)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

espconn_connect_callback connect_cb——registered callback function

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.11. espconn_secure_connect

Function: Secure connect(SSL) to a TCP server, and ESP8266 is the TCP client.

Function definition:

Sint8 espconn_secure_connect (struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.12. espconn_secure_sent

Function: send encrypted data（SSL）

Function definition:

Sint8 espconn_secure_sent (struct espconn *espconn, uint8 *psent, uint16 length)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

uint8 *psent——sent data pointer

uint16 length——sent data length

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

### 3.3.2.13. espconn_secure_disconnect

Function: secure TCP disconnection(SSL)

Function definition:

Sint8 espconn_secure_disconnect(struct espconn *espconn)

Input parameters:

struct espconn *espconn——corresponding connected control block structure

Return:

0 - succeed, #define ESPCONN_OK 0

Not 0 - error, pls refer to espconn.h

## 3.3.3. UDP APIs

### 3.3.3.1. espconn_create

Function： create UDP transmission.

Prototype：

Sin8 espconn_create(struct espconn *espconn)

Parameter：

struct espconn *espconn——corresponding connected control block structure

Return：

0 - succeed， #define ESPCONN_OK 0

Not 0 - Erro， pls refer to espconn.h

### 3.3.3.2. espconn_delete

Function: delete UDP transmission.

Prototype：

Sin8 espconn_delete(struct espconn *espconn)

Parameter：

struct espconn *espconn——corresponding connected control block structure

Return：

| |
|---|
| 0 - succeed，#define ESPCONN_OK 0<br>Not 0 - Erro，pls refer to espconn.h |

## 3.4. json APIs

Locate in : esp_iot_sdk\include\json\jsonparse.h & jsontree.h

### 3.4.1. jsonparse_setup

Function:json initialize parsing

Function definition:

void jsonparse_setup(struct jsonparse_state *state, const char *json, int len)

Input parameters:

struct jsonparse_state *state——json parsing pointer

const char *json——json parsing character string

int len——character string length

Return:

null

### 3.4.2. jsonparse_next

Function: jsonparse next object

Function definition:

int jsonparse_next(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsing result

### 3.4.3. jsonparse_copy_value

Function: copy current parsing character string to a certain buffer

Function definition:

int jsonparse_copy_value(struct jsonparse_state *state, char *str, int size)

Input parameters:

struct jsonparse_state *state——json parsing pointer

char *str——buffer pointer

int size——buffer size

Return:

int——copy result

### 3.4.4. jsonparse_get_value_as_int

Function: parse json to get integer

Function definition:

int jsonparse_get_value_as_int(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsing result

### 3.4.5. jsonparse_get_value_as_long

Function: parse json to get long integer

Function definition:

long jsonparse_get_value_as_long(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

long——parsing result

### 3.4.6. jsonparse_get_len

Function: get parsed json length

Function definition:

int jsonparse_get_value_len(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsed jason length

### 3.4.7. jsonparse_get_value_as_type

Function: parsed json data type

Function definition:

int jsonparse_get_value_as_type(struct jsonparse_state *state)

Input parameters:

struct jsonparse_state *state——json parsing pointer

Return:

int——parsed json data type

### 3.4.8. jsonparse_strcmp_value

Function: compare parsed json and certain character string

Function definition:

int jsonparse_strcmp_value(struct jsonparse_state *state, const char *str)

Input parameters:

struct jsonparse_state *state——json parsing pointer

const char *str——character buffer

Return:

int——comparison result

### 3.4.9. jsontree_set_up

Function: create json data tree

Function definition:

void jsontree_setup(struct jsontree_context *js_ctx,

struct jsontree_value *root,  int (* putchar)(int))

Input parameters:

struct jsontree_context *js_ctx——json tree element pointer

struct jsontree_value *root——root element pointer

int (* putchar)(int)——input function

Return:

null

### 3.4.10. jsontree_reset

Function: reset json tree

Function definition:

void jsontree_reset(struct jsontree_context *js_ctx)

Input parameters:

struct jsontree_context *js_ctx——json data tree pointer

Return:

null

### 3.4.11. jsontree_path_name

Function: get json tree parameters

Function definition:

const char *jsontree_path_name(const struct jsontree_cotext *js_ctx, int depth)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int depth——json tree depth

Return:

char*——parameter pointer

### 3.4.12. jsontree_write_int

Function: write integer to joson tree

Function definition:

void jsontree_write_int(const struct jsontree_context *js_ctx, int value)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int value——integer value

Return:

null

### 3.4.13. jsontree_write_int_array

Function: write integer array to json tree

Function definition:

void jsontree_write_int_array(const struct jsontree_context *js_ctx, const int *text, uint32 length)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int *text——array entry address

uint32 length——array length

Return:

null

### 3.4.14. jsontree_write_string

Function: write string to json tree

Function definition:

void jsontree_write_string(const struct jsontree_context *js_ctx, const char *text)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

const char* text——character string pointer

Return:

null

### 3.4.15. jsontree_print_next

Function: json tree depth

Function definition:

int jsontree_print_next(struct jsontree_context *js_ctx)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

Return:

int——json tree depth

### 3.4.16. jsontree_find_next

Function: find json tree element

Function definition:

struct jsontree_value *jsontree_find_next(struct jsontree_context *js_ctx, int type)

Input parameters:

struct jsontree_context *js_ctx——json tree pointer

int——type

Return:

struct jsontree_value *——json tree element pointer

# 4. Structure definition

## 4.1. Timer

```
typedef void ETSTimerFunc(void *timer_arg);


typedef struct _ETSTIMER_ {
struct _ETSTIMER_       *timer_next;
      uint32_t                    timer_expire;
      uint32_t                    timer_period;
     ETSTimerFunc              *timer_func;
     void                         *timer_arg;
} ETSTimer;
```

## 4.2. Wifi parameters

### 4.2.1. station parameters

```
struct station_config {
uint8 ssid[32];
uint8 password[64];
};
```

### 4.2.2. softap parameters

```
typedef enum _auth_mode {
      AUTH_OPEN                = 0,
      AUTH_WEP,
      AUTH_WPA_PSK,
      AUTH_WPA2_PSK,
```

```
    AUTH_WPA_WPA2_PSK
} AUTH_MODE;
struct softap_config {
    uint8 ssid[32];
    uint8 password[64];
    uint8 channel;
    uint8 authmode;
    uint8 ssid_hidden;
    uint8 max_connection;
};
```

### 4.2.3. scan parameters

```
struct bss_info {
    STAILQ_ENTRY(bss_info)    next;
    u8 bssid[6];
    u8 ssid[32];
    u8 channel;
    s8 rssi;
    u8 authmode;
};
typedef void (* scan_done_cb_t)(void *arg, STATUS status);
```

## 4.3. json related structure

### 4.3.1. json structure

```
struct jsontree_value {
uint8_t type;
};
```

```
struct jsontree_pair {
const char *name;
struct jsontree_value *value;
};

struct jsontree_context {
struct jsontree_value *values[JSONTREE_MAX_DEPTH];
uint16_t index[JSONTREE_MAX_DEPTH];
int (* putchar)(int);
uint8_t depth;
uint8_t path;
int callback_state;
};

struct jsontree_callback {
uint8_t type;
int (* output)(struct jsontree_context *js_ctx);
int (* set)(struct jsontree_context *js_ctx, struct jsonparse_state *parser);
};

struct jsontree_object {
uint8_t type;
uint8_t count;
struct jsontree_pair *pairs;
};

struct jsontree_array {
uint8_t type;
```

```
uint8_t count;
struct jsontree_value **values;
};

struct jsonparse_state {
const char *json;
int pos;
int len;
int depth;
int vstart;
int vlen;
char vtype;
char error;
char stack[JSONPARSE_MAX_DEPTH];
};
```

## 4.3.2. json macro definition

```
#define JSONTREE_OBJECT(name, ...)                                          \
static struct jsontree_pair jsontree_pair_##name[] = {__VA_ARGS__};     \
static struct jsontree_object name = {                                      \
    JSON_TYPE_OBJECT,                                   \
sizeof(jsontree_pair_##name)/sizeof(struct jsontree_pair),              \
    jsontree_pair_##name }

#define JSONTREE_PAIR_ARRAY(value) (struct jsontree_value *)(value)
#define JSONTREE_ARRAY(name, ...)                                           \
static struct jsontree_value* jsontree_value_##name[] = {__VA_ARGS__};     \
static struct jsontree_array name = {                                       \
```

```
    JSON_TYPE_ARRAY,                                    \
sizeof(jsontree_value_##name)/sizeof(struct jsontree_value*),          \
    jsontree_value_##name }
```

## 4.4. espconn parameters

### 4.4.1 callback function

```
/** callback prototype to inform about events for a espconn */
typedef void (* espconn_recv_callback)(void *arg, char *pdata, unsigned short len);
typedef void (* espconn_callback)(void *arg, char *pdata, unsigned short len);
typedef void (* espconn_connect_callback)(void *arg);
```

### 4.4.2 espconn

```
    typedef void* espconn_handle;
    typedef struct _esp_tcp {
        int client_port;
        int server_port;
        char ipaddr[4];
        espconn_connect_callback connect_callback;
        espconn_connect_callback reconnect_callback;
        espconn_connect_callback disconnect_callback;
    } esp_tcp;

    typedef struct _esp_udp {
        int _port;
        char ipaddr[4];
    } esp_udp;

    /** Protocol family and type of the espconn */
```

```
enum espconn_type {
    ESPCONN_INVALID     = 0,
    /* ESPCONN_TCP Group */
    ESPCONN_TCP         = 0x10,
    /* ESPCONN_UDP Group */
    ESPCONN_UDP         = 0x20,
};

/** Current state of the espconn. Non-TCP espconn are always in state ESPCONN_NONE! */
enum espconn_state {
    ESPCONN_NONE,
    ESPCONN_WAIT,
    ESPCONN_LISTEN,
    ESPCONN_CONNECT,
    ESPCONN_WRITE,
    ESPCONN_READ,
    ESPCONN_CLOSE
};

/** A espconn descriptor */
struct espconn {
    /** type of the espconn (TCP, UDP) */
    enum espconn_type type;
    /** current state of the espconn */
    enum espconn_state state;
    union {
        esp_tcp *tcp;
        esp_udp *udp;
```

```
    } proto;
    /** A callback function that is informed about events for this espconn */
    espconn_recv_callback recv_callback;
    espconn_sent_callback sent_callback;
    espconn_handle esp_pcb;
    uint8 *ptrbuf;
    uint16 cntr;
};
```

# 5. Driver

## 5.1. GPIO APIs

Please refer to \user\ user_plug.c。

### 5.1.1. PIN setting macro

- ✓ PIN_PULLUP_DIS(PIN_NAME)

  Disable pin pull up

- ✓ PIN_PULLUP_EN(PIN_NAME)

  Enable pin pull up

- ✓ PIN_PULLDWN_DIS(PIN_NAME)

  Disable pin pull down

- ✓ PIN_PULLDWN_EN(PIN_NAME)

  Enable pin pull down

- ✓ PIN_FUNC_SELECT(PIN_NAME, FUNC)

  Select pin function

  Example：PIN_FUNC_SELECT(PERIPHS_IO_MUX_MTDI_U, FUNC_GPIO12);

  Use MTDI pin as GPIO12。

### 5.1.2. gpio_output_set

Function: set gpio property

Function definition:

void gpio_output_set(uint32 set_mask, uint32 clear_mask, uint32 enable_mask, uint32 disable_mask)

Input parameters:

uint32 set_mask——set high output: 1 means high output; 0 means no status change

uint32 clear_mask——set low output: 1 means low output; 0 means no status

change

uint32 enable_mask——enable outpout bit

uint32 disable_mask——enable input bit

Return:

Null

Example：

- ✓ Set GPIO12 as high-level output：gpio_output_set(BIT12, 0, BIT12, 0);
- ✓ Set GPIO12 as low-level output：gpio_output_set(0, BIT12, BIT12, 0);
- ✓ Set GPIO12 as high-level output，GPIO13 as low-level output，则：

  gpio_output_set(BIT12, BIT13, BIT12|BIT13, 0);
- ✓ Set GPIO12 as input : gpio_output_set(0, 0, 0, BIT12);

### 5.1.3. GPIO input and output macro

- ✓ GPIO_OUTPUT_SET(gpio_no, bit_value)

  Set gpio_no as output bit_value，the same as the output example in 5.1.2
- ✓ GPIO_DIS_OUTPUT(gpio_no)

  Set gpio_no as input, the same as the input example in 5.1.2.
- ✓ GPIO_INPUT_GET(gpio_no)

  Get the level status of gpio_no.

### 5.1.4. GPIO interrupt

- ✓ ETS_GPIO_INTR_ATTACH(func, arg)

  Register GPIO interrupt control function
- ✓ ETS_GPIO_INTR_DISABLE()

  Disable GPIO interrupt
- ✓ ETS_GPIO_INTR_ENABLE()

  Enable GPIO interrupt

### 5.1.5. gpio_pin_intr_state_set

Function: set gpio interrupt state

Function definition:

void gpio_pin_intr_state_set(uint32 i, GPIO_INT_TYPE intr_state)

Input parameters:

uint32 i——GPIO pin ID, if you want to set GPIO14, pls use GPIO_ID_PIN(14);

GPIO_INT_TYPE intr_state——interrupt type

as：

```
typedef enum{
    GPIO_PIN_INTR_DISABLE = 0,
    GPIO_PIN_INTR_POSEDGE= 1,
    GPIO_PIN_INTR_NEGEDGE= 2,
    GPIO_PIN_INTR_ANYEGDE=3,
    GPIO_PIN_INTR_LOLEVEL= 4,
    GPIO_PIN_INTR_HILEVEL = 5
}GPIO_INT_TYPE;
```

Return：

NULL

### 5.1.6. GPIO interrupt handler

Follow below steps to clear interrupt status in GPIO interrupt processing function：

```
uint32 gpio_status;
gpio_status = GPIO_REG_READ(GPIO_STATUS_ADDRESS);
//clear interrupt status
GPIO_REG_WRITE(GPIO_STATUS_W1TC_ADDRESS, gpio_status);
```

## 5.2. UART APIs

By default, UART0 is debug output interface. In the case of dual Uart, UART0 works as data receive and transmit interface, and UART1as debug output interface.

Please make sure all hardware are correctly connected.

### 5.2.1. uart_init

Function: initialize baud rates of the two uarts

Function definition:

void uart_init(UartBautRate uart0_br, UartBautRate uart1_br)

Parameters：

UartBautRate uart0_br——uart0 baud rate

UartBautRate uart1_br——uart1 baud rate

As：

```
typedef enum {
    BIT_RATE_9600     = 9600,
    BIT_RATE_19200    = 19200,
    BIT_RATE_38400    = 38400,
    BIT_RATE_57600    = 57600,
    BIT_RATE_74880    = 74880,
    BIT_RATE_115200 = 115200,
    BIT_RATE_230400 = 230400,
    BIT_RATE_460800 = 460800,
    BIT_RATE_921600 = 921600
} UartBautRate;
```

Return:

NULL

### 5.2.2. uart0_tx_buffer

Function: send user-defined data through UART0

Function definition:

Void uart0_tx_buffer(uint8 *buf, uint16 len)

Parameter:

Uint8 *buf——data to send later

Uint16 len——the length of data to send later

Return:

NULL

### 5.2.3. uart0_rx_intr_handler

Function: UART0 interrupt processing function. Users can add the processing of received data in this function. (Receive buffer size: 0x100; if the received data are more than 0x100, pls handle them yourselves.）

Function definition:

Void uart0_rx_intr_handler(void *para)

Parameter:

Void*para——the pointer pointing to RcvMsgBuff structure

Return:

NULL

## 5.3. i2c master APIs

### 5.3.1. i2c_master_gpio_init

Function: set GPIO in i2c master mode

Function definition:

void i2c_master_gpio_init (void)

Input parameters:

null

Return:

null

### 5.3.2. i2c_master_init

Function: initialize i2c

Function definition:

void i2c_master_init(void)

Input parameters:

null

Return:

null

### 5.3.3. i2c_master_start

Function: set i2c to start data delivery

Function definition:

void i2c_master_start(void)

Input parameters:

null

Return:

null

### 5.3.4. i2c_master_stop

Function: set i2c to stop data delivery

Function definition:

Void i2c_master_stop(void)

Input parameters:

null

Return:

null

### 5.3.5. i2c_master_setAck

Function: set i2c ACK

Function definition:

void i2c_master_setAck (uint8 level)

Input parameters:

uint8 level——ack 0 or 1

Return:

null

### 5.3.6. i2c_master_getAck

Function: get ACK from slave

Function definition:

uint8 i2c_master_getAck (void)

Input parameters:

null

Return:

uint8——0 or 1

### 5.3.7. i2c_master_readByte

Function: read a byte from slave

Function definition:

uint8 i2c_master_readByte (void)

Input parameters:

null

Return:

uint8——the value you read

### 5.3.8. i2c_master_writeByte

Function: write a byte to slave

Function definition:

void i2c_master_writeByte (uint8 wrdata)

Input parameters:

uint8 wrdata——data to write

Return:

null

## 5.4. pwm

4 PWM outputs are supported, more details in pwm.h.

### 5.4.1. pwm_init

Function: initialize pwm function, including gpio, frequency, and duty cycle

Function definition:

void pwm_init(uint16 freq, uint8 *duty)

Input parameters:

uint16 freq——pwm's frequency;

uint8 *duty——duty cycle of each output

Return:

null

### 5.4.2. pwm_start

Function: start PWM. This function need to be called after every pwm config changing.

Prototype：

Void pwm_start (void)

Parameter：

null

Return：

null

### 5.4.3. pwm_set_duty

Function: set duty cycle of an output

Function definition:

void pwm_set_duty(uint8 duty, uint8 channel)

Input parameters:

uint8 duty——duty cycle

uint8 channel——an output

Return:

null

### 5.4.4. pwm_set_freq

Function: set pwm frequency

Function definition:

void pwm_set_freq(uint16 freq)

Input parameters:

uint16 freq——pwm frequency

Return:

null

### 5.4.5. pwm_get_duty

Function: get duty cycle of an output

Function definition:

uint8 pwm_get_duty(uint8 channel)

Input parameters:

uint8 channel——channel of which to get duty cycle

Return:

uint8——duty cycle

### 5.4.6. pwm_get_freq

Function: get pwm frequency

Function definition:

uint16 pwm_get_freq(void)

Input parameters:

null

Return:

uint16——frequency

# 6. Appendix

## A. ESPCONN Programming

Programming guide for ESP8266 running as TCP client and TCP server.

## A.1. TCP Client Mode

### A.1.1. Instructions

ESP8266, working in Station mode, will start client connection when given an IP address.

ESP8266, working in softap mode, will start client connection when the devices which are connected to ESP8266 are given an IP address.

### A.1.2. Steps

1) Initialize espconn parameters according to protocols.
2) Register connect callback function, and register recv callback function.
3) Call espconn_connect function and set up the connection with TCP Server.
4) Call registered connect function after successful connection, which will register the corresponding callback function. Recommend to register disconnect callback function.
5) When using recv callback function or sent callback function to run disconnect, it is recommended to set a time delay to make sure that the all the firmware functions are completed.

# A.2. TCP Server Mode

## A.2.1. Instructions

ESP8266, working in Station mode, will start server listening when given an IP address.

ESP8266, working in softAP mode, will start server listening.

## A.2.2. Steps

(1) Initialize espconn parameters according to protocols.

(2) Register connect callback function. You can omit this step in udp protocol, but register recv callback function.

(3) Call espconn_accept function to listen to the connection with host.

(4) Call registered connect function after successful connection, which will register corresponding callback function.