Cloud Speaker
API Specification

Rev. 2.8.6
2022/5/31

Revision History

Date	Version	Describe	Author	Reviewer
2019.06.03	V1.0	Initial version.		
2019.06.18	V2.0	1. Add key status query interface. 2. Add WIFI interfaces.		
2019.07.02	V2.1	1. Add completion call back of PCM/AMR playing. 2. Add AP mode functions. 3. Add WIFI signal intensity. 4. Add WIFI list. 5. Add battery remain capacity.		
2019.07.19	V2.2	1. Add sdk_http_post interface.		
2019.07.30	V2.3	1. Fix sdk_MQTT_connect, add PING timeout param. 2. Fix LED interfaces. 3. Fix HTTP post interface.		
2019.08.20	V2.4	1. Add sdk_fota_update verification, e.g. MD5/SHA256. 2. Add sdk_get_wifi_MAC. 3. Add sdk_MQTT_publish. 4. Remove WIFI's auto-connection while resetting.		
2019.09.06	V2.5	1. Add sdk_wifi_power_off. 2. Add sdk_wifi_power_on.		
2019.09.07	V2.6	1. Remove sdk_get_battery_percent. 2. Add sdk_wifi_ping. 3. Add sdk_get_wifi_firmware_version		
2019.11.09	V2.7	1. Add user_agent param in http_post. 2. Add sdk_network_gprs_init. 3. Add sdk_network_wifi_init. 4. Add sdk_wifi_deep_sleep_mode. 5. Add sdk_wifi_wakeup.		
2019.12.05	V2.8	Add interfaces: 1. SDK_Lcd_reset 2. SDK_LCD_DisplayOn 3. SDK_LCD_DisplayOff		

		4. SDK_LCD_IdleOn 5. SDK_LCD_IdleOff 6. SDK_LCD_Clear 7. SDK_LCD_Fill 8. SDK_LCD_SetCursor 9. SDK_LCD_SetDispAera 10. SDK_LCD_DrawPoint 11. SDK_LCD_DrawLine 12. SDK_LCD_DrawRectangle 13. SDK_LCD_DrawCircle 14. SDK_LCD_HorizLine 15. SDK_LCD_VertiLine 16. SDK_LCD_SetColor 17. SDK_LCD_DarwBmp 18. SDK_LCD_DisplayQR 19. SDK_LCD_DispChar		
2021/5/3	V2.8.1	1. Review V2.8; 2. Write English Version; 3. Unify doc format; 4. Fix the errors.		
2021/5/5	V2.8.2	1. Unify the section title format. 2. Fix the multi-level numbering problem. 3. Remove LCD display interfaces.		
2021/5/11	V2.8.3	1. Change doc title to: Sound Box App API Spec. 2. Change API names to SDK_xxx.		
2021/5/16	V2.8.4	1. Add: get / set device ID. 2. Add: get / set device token. 3. Add: RAM total / free / stack / heap. 4. Add: ROM total / free.		
2021/9/26	V2.8.5	1. Change system info APIs' status, need more consideration. 2. Change the document title to "Cloud Speaker API Specification".		
2022/5/30	V2.8.6	Add interfaces: 1. sdk_http_get 2. sdk_http_post_auto 3. sdk_bcs_init 4. sdk_bcs_number 5. sdk_bcs_set_rssi		

		6. sdk_bcs_set_speaker 7. sdk_bcs_set_bluetooth 8. sdk_fota_download_http_plain 9. sdk_keypad_scan_handle 10. sdk_reg_key_short_cb 11. sdk_reg_key_long_cb 12. sdk_reg_key_double_cb		

Catalog

1. App Initialization

1.1. sdk_System_Init

This function is used for the system initialization. Please call it at the beginning of the application, such as main function. The function should be called for only one time in the main procedure.

Prototype

int sdk_System_Init (void);

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

1.2. sdk_network_gprs_init

This function is designed for GPRS initialization. A thread is created in its internal procedure to do the task, consuming about 3-10 seconds. The function should be called once, and be called after **sdk_System_Init**. The GPRS initializing status can be queried by calling **sdk_get_gprs_network_status**.

Prototype

int sdk_network_gprs_init (void);

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

1.3. sdk_network_wifi_init

This function is designed for WIFI initialization. A thread is created in its internal procedure to do the task. The function should be called once, and be called after **sdk_System_Init**. The WIFI initializing status can be queried by calling **sdk_get_wifi_init_OK**.

Prototype

int sdk_network_wifi_init (void);

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

2. Network Mode Switch

2.1. sdk_set_network_mode

This function is used to switch the network mode.

Prototype

```
typedef enum
{
    NETWORK_MODE_GPRS = 0,
    NETWORK_MODE_WIFI = 1,
} NETWORK_MODE_E;
int sdk_set_network_mode (NETWORK_MODE_E mode);
```

Parameters

mode

[in] network mode enum value.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

2.2. sdk_get_network_mode

This function returns the network mode (enum value).

Prototype

NETWORK_MODE_E sdk_get_network_mode (void);

Parameters

None.

Return Value

Network mode -- NETWORK_MODE_GPRS / NETWORK_MODE_WIFI.

2.3. sdk_get_gprs_network_status

This function is called for check the GPRS networking status, which means if GPRS is connected.

Prototype

BOOL sdk_get_gprs_network_status (void);

Parameters

None.

Return Value

TRUE indicates that GPRS is connected, FALSE indicates failed.

3. TCP

3.1. sdk_tcp_connect

A client can call this function to establish the TCP or TLS connection with the server.

Prototype

int sdk_tcp_connect (const char *ipaddress, int port, int tls, unsigned long * phConnect);

Parameters

ipaddress

[in] Server IP address.

port

[in] Server port number.

tls

[in] Connection type, 0 indicates the TCP connection, and 1 indicates TLS.

phConnect

[out] A handle is assigned after TCP or TLS connection is established.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

3.2. sdk_tcp_close

This function closes the TCP/TLS connection.

Prototype

int sdk_tcp_close (unsigned long hConnect);

Parameters

hConnect

[in] TCP/TLS connection handle created by sdk_tcp_connect.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

3.3. sdk_tcp_recv

This function receives the data from the TCP/TLS connection.

Prototype

int sdk_tcp_recv (unsigned long hConnect, unsigned char *buffer, int bufferSize, int

*pReadLen, int timeout);

Parameters

hConnect

[in] TCP/TLS connection handle created by sdk_tcp_connect.

buffer

[in] Buffer address for receiving the TCP/TLS data.

bufferSize

[in] Receiving data buffer size.

pReadLen

[out] Pointer to the int variable to save the received data length.

timeout

[in] Time out to receiving the data. The unit is second.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

3.4. sdk_tcp_send

This function sends the data to the server by the TCP/TLS connection.

Prototype

int sdk_tcp_send (unsigned long hConnect, unsigned char *buffer, int bufferSize, int *pWrittenLen, int timeout);

Parameters

hConnect

[in] TCP/TLS connection handle created by sdk_tcp_connect.

buffer

[in] Buffer address for storing the data to be sent.

bufferSize

[in] Length of the data to be sent.

pWrittenLen

[out] Pointer to the int variable to save the written data length.

timeout

[in] Time out to receiving the data. The unit is second.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

4. DNS

4.1. sdk_hosttoip

This function is used to transform the host name to the host IP address.

Prototype

```
int sdk_hosttoip (const char *hostname, char *ipaddress);
```

Parameters

hostname

[in] Buffer address for storing the host name.

ipaddress

[out] Buffer address for storing the host IP address transformed from host name.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5. MQTT

5.1. sdk_MQTT_connect

This function establishes the MQTT connection over the TCP/TLS protocol.

Prototype

```
typedef void (*MQTTClient_connectionLost) (void *context, const char *cause);
```

```
int sdk_MQTT_connect (
    const char *ipaddress,
    int port,
    int tls,
    int timeout,
    const char *clientid,
    const char *username,
    const char *password,
    int keep_alive_interval,
    MQTTClient_connectionLost lostCallback,
    unsigned long * hConnect,
    BOOL will,
    unsigned char *will_topic,
    unsigned char *will_message,
    int ping_timeout,
    int cleansession);
```

Parameters

ipaddress

[in] IP address of MQTT server.

port

[in] Port number of MQTT server.

tls

[in] TCP/TLS mode, 0 indicates MQTT over TCP protocol, 1 indicates MQTT over TLS protocol.

timeout

[in] Time out while establishing the MQTT connection. The unit is second.

clientid

[in] MQTT Client ID.

username

[in] Client uses the user name to be authenticated by MQTT server.

password

[in] Client uses the password to be authenticated by MQTT server.

keep_alive_interval

[in] Idle time between the PINGREQ to keep alive.

lostCallback

[in] Pointer of call back function for processing the connection lost event.

hConnect

[out] Pointer of the unsigned long variable to save the MQTT connection handle.

will

[in] Flag to send the last will. TRUE indicates the last will would be sent, FALSE indicates not to send.

will_ topic

[in] Topic for sending the will message.

will_ message

[in] Will message to be sent while the connection is lost.

ping_timeout

[in] Time out to wait the response of the ping. The unit is millisecond.

cleansession

[in] Flag used to tell the server if the topic should be retained. TRUE indicates not retaining, FALSE indicates retaining.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5.2. sdk_MQTT_close

This function closes the MQTT connection.

Prototype

int sdk_MQTT_close (unsigned long hConnect);

Parameters

hConnect

[in] MQTT connection handle created by sdk_MQTT_connect.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5.3. sdk_MQTT_subcribe

This function is used to subscribe the topic provided by the MQTT server.

Prototype

typedef void (*MSG_ARRIVED) (MessageData *data);

int sdk_MQTT_subcribe (unsigned long hConnect, const char *topic, unsigned long qos, MSG_ARRIVED messageCallback);

Parameters

hConnect

[in] MQTT connection handle created by sdk_MQTT_connect.

topic

[in] Topic name.

qos

[in] Service's QOS flag for the message subscribing.

messageCallback

[in] Pointer of call back function for processing the message arrived.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5.4. sdk_MQTT_unsubcribe

This function is used to unsubscribe the topic provided by the MQTT server.

Prototype

int sdk_MQTT_unsubcribe (unsigned long hConnect, const char *topic);

Parameters

hConnect

[in] MQTT connection handle created by sdk_MQTT_connect.

topic

[in] Topic name.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5.5. sdk_MQTT_yield

This function is used to trying receiving the message from the MQTT server. It is a block procedure. If no message would be received, it will block some time before returning to the caller.

Prototype

int sdk_MQTT_yield (unsigned long hConnect, int seconds);

Parameters

hConnect

[in] MQTT connection handle created by sdk_MQTT_connect.

seconds

[in] Block time, unit is seconds.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

5.6. sdk_MQTT_publish

This function is used for client to tell the server to publish topic messages.

Prototype

int sdk_MQTT_publish (unsigned long hConnect, const char *topic, unsigned long qos, void *payload, int payloadlen);

Parameters

hConnect

[in] MQTT connection handle created by sdk_MQTT_connect.

topic

[in] Topic name.

qos

[in] Service's QOS flag for the message subscribing.

payload

[in] Pointer to the Payload buffer.

payloadlen

[in] Payload data length.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6. Audio play

6.1. sdk_PA_ENABLE

This function is used for enable / disable the audio amplifier.

Prototype

void sdk_PA_ENABLE (BOOL enable);

Parameters

enable

[in] TRUE indicates enable, FALSE indicates disable.

Return Value

None.

6.2. sdk_voiceplay

This function is used for play the audio / voice.

Prototype

int sdk_voiceplay (int tts, const char *message);

Parameters

tts

[in] Flag to control the TTS voice play. 0 indicates NOT using TTS mode, 1 indicates using TTS.

message

[in] Pointer of buffer saving the message to be played.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.3. sdk_audio_playing

This function is used for check whether the audio / voice is playing or not.

Prototype

BOOL sdk_audio_playing (void);

Parameters

Return Value

Equal zero (0) indicates no audio playing, otherwise indicates audio playing.

6.4. sdk_audio_play_nv

This function is used for play the audio / voice from NV. Only amr format audio supported.

Prototype

int sdk_audio_play_nv (int index, int len);

Parameters

index

[in] index of audio stored in NV.

len

[in] length of audio data to play.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.5. sdk_audio_play_data

This function is used for play the audio / voice by async mode..

Prototype

int sdk_audio_play_data (const char *audio_data, unsigned int len, int type);

Parameters

audio_data

[in] the pointer of audio data to play, the pointer of memory should be malloc with LocalAlloc function. The memory of audio_data will free by SDK after playing, so application should not free the memory.

len

[in] the length of audio data to play.

type

[in] the type of audio data to play. The value can be: AUDIO_TYPE_AMR, AUDIO_TYPE_WAV, AUDIO_TYPE_MP3 and AUDIO_TYPE_PCM.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.6. sdk_audio_play_data_sync

This function is used for play the audio / voice by sync mode..

Prototype

int sdk_audio_play_data_sync (const char *audio_data, unsigned int len, int type);

Parameters

audio_data

[in] the pointer of audio data to play. The memory of audio_data should be free by application.

len

[in] the length of audio data to play.

type

[in] the type of audio data to play. The value can be: AUDIO_TYPE_AMR, AUDIO_TYPE_WAV, AUDIO_TYPE_MP3 and AUDIO_TYPE_PCM.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.7. sdk_audio_play_file

This function is used for play the audio / voice file by async mode.

Prototype

int sdk_audio_play_file (const char *filepath);

Parameters

File_path

[in] the path of audio file to play, usually the file stored in flash .

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.8. sdk_audio_play_array

This function is used for play serials of audio / voice stored in NV.

Prototype

int sdk_audio_play_array (int array[20], int array_num);

Parameters

array

[in] the array of audio index stored in NV.

array_num

[in] the real number of the audio.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

6.9. sdk_reg_tts_complete_callback

This function is used to register the call back after the TTS play is complete.

Prototype

void sdk_reg_tts_complete_callback (int tts_complete_callback (void));

Parameters

tts_complete_callback

[in] Pointer of call back function. The function will be called while the TTS voice playing is complete.

Return Value

None.

6.10. sdk_reg_pcm_complete_callback

This function is used to register the call back after the PCM play is complete.

Prototype

sdk_reg_pcm_complete_callback (int pcm_complete_callback (void));

Parameters

pcm_complete_callback

[in] Pointer of call back function. The function will be called while the PCM playing is complete.

Return Value

None.

6.11. sdk_reg_amr_complete_callback

This function is used to register the call back after the AMR play is complete.

Prototype

sdk_reg_amr_complete_callback (int amr_complete_callback (void));

Parameters

amr_complete_callback

[in] Pointer of call back function. The function will be called while the AMR playing is complete.

Return Value

None.

7. OTA

7.1. sdk_fota_download_http

This function downloads the newer version of the equipment firmware files, and saves them into the local storage.

Prototype

int sdk_fota_download_http (const char *url);

Parameters

url

[in] URL address for requesting the equipment firmware files to be upgraded. It supports the HTTP/HTTPS protocols.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

7.2. sdk_fota_update

This function checks the upgrade firmware files. If the files are verified, the function sets the upgrade flag so that the equipment will complete the upgrade process at the next power-on.

Prototype

```
typedef enum {
    ALGORITHMS_TYPE_NONE = 0,     // Not using any verification algorithm.
    ALGORITHMS_TYPE_MD5 = 1,      // Using MD5 to verify the FW files.
    ALGORITHMS_TYPE_SHA256 = 2,   // Using SHA256 to verify the FW files.
} ALGORITHMS_TYPE_E;
int sdk_fota_update (const unsigned char *checkvalue, ALGORITHMS_TYPE_E type);
```

Parameters

checkvalue

[in] Pointer to the buffer storing the verification values. **Reserved, please set it to NULL.**

length

[in] Length of the verification values. **Reserved, please set it to 0.**

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

7.3. sdk_fota_download_http_plain

This function downloads the newer version of the equipment firmware files, and saves them into the local storage.

Prototype

```
int sdk_fota_download_http_plain (const char *url, char *body, unsigned int file_len);
```

Parameters

url

[in] URL address for requesting the equipment firmware files to be upgraded. It supports the HTTP/HTTPS protocols.

body

[in] Pointer to the message that responds to the server.

file_len

[in] Length of the new firmware version.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

8. LED control

8.1. sdk_light_control

This function controls the switch ON/OFF status of the LED lights.

Prototype

```
typedef enum
{
    LED_OFF   = 0,
    LED_RED   = 1<<0,
    LED_GREEN =1 <<1,
} LED_NUM_E;
int sdk_light_control (LED_NUM_E led_num);
```

Parameters

led_num

[in] The flag indicates the LED status and red /green lights. 0 indicates LED switch OFF, 1 indicates red light, 2 indicates green light.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

9. Keypad

9.1. Key press state

```
typedef enum
{
    KEY_PRESS = 0,
    KEY_RELEASE = 1,
} key_state_E;
```

9.2. sdk_get_key_up_state

This function gets the press state of the volume-up key.

Prototype

```
key_state_E sdk_get_key_up_state (void);
```

Parameters

None.

Return Value

KEY_PRESS or KEY_RELEASE.

9.3. sdk_get_key_down_state

This function gets the press state of the volume-down key.

Prototype

```
key_state_E sdk_get_key_down_state (void);
```

Parameters

None.

Return Value

KEY_PRESS or KEY_RELEASE.

9.4. sdk_get_key_replay_state

This function gets the press state of the replay key.

Prototype

key_state_E sdk_get_key_replay_state (void);

Parameters

None.

Return Value

KEY_PRESS or KEY_RELEASE.

9.5. sdk_keypad_scan_handle

This function is used to scan whether keyboard is pressed or not. Usually, application uses a thread to scan and registers the relative callback function to process, register callback function refer to sdk_reg_key_short_cb, sdk_reg_key_long_cb and sdk_reg_key_double_cb.

Prototype

void sdk_keypad_scan_handle(void);

Parameters

None.

Return Value

None.

9.6. sdk_reg_key_short_cb

This function is used to register key short press callback function.

Prototype

```
void sdk_reg_key_short_cb(
    QYY_GPIO_KEY_CB vu,
    QYY_GPIO_KEY_CB vd,
    QYY_GPIO_KEY_CB rep);
```

Parameters

vu

[in] Pointer of the callback function when volume up key short press.

vd

[in] Pointer of the callback function when volume down key short press.

rep

[in] Pointer of the callback function when replay key short press.

Return Value

None

9.7. sdk_reg_key_long_cb

This function is used to register key long press callback function. If the interval between release key and press key is more than 3 seconds, the registered function will be trigger.

Prototype

```
void sdk_reg_key_long_cb(
    QYY_GPIO_KEY_CB lvu,
    QYY_GPIO_KEY_CB lvd,
    QYY_GPIO_KEY_CB lrep,
    QYY_GPIO_KEY_CB lvu_lrep);
```

Parameters

lvu

[in] Pointer of the callback function when volume up key pressed for long time.

lvd

[in] Pointer of the callback function when volume down key pressed for long time.

lrep

[in] Pointer of the callback function when replay key pressed for long time.

lvu_lrep

[in] Pointer of the callback function when both replay key and volume up key pressed for

long time.

Return Value

None

9.8. sdk_reg_double_cb

This function is used to register key double press callback function. If the interval between the first release key and the second press key is less than 300 milliseconds, the registered function will be trigger.

Prototype

```
void sdk_reg_key_double_cb(
    QYY_GPIO_KEY_CB vu_db,
    QYY_GPIO_KEY_CB vd_db,
    QYY_GPIO_KEY_CB rep_db);
```

Parameters

du

[in] Pointer of the callback function when volume up key pressed twice.

db

[in] Pointer of the callback function when volume down key pressed twice.

Rep_db

[in] Pointer of the callback function when replay key pressed twice.

Return Value

None

10. Power control

10.1. sdk_get_battery_percent

(deprecated)

This function gets the remaining percent of the battery energy.

Prototype

uint32 sdk_get_battery_percent (void);

Parameters

None.

Return Value

Remaining percent of the battery energy.

10.2. sdk_power_off

This function turns off the equipment.

Prototype

void sdk_power_off (const char *param);

Parameters

param

[in] **Reserved, please set it to NULL.**

Return Value

None.

10.3. sdk_system_restart

This function restarts the equipment.

Prototype

int sdk_system_restart (const char *param);

Parameters

param

[in] **Reserved, please set it to NULL.**

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

10.4. sdk_get_poweron_type

This function gets the power-on type.

Prototype

```
typedef enum
{
    POWERON_TYPE_USB = 0,
    POWERON_TYPE_POWERKEY = 1,
} POWERON_TYPE_E;
POWERON_TYPE_E sdk_get_poweron_type (void);
```

Parameters

None.

Return Value

Power-on type.

10.5. sdk_reg_powerkey_shortpress_callback

This function is used to register the power key single pressed call back function.

Prototype

```
void sdk_reg_powerkey_shortpress_callback (int powerkey_shortpress_callback (void));
```

Parameters

powerkey_shortpress_callback

[in] Pointer to the call back function for processing the power key pressed event.

Return Value

None.

10.6. sdk_reg_poweroff_callback

This function is used to register the power off call back function. The power off event will be

produced after the power key is long-term pressed. If this function returns 0, the equipment will be powered off, otherwise maintains the power on status.

Prototype

void sdk_reg_poweroff_callback (int poweroff_callback (void));

Parameters

poweroff_callback

[in] Pointer to the call back function for processing the power off event.

Return Value

None.

11.X509 certificate

11.1. sdk_setx509cer

This function sets the root certificate of the server. The certificate is used in the procedure of setting up the TLS connection.

Prototype

int sdk_setx509cer (const char *base64cer, int datalen);

Parameters

base64cer

[in] Pointer of the buffer to store the certificate data.

datalen

[in] Length of the certificate data.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

12. Tag value persistence

12.1. sdk_NV_Read

This function is used to read the item value form the NV storage.

Prototype

int sdk_NV_Read (WORD itemID, WORD cchSize, BYTE *pBuf);

Parameters

itemID

[in] Item ID to be read. Value ranges from 0 to 499. If the item does not exist, the function will return error code.

cchSize

[in] The item length to be read. Unit is bytes.

pBuf

[in] Pointer of the buffer to store the item value data.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

12.2. sdk_NV_Write

This function is used to write the item value into the NV storage.

Prototype

int sdk_NV_Write (WORD itemID, WORD cchSize, BYTE *pBuf);

Parameters

itemID

[in] Item ID to be read. Value ranges from 0 to 499.

cchSize

[in] The item length to be read. Unit is bytes. If the item exists and cchSize is greater than

the item's original size, the function will return error code.

pBuf

[in] Pointer of the buffer to store the item value data.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

12.3. sdk_NV_Delete

This function is used to delete the item in the NV storage.

Prototype

int sdk_NV_Delete (WORD itemID);

Parameters

itemID

[in] Item ID to be read. Value ranges from 0 to 499.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13. WIFI

13.1. sdk_get_wifi_init_OK

This function is used to get the initialization status of the WIFI module. In the procedure of the function of sdk_System_Init (), a thread is started to initialize the WIFI module. The whole process takes 1-3 seconds, and all of the WIFI's associated interfaces cannot be called until its successful completion.

Prototype

BOOL sdk_get_wifi_init_OK (void);

Parameters

None.

Return Value

TRUE indicates success, FALSE indicates failed.

13.2. sdk_start_airkiss

This function enters AirKiss mode, which is a fast access configuration technology of WIFI equipment.

Prototype

```
typedef void(*get_wifi_connect) (unsigned char *ssid, unsigned char *pwd, int state)
GET_WIFI_CONNECT;
int sdk_start_airkiss (GET_WIFI_CONNECT connectCallback, int timeout);
```

Parameters

connectCallback

[in] Call back function of the WIFI access configuration. If the procedure has completed successfully, the parameter 'state' equals 1, otherwise equals 0.

timeout

[in] Time out of the WIFI access configuration. Unit is seconds.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.3. sdk_start_connect

This function connects the equipment to the given Access Point.

Prototype

```
typedef void(*get_wifi_connect) (unsigned char *ssid, unsigned char *pwd, int state)
GET_WIFI_CONNECT;
int sdk_start_connect(unsigned char *ssid,unsigned char *pwd , GET_WIFI_CONNECT
connectCallback, int timeout );
```

Parameters

ssid

[in] SSID of the Access Point.

pwd

[in] Password of the Access Point.

connectCallback

[in] Call back function of the connecting procedure. If the procedure has completed successfully, the parameter 'state' equals 1, otherwise equals 0.

timeout

[in] Time out of the connecting procedure. Unit is seconds.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.4. sdk_get_wifi_singnal

This function gets the signal intensity of the given WIFI Access Point.

Prototype

int sdk_get_wifi_singnal (unsigned char *ssid, unsigned char *pwd, int *strength);

Parameters

ssid

[in] SSID of the Access Point.

pwd

[in] Password of the Access Point.

strength:

[out] Pointer to the variant to store the signal intensity.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.5. sdk_get_wifi_list

This function gets the list of WIFI Access Points.

Prototype

```
typedef enum
{
    ECN_TYPE_OPEN          = 0,
    ECN_TYPE_WEP           = 1,
    ECN_TYPE_WPA_PSK       = 2,
    ECN_TYPE_WPA2_PSK      = 3,
    ECN_TYPE_WPA_WPA2_PSK  = 4,
    ECN_TYPE_WPA2_Enterprise = 5,
} ECN_TYPE_E;
typedef struct
{
    ECN_TYPE_E ecn_type;
    unsigned char ssid [64];
    int rssi;
    unsigned char mac [18];
} AP_INFO_S;
int sdk_get_wifi_list (AP_INFO_S **ap_info, int *items);
```

Parameters

ap_info:

[out] Pointer of the start address of the WIFI list.

items:

[out] Count of the WIFI Access Points.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.6. sdk_start_AP

This function is used to activate the WIFI module to the Access Point modes.

Prototype

typedef void(*get_wifi_connect) (unsigned char *ssid, unsigned char *pwd, int state)

GET_WIFI_CONNECT;

int sdk_start_AP (GET_WIFI_CONNECT connectCallback, unsigned char *ap_ssid, int timeout);

Parameters

connectCallback:

[in] Call back function of the AP configuration. If the procedure has completed successfully, the parameter 'state' equals 1, otherwise equals 0.

ap_ssid:

[in] SSID of the Access Point.

timeout:

[in] Time out of the configuration of the Access Point mode. Unit is seconds.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.7. sdk_get_wifi_MAC

This function gets the MAC address of the WIFI module.

Prototype

int sdk_get_wifi_MAC (char *sta_mac);

Parameters

sta_mac:

[in] Pointer of the buffer to store the MAC data.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.8. sdk_wifi_power_off

Call this function to power off the WIFI module.

Prototype

void sdk_wifi_power_off (void);

Parameters

None.

Return Value

None.

13.9. sdk_wifi_power_on

Call this function to power on the WIFI module.

Prototype

void sdk_wifi_power_on (void);

Parameters

None.

Return Value

None.

13.10. sdk_wifi_deep_sleep_mode

Call this function to switch the WIFI module into the Deep Sleep mode. In this mode, the power consumption of WIFI module is 20 μA. Recommend this way to reduce power consumption of the equipment.

Prototype

int sdk_wifi_deep_sleep_mode (void);

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.11. sdk_wifi_wakeup

This function is used to wake up the WIFI module. After waking up, the WIFI module is reset, and the Access Point need to be reconnected.

Prototype

```
int sdk_wifi_wakeup (void);
```

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.12. sdk_get_wifi_firmware_version

This function is used to get the firmware version of the WIFI module.

Prototype

```
typedef struct {
    char *AT_version [32];
    char *SDK_version [32];
    char *compile_time [32];
    char *Bin_version [32];
} WIFI_FIRMWARE_VERSION_S;
int sdk_get_wifi_firmware_version (WIFI_FIRMWARE_VERSION_S *wifi_firmware_version);
```

Parameters

wifi_firmware_version

[out] Pointer of the buffer to store the firmware version data.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

13.13. sdk_wifi_ping

This function is used to ping the host through the WIFI module.

Prototype

```
int sdk_wifi_ping (const char *hostname, int *time);
```

Parameters

hostname

[in] Host name or IP address.

time

[out] Pointer of the variant to store the time consumption of the PING procedure.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

14. HTTP request

14.1. sdk_http_post

Prototype

```
int sdk_http_post (
    unsigned long hConnect,
    unsigned char* url,
    unsigned char* body,
    unsigned char* authorization,
    unsigned char* content_type,
    unsigned char *user_agent,
    unsigned char* response,
    unsigned int timeout);
```

Parameters

hConnect

[in] TCP/TLS connection handle created by sdk_tcp_connect.

url

[in] Pointer of the buffer storing the URL of the request. For example: https://192.168.31.100/test.html for TLS, http://192.168.31.100/test.html for TCP.

body

[in] Pointer of the buffer storing the post data.

authorization

[in] Pointer of the buffer storing the authorizing data, in the format of “[AuthType] [Space] [base64(user: pwd)]”. For example: "Basic cm9vdDoxMjM0NTY=", user is “root”, and pwd is “123456”.

content_type：

[in] Content type of the request body. If set to NULL, it indicates the default type “application/json”.

user_agent：

[in] User agent description of the client.

response：

[out] Pointer of the buffer to receive the response.

timeout：

[in] Timeout of this POST request. Unit is seconds.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

14.2. sdk_http_get

Prototype

```
int sdk_http_get(
    unsigned char* url,
    unsigned char* body,
```

```
    unsigned char* content_type,
    unsigned char *response,
    unsigned int buff_len,
    unsigned int *recv_len);
```

Parameters

url:
[in] Pointer of the buffer storing the URL of the request.

For example: https://192.168.31.100/test.html for TLS, http://192.168.31.100/test.html for TCP.

body:
[in] Pointer of the buffer storing the post data.

content_type:
[in] Content type of the request body. If set to NULL, it indicates the default type "application/json".

response:
[out] Pointer of the buffer to receive the response.

buff_len:
[in] Length of the buffer storing the post data.
recv_len:
[out] Length of the buffer to receive the response.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

14.3. sdk_http_post_auto

Prototype

```
int sdk_http_post_auto(
    unsigned char* url,
    unsigned char* body,
    unsigned char* content_type,
    unsigned char *response,
```

unsigned int buff_len,

unsigned int *recv_len);

Parameters

url:

[in] Pointer of the buffer storing the URL of the request.

For example: https://192.168.31.100/test.html for TLS, http://192.168.31.100/test.html for TCP.

body:

[in] Pointer of the buffer storing the post data.

content_type:

[in] Content type of the request body. If set to NULL, it indicates the default type "application/json".

response:

[out] Pointer of the buffer to receive the response.

buff_len:

[in] Length of the buffer storing the post data.

recv_len:

[out] Length of the buffer to receive the response.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

15. BCS control

15.1. sdk_bcs_init

The **sdk_bcs_init** function initializes the **BCS** . This initialization function must be called before other functions are called.

Prototype

void sdk_bcs_init(void);

Parameters

None.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

15.2. sdk_bcs_number

The **sdk_bcs_number** function displays the number to the **BCS**.

Prototype

```
int sdk_bcs_number(int number, int bit);
```

Parameters

number:

[in] A number representing the passed-in function. The range of **number** is 0 to 999999.

bit

[in] The number of decimal places representing the number passed into the function. 0 means no decimal point, 1 means one decimal place, 2 means two decimal places.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

15.3. sdk_bcs_set_rssi

The **sdk_bcs_set_rssi** function controls the display of ICONS for mobile network and wifi network status.

Prototype

```
int sdk_bcs_set_rssi(BCS_ICON channel, BOOL bConnect);
```

Parameters

channel:

[in] The value of this variable is selected in the range enum{}; **GPRS_ICON** indicates the mobile network icon, and **WIFI_ICON** indicates the wifi network icon.

```
typedef enum{
    GPRS_ICON = 0,
    WIFI_ICON
```

```
}BCS_ICON;
```

bConnect:

[in] **TRUE** or **FALSE**; True indicates that the network connection is successful. False indicates that the network connection fails.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

15.4. sdk_bcs_set_speaker

The **sdk_bcs_set_speaker** function controls the speaker icon.

Prototype

```
int sdk_bcs_set_speaker(BOOL bPlaying);
```

Parameters

bPlaying:

[in] **TRUE** or **FALSE**; True: turns on the speaker icon. False: turns off the speaker icon.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.

15.5. sdk_bcs_set_bluetooth

The **sdk_bcs_set_bluetooth** function controls bluetooth icon on and off.

Prototype

```
int sdk_bcs_set_bluetooth(BOOL bDisIocn, BOOL bPlaying);
```

Parameters

bDisIcon:

[in] **TRUE** or **FALSE**; True: turns on the bluetooth icon. False: turns off the bluetooth icon.

bPlaying:

[in] **TRUE** or **FALSE**; Ture indicates that the Bluetooth connection is successful. False indicates that the Bluetooth connection fails.

Return Value

Equal zero (0) indicates success, otherwise indicates failed.