

Subscribe to DeepL Pro to edit this document.  
Visit [www.DeepL.com/pro](https://www.deepl.com/pro?cta=edit-document) for more information.

**Graffiti Smart WIFI-SDK Description**

(Realtek platform)

## Versions of records:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| releases | Preparation/revision of notes | proposer | revision date | note |
| 1.0.0 | Creating Documents | Wang Qiuping (1973-), table tennis player, several times world and Olympic winner | 20181227 |  |

1. **Framework Basic Interface (required for accessing Doodle Cloud)**
   * 1. **For system initialization, the first call must be to**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_init(IN CONST CHAR\_T \*fs\_storge\_path). |
| **Functional Description** | Used for initialization of the tuya iot sdk system and must be the first to be called |
| **input parameter** | fs\_storge\_path->filesystem read write storge path  (if os have no fs,then fs\_storge\_path is should be NULL) |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | not have |

**Example of use:**

**OPRATE\_RET ret = tuya\_iot\_init(NULL);**

**If(OPRT\_OK ! = ret ) {**

**return OPRT\_COM\_ERR.**

**}**

* + 1. **Get sdk version information**

|  |  |
| --- | --- |
| **function prototype** | CHAR\_T \*tuya\_iot\_get\_sdk\_info(VOID). |
| **Functional Description** | Get tuya iot sdk version information |
| **input parameter** | VOID |
| **output parameter** | not have |
| **return value** | sdk version information, such as "< TUYA\_IOT\_SDK V1.0.2 FOR rt871x  BUILD AT 2018\_11\_29\_15\_49\_56 D:1 S:0 W:1 V:11.02\_2.2\_3.2 >" |
| **note** | not have |

**Example of use:**

* + 1. **Networking module firmware upgrade processing interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_upgrade\_gw(IN CONST FW\_UG\_S \*fw,\  IN CONST GET\_FILE\_DATA\_CB get\_file\_cb,\  IN CONST UPGRADE\_NOTIFY\_CB upgrd\_nofity\_cb,IN CONST PVOID\_T pri\_data). |
| **Functional Description** | Networking module firmware upgrade processing interface |
| **input parameter** | fw->fireware upgrade.  get\_file\_cb->get file callback;  upgrd\_nofity\_cb->upgrade notify callback;  pri\_data->private data; |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | not have |

* + 1. **Networking module firmware upgrade processing interface (high priority)**

|  |  |
| --- | --- |
| **function prototype** | VOID tuya\_iot\_force\_reg\_gw\_ug\_cb(IN CONST GW\_UG\_INFORM\_CB ug\_cb). |
| **Functional Description** | Networking module firmware upgrade processing interface (higher priority than 1.1.3) |
| **input parameter** | ug\_cb |
| **output parameter** | not have |
| **return value** | not have |
| **note** | force register gw\_ug\_cb,Once set,take it as the main Highest priority |

* + 1. **Device firmware upgrade interface library, including the main control device on the gateway and the upgrade of each terminal sub-device.**

|  |  |
| --- | --- |
| **function prototype** | PERATE\_RET tuya\_iot\_upgrade\_dev(IN CONST CHAR\_T \*devid,\  IN CONST FW\_UG\_S \*fw, IN CONST GET\_FILE\_DATA\_CB get\_file\_cb,\  IN CONST UPGRADE\_NOTIFY\_CB upgrd\_nofity\_cb,\  IN CONST PVOID\_T pri\_data). |
| **Functional Description** | Device firmware upgrade interface library, including the main control device on the gateway and the upgrade of each terminal sub-device. |
| **input parameter** | dev\_id->device id  If the ID is null it means upgrading other modules that are physically together in the GW  (e.g. zigbee, ble, single mcu (DEV\_NM\_NOT\_ATH\_SNGL), etc.)  Not null: indicates upgrading of a module physically separated from GW  fw->fireware upgrade  get\_file\_cb->get file callback  upgrd\_nofity\_cb->upgrade notify callback  pri\_data->private data |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Get information describing a specific function point of a device**

|  |  |
| --- | --- |
| **function prototype** | DP\_DESC\_IF\_S \*tuya\_iot\_get\_dp\_desc(IN CONST CHAR\_T \*dev\_id,IN CONST BYTE\_T dpid); |
| **Functional Description** | Get information describing a specific function point of a device |
| **input parameter** | dev\_id dpid |
| **output parameter** | not have |
| **return value** | (DP\_DESC\_IF\_S \*) fail(NULL) |
| **note** |  |

* + 1. **Getting cached data for a specific function point on a device**

|  |  |
| --- | --- |
| **function prototype** | DP\_PROP\_VALUE\_U \*tuya\_iot\_get\_dp\_prop\_value(IN CONST CHAR\_T \*dev\_id, IN CONST BYTE\_T dpid); |
| **Functional Description** | Getting cached data for a specific function point on a device |
| **input parameter** | not have |
| **output parameter** | not have |
| **return value** | DP\_PROP\_VALUE\_U \* |
| **note** |  |

* + 1. **Get Device Information Interface**

|  |  |
| --- | --- |
| **function prototype** | DP\_PROP\_VALUE\_U \*tuya\_iot\_get\_dp\_prop\_value(IN CONST CHAR\_T \*dev\_id, IN CONST BYTE\_T dpid); |
| **Functional Description** | Getting device information |
| **input parameter** | dev\_id->device id,Note:if(dev\_id == NULL) then means the gateway device |
| **output parameter** | not have |
| **return value** | (DEV\_DESC\_IF\_S \*) fail(NULL) |
| **note** |  |

* + 1. **Gateway Binding Sub-Device Interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_gw\_bind\_dev(IN CONST GW\_PERMIT\_DEV\_TP\_T tp,\  IN CONST USER\_DEV\_DTL\_DEF\_T uddd,IN CONST CHAR\_T \*id,\  IN CONST CHAR\_T \*pk,IN CONST CHAR\_T \*ver). |
| **Functional Description** | Gateway Bind Subdevice interface, called when the Graffiti app enables the gateway to add subdevices to bind the discovered subdevices. |
| **input parameter** | tp uddd id pk ver |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Gateway Unbundled Sub-Device Interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_gw\_unbind\_dev(IN CONST CHAR\_T \*id); |
| **Functional Description** | Gateway Unbundled Sub-Device Interface |
| **input parameter** | id |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Gateway updates sub-device online/offline status**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_dev\_online\_stat\_update(IN CONST CHAR\_T \*dev\_id,IN CONST BOOL\_T online); |
| **Functional Description** | Gateway updates sub-device online/offline status |
| **input parameter** | dev\_id online |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device structured data asynchronous reporting interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET dev\_report\_dp\_json\_async(IN CONST CHAR\_T \*dev\_id,IN CONST TY\_OBJ\_DP\_S \*dp\_data,IN CONST UINT\_T cnt); |
| **Functional Description** | Sub-device structured data asynchronous reporting interface, the background to ensure the reliability of data uploads |
| **input parameter** | dev\_id->device id,Note: if(dev\_id == NULL) then means  the gateway device report data  dp\_data->dp data,note "time\_stamp" is useless.The data  reporting time is based on the actual arrival of the server  cnt->count |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Interface for synchronized reporting of sub-device passthrough data**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET dev\_report\_dp\_raw\_sync(IN CONST CHAR\_T \*dev\_id,\  IN CONST BYTE\_T dpid,IN CONST BYTE\_T \*data,IN CONST UINT\_T len,\  IN CONST UINT\_T timeout). |
| **Functional Description** | Synchronized reporting interface for sub-device passthrough data, with reliability of data reporting guaranteed by the caller |
| **input parameter** | dev\_id->device id,Note: if(dev\_id == NULL) then means  the gateway device report data  data->raw data  len->raw data len  timeout |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device structured data synchronization reporting interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET dev\_report\_dp\_raw\_sync(IN CONST CHAR\_T \*dev\_id,\  IN CONST BYTE\_T dpid,IN CONST BYTE\_T \*data,IN CONST UINT\_T len,\  IN CONST UINT\_T timeout). |
| **Functional Description** | Sub-device structured data synchronization reporting interface, by the caller to ensure the reliability of data reporting, usually used for statistical data reporting |
| **input parameter** | dev\_id->device id,Note: if(dev\_id == NULL) then means  the gateway device report data  data->raw data  len->raw data len  timeout |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Initiate sub-device heartbeat management capability**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_sys\_mag\_hb\_init(IN CONST DEV\_HEARTBEAT\_SEND\_CB hb\_send\_cb); |
| **Functional Description** | Initiate sub-device heartbeat management capability |
| **input parameter** | hb\_send\_cb  if(NULL == hb\_send\_cb)  passive heartbeat pattern  else  active heartbeat pattern |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device heartbeat timeout settings**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_set\_dev\_hb\_timeout(IN CONST CHAR\_T \*dev\_id, IN CONST TIME\_S hb\_timeout\_time); |
| **Functional Description** | Sub-device heartbeat timeout time setting, if the gateway does not receive a sub-device heartbeat for more than the set time again, the gateway will set the sub-device as offline. |
| **input parameter** | dev\_id  hb\_timeout\_time->if(0xffffffff == hb\_timeout\_time)  then device online forever |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device refresh timeout offline time**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_fresh\_dev\_hb(IN CONST CHAR\_T \*dev\_id). |
| **Functional Description** | Sub-device refresh timeout offline time |
| **input parameter** | dev\_id |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device firmware version update**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_gw\_subdevice\_update(IN CONST CHAR\_T \*id,IN CONST CHAR\_T \*ver); |
| **Functional Description** | Sub-device firmware version update |
| **input parameter** | id ver |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Sub-device traversal, through this interface can traverse all the sub-devices under the gateway**

|  |  |
| --- | --- |
| **function prototype** | DEV\_DESC\_IF\_S \*tuya\_iot\_dev\_traversal(INOUT VOID \*\*iterator). |
| **Functional Description** | Sub-device traversal, through this interface can traverse all the sub-devices under the gateway |
| **input parameter** | iterator |
| **output parameter** | not have |
| **return value** | (DEV\_DESC\_IF\_S \*) (NULL means end) |
| **note** |  |

1. **Wifi Configuration Class Interface**

**See "tuya\_iot\_wifi\_api.h" for details.**

* + 1. **Setting up authorization information for configuration class wifi devices**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_set\_wf\_gw\_prod\_info(IN CONST WF\_GW\_PROD\_INFO\_S \*wf\_prod\_info); |
| **Functional Description** | Set the authorization information of the configuration class wifi device, the authorization information should be obtained through the Graffiti background, otherwise the device can not be used normally |
| **input parameter** | wf\_prod\_info |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **wifi mcu device initialization interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_wf\_mcu\_dev\_init(IN CONST GW\_WF\_CFG\_MTHD\_SEL cfg,\  IN CONST GW\_WF\_START\_MODE start\_mode,\  IN CONST TY\_IOT\_CBS\_S \*cbs,IN CONST CHAR\_T \*product\_key,\  IN CONST CHAR\_T \*wf\_sw\_ver,IN CONST CHAR\_T \*mcu\_sw\_ver). |
| **Functional Description** | wifi mcu device initialization interface |
| **input parameter** | cfg  start\_mode  cbs->tuya wifi sdk user callbacks  product\_key->product key/proudct id,get from tuya open platform  wf\_sw\_ver->wifi module software version format:xx.xx.xx (0<=x<=9)  mcu\_sw\_ver->mcu software version format:xx.xx.xx (0<=x<=9) |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | typedef BYTE\_T GW\_WF\_START\_MODE.  #define WF\_START\_AP\_ONLY 0 // only have ap-cfg mode  #define WF\_START\_SMART\_ONLY 1 // only have smart-cfg mode  #define WF\_START\_AP\_FIRST 2 // default is ap-cfg mode  #define WF\_START\_SMART\_FIRST 3 // default is smart-cfg mode |

* + 1. **wifi soc device initialization interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_wf\_soc\_dev\_init(IN CONST GW\_WF\_CFG\_MTHD\_SEL cfg,\  IN CONST GW\_WF\_START\_MODE start\_mode, \  IN CONST TY\_IOT\_CBS\_S \*cbs,IN CONST CHAR\_T \*product\_key,\  IN CONST CHAR\_T \*wf\_sw\_ver). |
| **Functional Description** | wifi soc device initialization interface |
| **input parameter** | Cfg  start\_mode  cbs->tuya wifi sdk user callbacks,note cbs->dev\_ug\_cb is useless  product\_key->product key/proudct id,get from tuya open platform  wf\_sw\_ver->wifi module software version format:xx.xx.xx (0<=x<=9) |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | typedef BYTE\_T GW\_WF\_START\_MODE.  #define WF\_START\_AP\_ONLY 0 // only have ap-cfg mode  #define WF\_START\_SMART\_ONLY 1 // only have smart-cfg mode  #define WF\_START\_AP\_FIRST 2 // default is ap-cfg mode  #define WF\_START\_SMART\_FIRST 3 // default is smart-cfg mode |

* + 1. **wifi gateway initialization interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_wf\_gw\_init(IN CONST GW\_WF\_CFG\_MTHD\_SEL cfg,\  IN CONST TY\_IOT\_CBS\_S \*cbs,IN CONST TY\_IOT\_GW\_CBS\_S \*gw\_cbs,\  IN CONST CHAR\_T \*product\_key,IN CONST CHAR\_T \*wf\_sw\_ver,\  IN CONST GW\_ATTACH\_ATTR\_T \*attr,IN CONST UINT\_T attr\_num). |
| **Functional Description** | wifi gateway initialization interface |
| **input parameter** | cfg  cbs->tuya wifi sdk user callbacks  gw\_cbs->tuya gw user callbacks  product\_key->product key/proudct id,get from tuya open platform  wf\_sw\_ver->wifi module software version format:xx.xx.xx (0<=x<=9)  attr->gw attr |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **wifi gateway plus device initialization interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_wf\_gw\_dev\_init(IN CONST GW\_WF\_CFG\_MTHD\_SEL cfg,\  IN CONST TY\_IOT\_CBS\_S \*cbs,IN CONST TY\_IOT\_GW\_CBS\_S \*gw\_cbs,\  IN CONST CHAR\_T \*product\_key,IN CONST CHAR\_T \*wf\_sw\_ver,\  IN CONST GW\_ATTACH\_ATTR\_T \*attr,IN CONST UINT\_T attr\_num). |
| **Functional Description** | wifi gateway plus device initialization interface |
| **input parameter** | cfg  cbs->tuya wifi sdk user callbacks  gw\_cbs->tuya gw user callbacks  product\_key->product key/proudct id,get from tuya open platform  wf\_sw\_ver->wifi module software version format:xx.xx.xx (0<=x<=9)  attr->gw attr  attr\_num |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Get wifi status interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_reg\_get\_wf\_nw\_stat\_cb(IN CONST GET\_WF\_NW\_STAT\_CB wf\_nw\_stat\_cb); |
| **Functional Description** | Get wifi status interface |
| **input parameter** | wf\_nw\_stat\_cb |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **wifi device reset interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_wf\_gw\_unactive(VOID). |
| **Functional Description** | wifi device reset interface |
| **input parameter** | VOID |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **Setting the default ssid and passwd for wifi device ap pairing**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_set\_user\_def\_ap\_if(IN CONST CHAR\_T \*ssid,IN CONST CHAR\_T \*passwd); |
| **Functional Description** | Set the default ssid and passwd when the wifi device ap is paired to the network. if not set, the default hotspot name when the device is in ap state is: SmartLife-xxxx (xxxx is the last four digits of the mac address). |
| **input parameter** | ssid passwd |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | need call before tuya\_iot\_wf\_xxx\_init |

* + 1. **Other distribution mode call interfaces (not ap and smart config)**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_iot\_gw\_wf\_user\_cfg(IN CONST CHAR\_T \*ssid,IN CONST CHAR\_T \*passwd,IN CONST CHAR\_T \*token); |
| **Functional Description** | This interface is called to process when mating via other modes (non-ap non-smartconfig), such as QR code mating for cameras, and sonic mating. |
| **input parameter** | ssid passwd token |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | use to TY\_APP\_CFG\_WF mode |

1. **Introduction to Application Layer Entry Functions**
   * 1. **tuya\_main.c**

|  |  |
| --- | --- |
| **function prototype** | tuya\_main.c : user\_main() : |
| **Functional Description** | The entry function of tuya\_iot,tuya\_iot initialization of each module (timer, event, flash related), factory test process, gateway upgrade registration, and finally start the user code entry function device\_init(). |
| **input parameter** | not have |
| **output parameter** | not have |
| **return value** | not have |
| **note** |  |

* + 1. **tuya\_device.c**

|  |  |
| --- | --- |
| **function prototype** | tuya\_device.c : device\_init(): |
| **Functional Description** | Start the device initialization process such as soc device tuya\_iot\_wf\_soc\_dev\_init, start the wifi networking status monitoring timer and project related functions. |
| **input parameter** | not have |
| **output parameter** | not have |
| **return value** | not have |
| **note** |  |

1. **flash read-write interface**

flash read/write interface header file tuya\_ws\_db.h

* + 1. **flash read interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET wd\_common\_read(IN CONST CHAR\_T \*key, OUT BYTE\_T \*\*value, OUT UINT\_T \*p\_len); |
| **Functional Description** | Read flash related content |
| **input parameter** | key |
| **output parameter** | value p\_len |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **flash write interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET wd\_common\_write(IN CONST CHAR\_T \*key, IN CONST BYTE\_T \*value, IN CONST UINT\_T len); |
| **Functional Description** | Write operations to flash |
| **input parameter** | key value len |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

1. **Configuration information read/write interface for oem firmware**
   * 1. **oem firmware configuration information read interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET ws\_db\_user\_param\_read(OUT BYTE\_T \*\*buf,OUT UINT\_T \*len); |
| **Functional Description** | Configuration information read interface for oem firmware |
| **input parameter** | buf len |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

* + 1. **oem firmware configuration information write interface**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET ws\_db\_user\_param\_write(IN CONST BYTE\_T \*data,IN CONST UINT\_T len); |
| **Functional Description** | oem firmware configuration information write interface |
| **input parameter** | data len |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |

1. **Log Printing Related**
   * 1. **Description of fields related to log level**

|  |  |
| --- | --- |
| **Related Description.** | #define LOG\_LEVEL\_ERR 0 // Error messages, messages that should not occur for normal program operation  #define LOG\_LEVEL\_WARN 1 // Warning message.  #define LOG\_LEVEL\_NOTICE 2 // Information to be aware of  #define LOG\_LEVEL\_INFO 3 // Notification messages  #define LOG\_LEVEL\_DEBUG 4 // Program run debugging information, removed in RELEASE version  #define LOG\_LEVEL\_TRACE 5 // Program run path information, removed in RELEASE version |

* + 1. **Interface for setting the print level**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET SetLogManageAttr(IN CONST LOG\_LEVEL curLogLevel); |
| **Functional Description** | Setting the print level |
| **input parameter** | curLogLevel  // Log output level  typedef INT\_T LOG\_LEVEL.  #define LOG\_LEVEL\_ERR 0 // Error messages, messages that should not occur for normal program operation  #define LOG\_LEVEL\_WARN 1 // Warning message.  #define LOG\_LEVEL\_NOTICE 2 // Information to be aware of  #define LOG\_LEVEL\_INFO 3 // Notification messages  #define LOG\_LEVEL\_DEBUG 4 // Program run debugging information, removed in RELEASE version  #define LOG\_LEVEL\_TRACE 5 // Program run path information, removed in RELEASE version |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** |  |
| **Examples of use** | SetLogManageAttr(LOG\_LEVEL\_INFO), then log messages of levels below LOG\_LEVEL\_INFO (LOG\_LEVEL\_DEBUG and LOG\_LEVEL\_TRACE) are not printed. |

1. **gpio test**

The gpio test simply calls gpio\_test\_cb from the gpio\_test interface in tuya\_device.c

* + 1. **gpio test interface**

|  |  |
| --- | --- |
| **function prototype** | BOOL\_T gpio\_test\_cb(BOARD\_TYPE type); |
| **Functional Description** | gpio test interface |
| **input parameter** | not have |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | The gpio test simply calls gpio\_test\_cb from the gpio\_test interface in tuya\_device.c |
| **Examples of use** | BOOL gpio\_test(VOID) {  return gpio\_test\_cb(RTL\_BOARD\_WR3) || gpio\_test\_cb(RTL\_BOARD\_WR1) || \  gpio\_test\_cb(RTL\_BOARD\_WR2)||gpio\_test\_cb(RTL\_BOARD\_WR4)|| \  gpio\_test\_cb(RTL\_BOARD\_WR7).  } |

1. **keypad interface**
   * 1. **Key Initialization**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET key\_init(IN CONST KEY\_USER\_DEF\_S \*p\_tbl,IN CONST INT\_T cnt,\  IN CONST INT\_T timer\_space) |
| **Functional Description** | Key Initialization Interface |
| **input parameter** | p\_tbl cnt  timer\_space->if timer (space == 0) then use default value(20ms) |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | Called before the key is registered |
| **Examples of use** | OPERATE\_RET op\_ret = OPRT\_OK;  op\_ret = key\_init(NULL,0,KEY\_TIMER\_MS);  if(op\_ret ! = OPRT\_OK) {  PR\_ERR("key\_init err:%d",op\_ret);  return op\_ret;  } |

* + 1. **Registering a keystroke callback function**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET reg\_proc\_key(IN CONST KEY\_USER\_DEF\_S \*key\_ud); |
| **Functional Description** | Registering a keystroke callback function |
| **input parameter** | KEY\_USER\_DEF\_S \*key\_ud  typedef struct { // user define  TY\_GPIO\_PORT\_E port; //io port  BOOL\_T low\_level\_detect; // Low level detect enable true: low active false: high active  KEY\_LONG\_PRESS\_TP\_E lp\_tp; //Trigger Type Setting  USHORT\_T long\_key\_time; // long press time ms (lp\_tp == LP\_ONCE\_TRIG then valid and must >= 1000ms)  USHORT\_T seq\_key\_detect\_time; //key\_detect\_time ms 0:disable default:400ms  KEY\_CALLBACK call\_back; //key action callback function  }KEY\_USER\_DEF\_S. |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | Register the key callback function before using the keys |
| **Examples of use** | OPERATE\_RET op\_ret = OPRT\_OK;  op\_ret = reg\_proc\_key(&hw\_table->rst\_button);//initialize reset button  if( op\_ret ! = OPRT\_OK){  PR\_ERR("reg\_proc\_key err:%d",op\_ret);  return;  } |

Note: In low power mode, only four gpio's are available as keys: GPIO5, GPIO18, GPIO22, GPIO23.

1. **LED interface**
   * 1. **Creating LED control handles**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_create\_led\_handle(IN CONST TY\_GPIO\_PORT\_E port,IN CONST BOOL\_T high,OUT LED\_HANDLE \*handle); |
| **Functional Description** | Creating LED control handles |
| **input parameter** | TY\_GPIO\_PORT\_E port // io port  IN CONST BOOL\_T high //high enable true: high enable  OUT LED\_HANDLE \*handle //LED handle  //LED creation success setting optional  typedef PVOID\_T LED\_HANDLE; // led handle  typedef enum {  OL\_LOW = 0, // output level low  OL\_HIGH, // output level high  OL\_FLASH\_LOW, // when led flash,the level output low first.  // when led flash end,the the level output low.  OL\_FLASH\_HIGH, // when led flash,the level output high first.  // when led flash end,the the level output high.  }LED\_LT\_E. |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | Create handles for relays and indicators before using them. |
| **Examples of use** | OPERATE\_RET op\_ret = OPRT\_OK;  UCHAR\_T i.  if(hw\_table->wifi\_stat\_led.type == IO\_DRIVE\_LEVEL\_HIGH) { //initialize light up led  op\_ret = tuya\_create\_led\_handle(hw\_table->wifi\_stat\_led.pin,FALSE,\  &hw\_table->wifi\_stat\_led\_handle).  }else if(hw\_table->wifi\_stat\_led.type == IO\_DRIVE\_LEVEL\_LOW) {  op\_ret = tuya\_create\_led\_handle(hw\_table->wifi\_stat\_led.pin,\  TRUE,&hw\_table->wifi\_stat\_led\_handle);  } |

Note: Generally indicator lamps and relays call this interface for registration.

* + 1. **LED control**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET tuya\_create\_led\_handle(IN CONST TY\_GPIO\_PORT\_E port,IN CONST BOOL\_T high,OUT LED\_HANDLE \*handle); |
| **Functional Description** | Creating LED control handles |
| **input parameter** | TY\_GPIO\_PORT\_E port // io port  IN CONST BOOL\_T high //high enable true: high enable  OUT LED\_HANDLE \*handle //LED handle  //LED creation success setting optional  typedef PVOID\_T LED\_HANDLE; // led handle  typedef enum {  OL\_LOW = 0, // output level low  OL\_HIGH, // output level high  OL\_FLASH\_LOW, // when led flash,the level output low first.  // when led flash end,the the level output low.  OL\_FLASH\_HIGH, // when led flash,the level output high first.  // when led flash end,the the level output high.  }LED\_LT\_E. |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | Create handles for relays and indicators before using them. |
| **Examples of use** | OPERATE\_RET op\_ret = OPRT\_OK;  UCHAR\_T i.  if(hw\_table->wifi\_stat\_led.type == IO\_DRIVE\_LEVEL\_HIGH) { //initialize light up led  op\_ret = tuya\_create\_led\_handle(hw\_table->wifi\_stat\_led.pin,FALSE,\  &hw\_table->wifi\_stat\_led\_handle).  }else if(hw\_table->wifi\_stat\_led.type == IO\_DRIVE\_LEVEL\_LOW) {  op\_ret = tuya\_create\_led\_handle(hw\_table->wifi\_stat\_led.pin,\  TRUE,&hw\_table->wifi\_stat\_led\_handle);  } |

1. **UART interface**
   * 1. **Serial port initialization**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET ty\_uart\_init(IN CONST TY\_UART\_PORT\_E port,\  IN CONST TY\_UART\_BAUD\_E badu, IN CONST TY\_DATA\_BIT\_E bits,\  IN CONST TY\_PARITY\_E parity,IN CONST TY\_STOPBITS\_E stop,\  IN CONST UINT\_T bufsz,IN CONST BOOL\_T unblock) |
| **Functional Description** | Serial port initialization settings |
| **input parameter** | IN CONST TY\_UART\_PORT\_E port \\\ Set serial port number  IN CONST TY\_UART\_BAUD\_E badu \\\ Set communication baud rate  IN CONST TY\_DATA\_BIT\_E bits \\ Data Bit  IN CONST TY\_PARITY\_E parity \\ Checksum  IN CONST TY\_STOPBITS\_E stop \\\ Stop Bits  IN CONST UINT\_T bufsz \\\ Buffer Size  IN CONST BOOL\_T unblock \\\ read uart whether to use non-blocking method 1: unblock, 0: block |
| **output parameter** | not have |
| **return value** | OPERATE\_RET |
| **note** | You need to set up the call before using the serial port again |
| **Examples of use** | OPERATE\_RET op\_ret = OPRT\_OK;  op\_ret = ty\_uart\_init(USER\_UART, baud, TYWL\_8B, TYP\_NONE, TYS\_STOPBIT1, DATA\_BUF\_SZ,TRUE); |

1. **Low Power Firmware**

**11.1.1 Setting the Low Power Mode**

|  |  |
| --- | --- |
| **function prototype** | VOID tuya\_set\_lp\_mode(BOOL\_T lp\_enable) |
| **Functional Description** | Setting the Low Power Mode |
| **input parameter** | BOOL\_T lp\_enable \\\\  [1-low power](\\\\1-enable) mode; 0-normal mode |
| **output parameter** | not have |
| **return value** | VOID |
| **note** | Called before device\_init, recommended to be called in pre\_device\_init. |
| **Examples of use** |  |

**11.1.2 Turning on low power consumption**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET wf\_lowpower\_enable(VOID) |
| **Functional Description** | Turn on low power |
| **input parameter** |  |
| **output parameter** | not have |
| **return value** | VOID |
| **note** | In low-power mode, some functions need to be turned off for low-power to work properly |
| **Examples of use** |  |

**11.1.2 Turning off low power consumption**

|  |  |
| --- | --- |
| **function prototype** | OPERATE\_RET wf\_lowpower\_disable(VOID) |
| **Functional Description** | Turn off low power consumption |
| **input parameter** |  |
| **output parameter** | not have |
| **return value** | VOID |
| **note** | In low-power mode, some functions need to be turned off for low-power to work properly |
| **Examples of use** |  |