5.x SDK Development Guide

Version: 20240612

Online Version

Contents

This topic describes how to run the IPC SDK demo and develop a fully functional IPC product by using the IPC SDK. Each section describes how to integrate with a specific feature. You can implement functionality as needed.

1 Terms

Term	Description
Pairing	A device can connect to the cloud through pairing with the mobile app. Pairing enables communications between the device, mobile app, and cloud.
Product ID	Product ID (PID) is automatically generated for each product created on the Tuya Developer Platform to record product information in the cloud. The PID is associated with the data points defined for a product. IPC devices of the same product category share the same PID.
UUID	A universally unique identifier (UUID) is a unique number generated by various algorithms to identify hardware or software.
AuthKey	AuthKey is used to authorize devices registered on the Tuya Developer Platform to access cloud services.
P2P ID	P2P ID stands for point-to-point ID. For SDK v3.0.0 and later, the P2P ID is automatically assigned by the cloud without manual assignment.
Token	A token is an identification code generated by the cloud when users scan the QR code to pair a device. A token is valid for 10 minutes.
Data point (DP)	A data point (DP) represents a feature defined for a product. The DP serves the communication between a device and the cloud.

Term	Description
Chromecast	Google Chromecast is a streaming media adapter and can stream content from multiple platforms to your digital TV.
Echo Show	Amazon Echo Show is a smart speaker enabled with touchscreen and supports live streaming services.
IFTTT	IFTTT stands for If This Then That. It is a service that lets users connect to cloud services and smart devices to create automated actions.
OTA	OTA stands for over-the-air and refers to any type of wireless transmission. It is most commonly used to describe the wireless delivery of new software, firmware, or other data to smart devices.

2 SDK directory

The following directory shows what is included in the SDK.

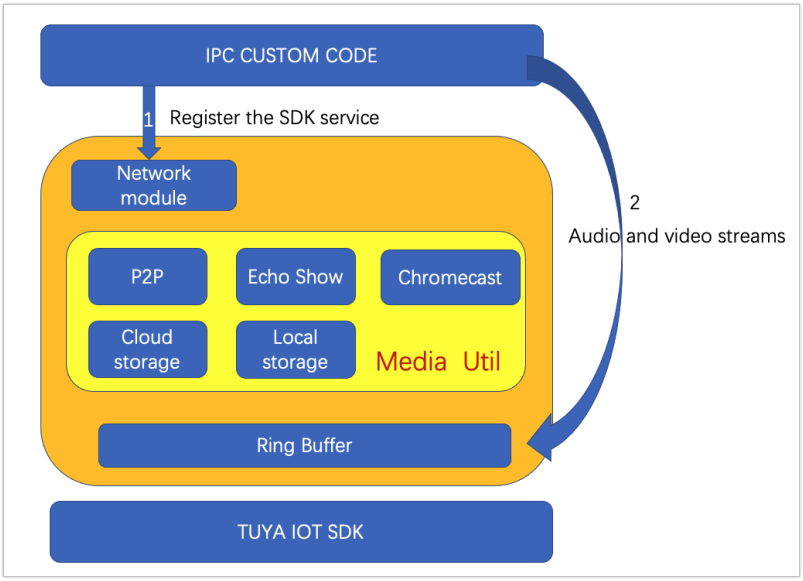
```
.
    tuya_ipc_sdk
        apps        // Application files. This folder is empty currently.
        platforms                       // This folder is empty currently.
        sdk                                             // The SDK files.
            include
                cJSON.h                                 // cJSON files.
                tuya_cloud_base_defs.h     // The basic definitions of the Tuya Developer Platform.
                tuya_cloud_com_defs.h            // The common definitions of the Tuya Developer Platform.
                tuya_cloud_error_code.h             // The definitions of error code.
                tuya_cloud_types.h         // The definitions of types.
                tuya_cloud_wifi_defs.h         // The definitions of Wi-Fi utils.
                codec
                    tuya_g711_utils.h          // G.711 codec.
                tuya_ipc_api.h               // APIs.
                tuya_ipc_cloud_storage.h            // Cloud storage.
                tuya_ipc_img_defs.h        // The definitions of IPC image gallery.
                tuya_ipc_img_proc.h        // IPC image APIs.
                tuya_ipc_media.h            // IPC media.
                tuya_ipc_p2p.h             // IPC P2P.
                tuya_ipc_ptz.h              // IPC PTZ.
                tuya_ipc_qrcode_proc.h              // QR code.
                tuya_ipc_skill.h           // IPC capabilities.
                tuya_ipc_stream_storage.h           // IPC local storage.
                tuya_ipc_video_msg.h       // IPC video messages.
                tuya_ipc_video_proc.h          // IPC video detection.
                tuya_ipc_webrtc.h              // IPC WebRTC streaming.
                tuya_ring_buffer.h         // Ring buffers.
                uni_network.h                       // Networking.
                ty_wifi_mgnt.h              // Wi-Fi connection APIs.
                ...
            libs
                libtuya_iot.a
                    libtuya_iot.a.stripped
        bulid_app.sh                        // Refer to the demo.
        CHANGELOG.md                                // Change logs.
        README.md                       // Description file.
```

3 Implementation process

To develop with the IPC SDK, you need to first register the SDK service. The network module establishes a connection to the cloud. Then, you can feed the media in the required format into the buffer that is allocated by the SDK. At this point, the registration is completed, and the SDK will take care of the subsequent services.

Users can control devices with a mobile app. With the raw media data, the SDK works with the mobile app to implement data storage, P2P networking, and third-party integration such as with Amazon Echo Show and Google Chromecast.

4 SDK architecture



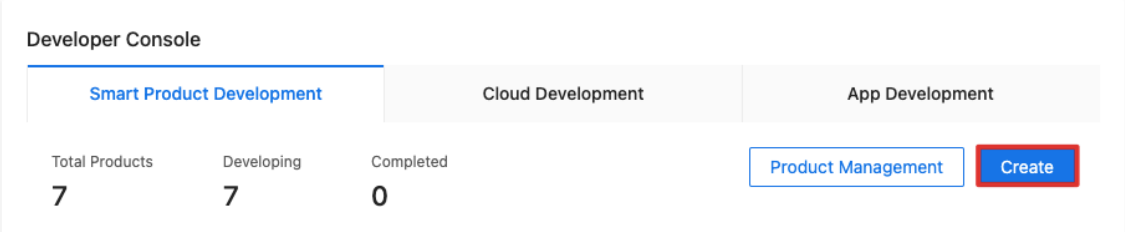
5 Get started

Try out the IPC SDK with the following procedure.

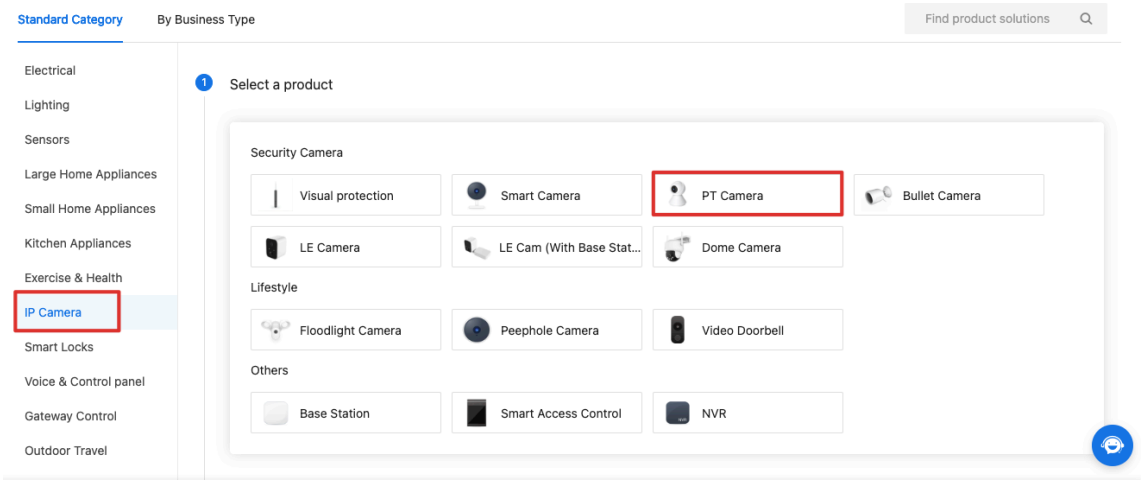
1. Log in to the Tuya Developer Platform and create a product.
2. Get Tuya’ s IPC SDK.
3. Download `tuya_ipc_demo`.
4. Download the Smart Life app from Apple Store or mobile app stores.
5. Install Ubuntu Linux and set up the IDE.
6. Run the demo on Ubuntu and try device functionality with the Smart Life app.

5.1 Step 1: Create product

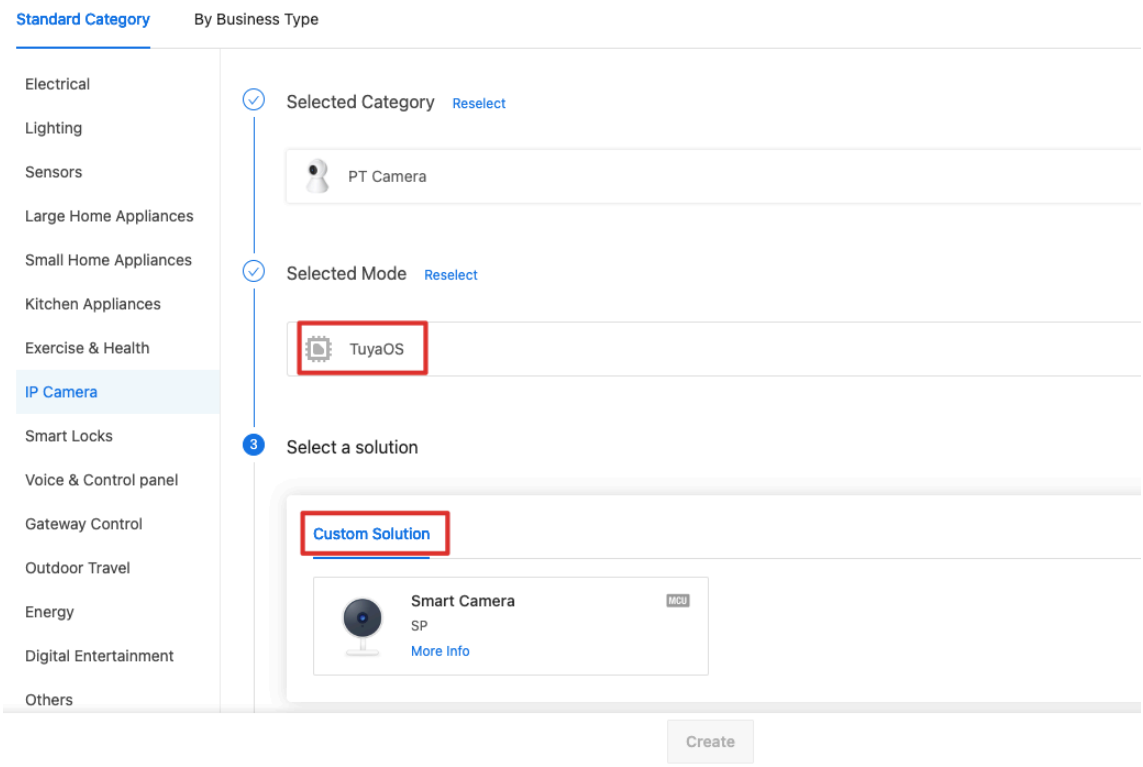
1. Log in to the Tuya Developer Platform.
2. Click **Create**.



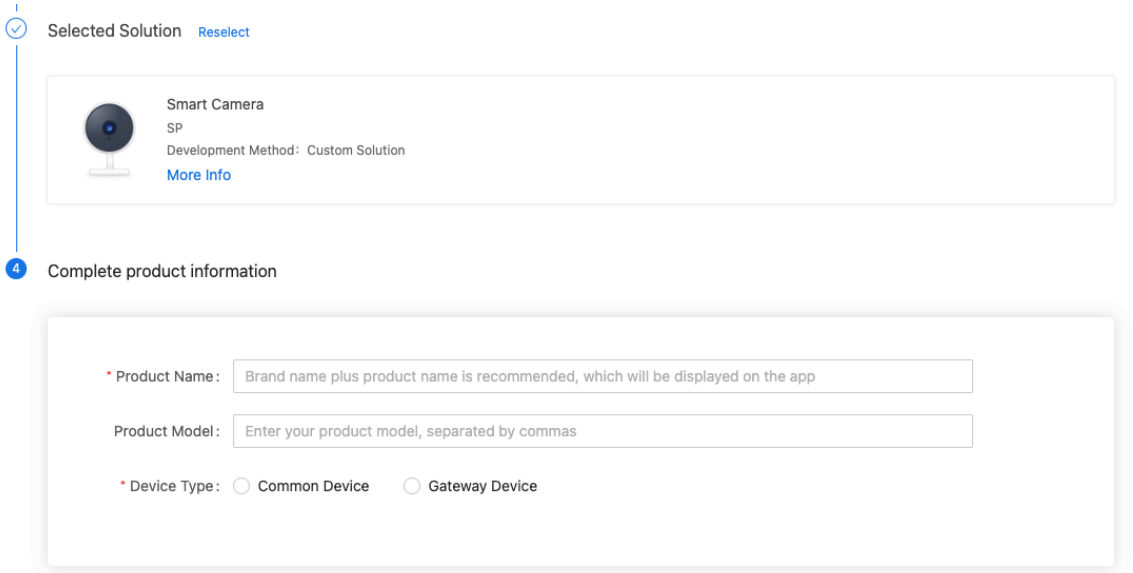
3. Choose **IP Camera** > **PT Camera**.



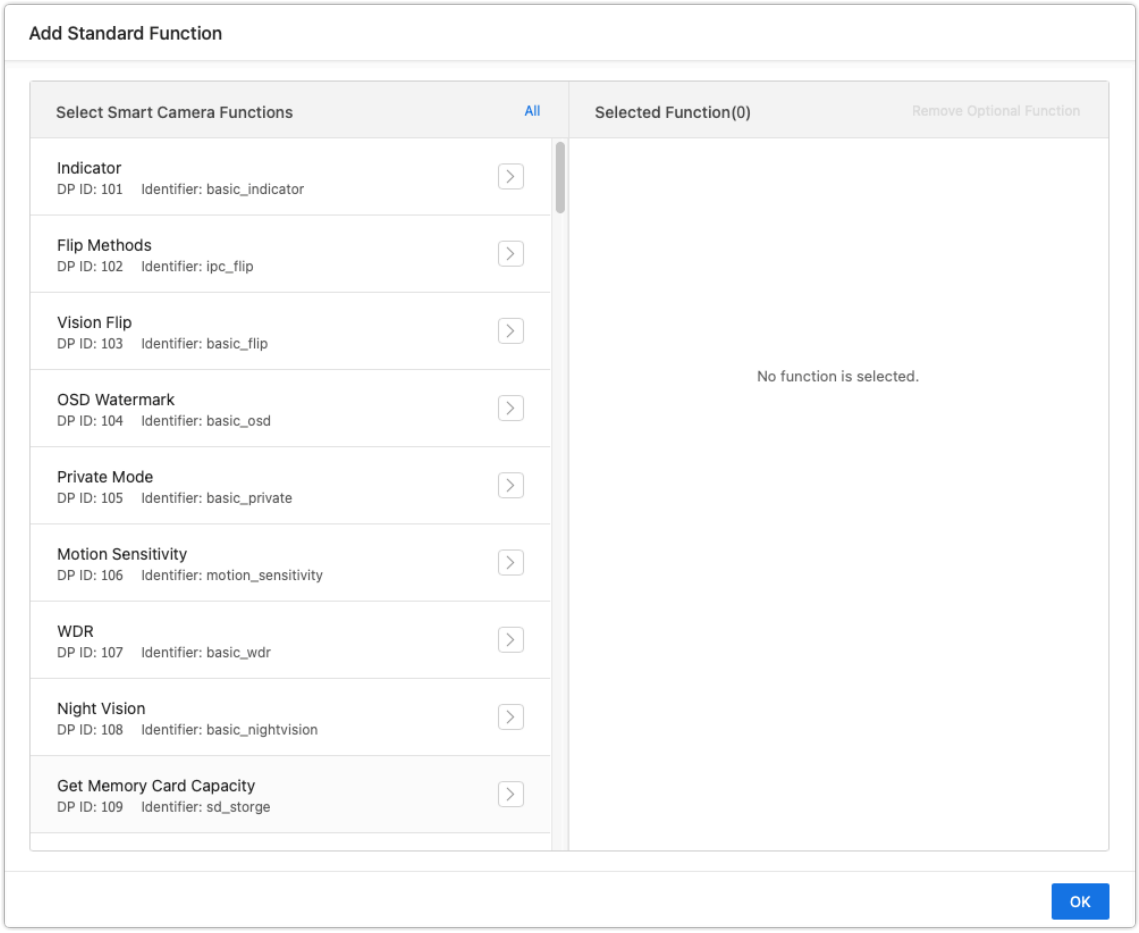
4. Choose **TuyaOS** for **Smart Mode** and choose **Custom Solution**.



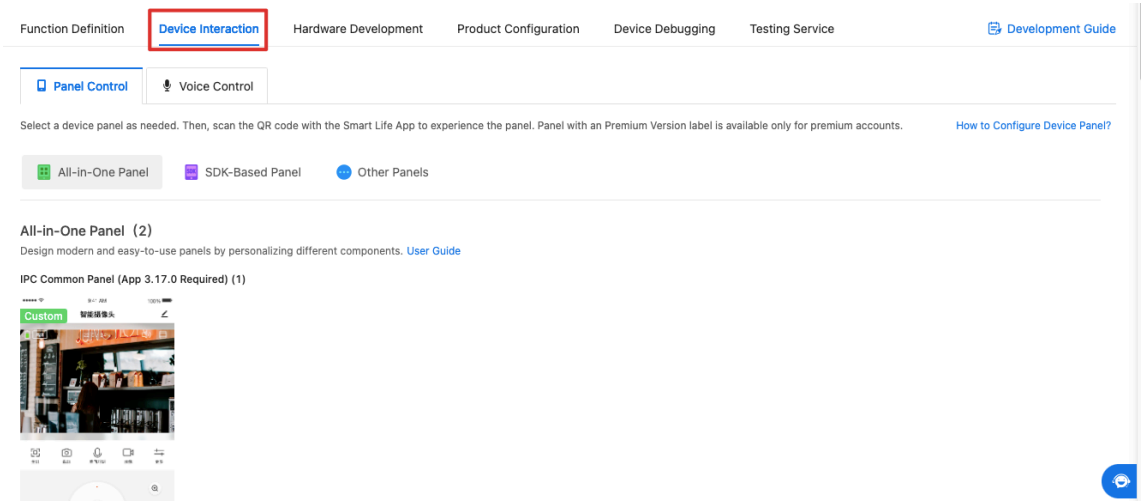
5. Complete the required information, such as product name and product model. Choose a **Device Type**. Click **Create**.



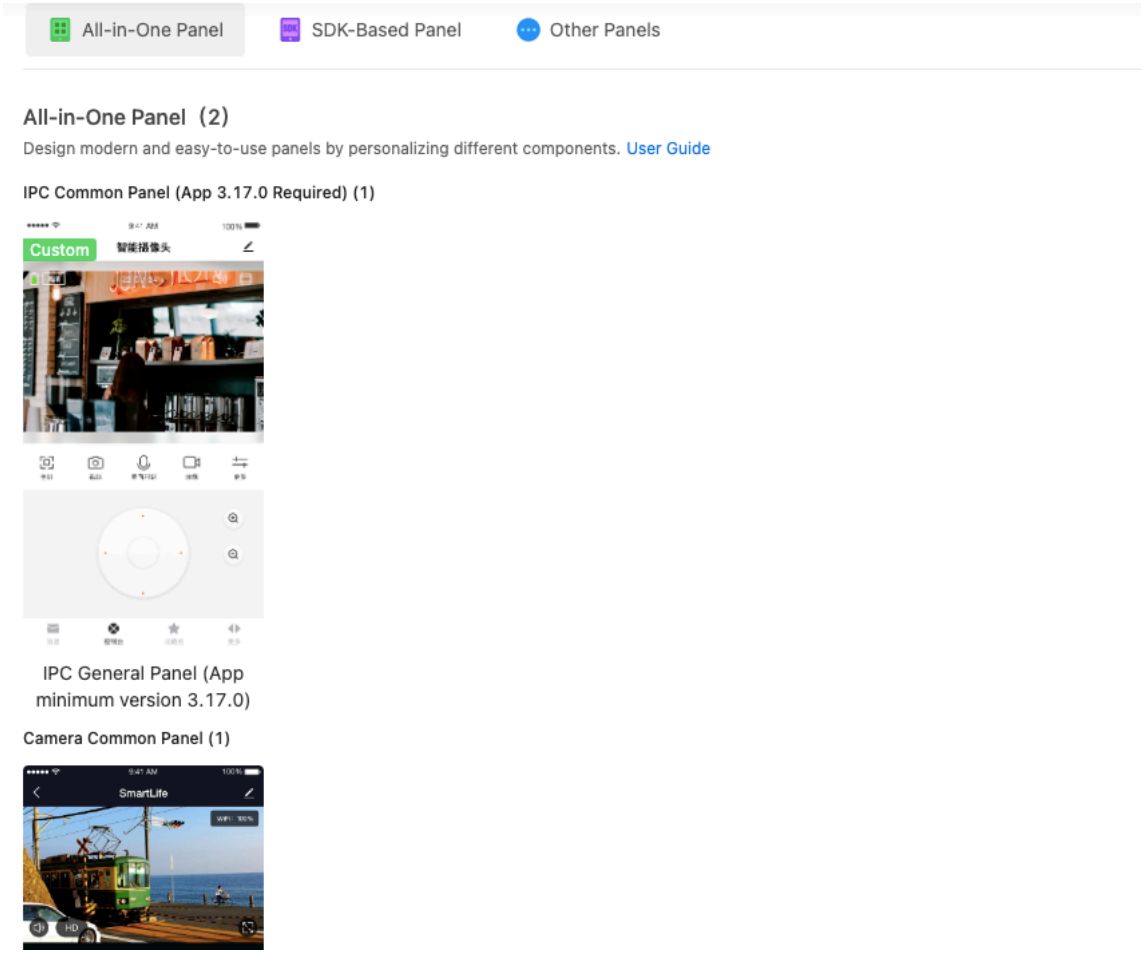
6. Select the standard function as needed.



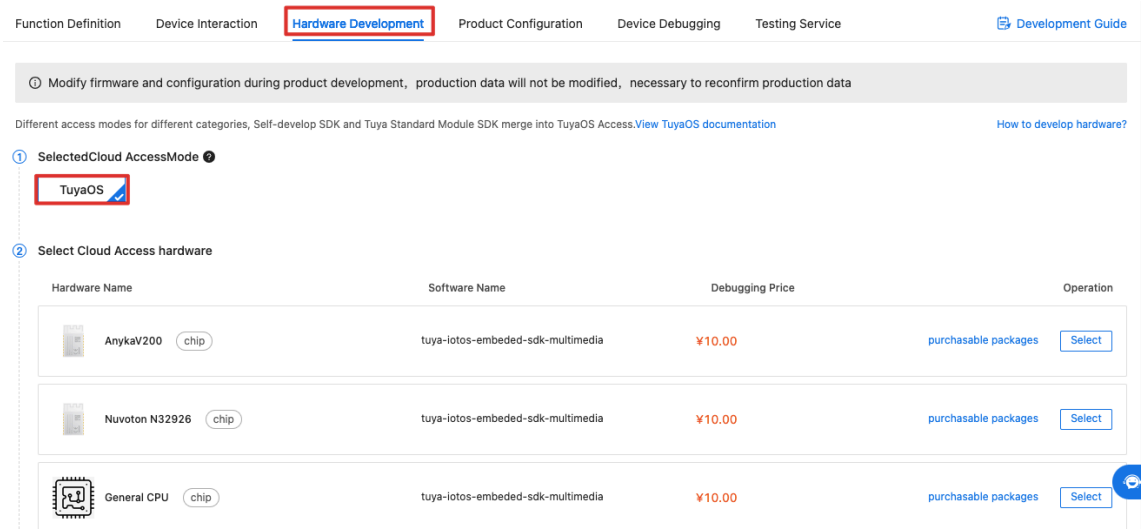
7. Click **Device Interaction**.



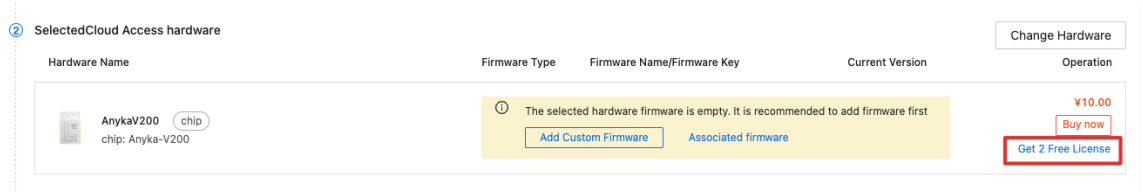
8. Scroll down the page and select a panel.



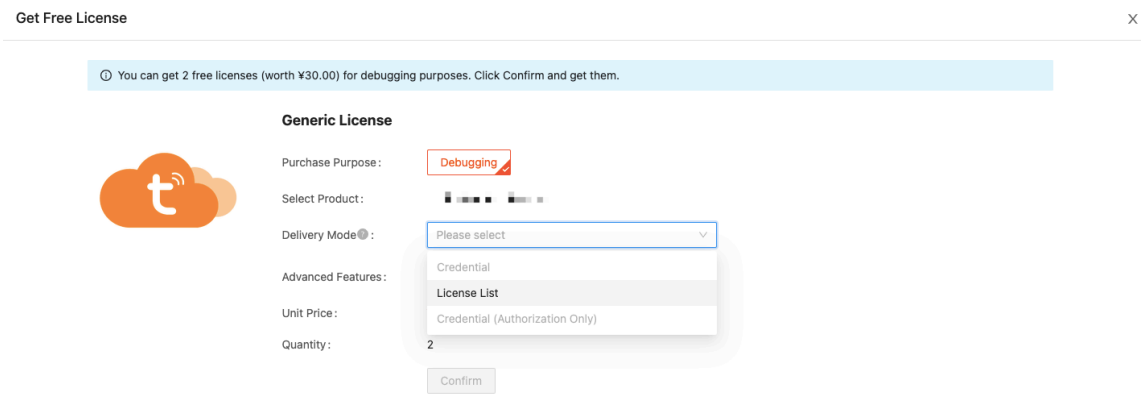
9. Click **Hardware Development**. Choose **TuyaOS** and select hardware.



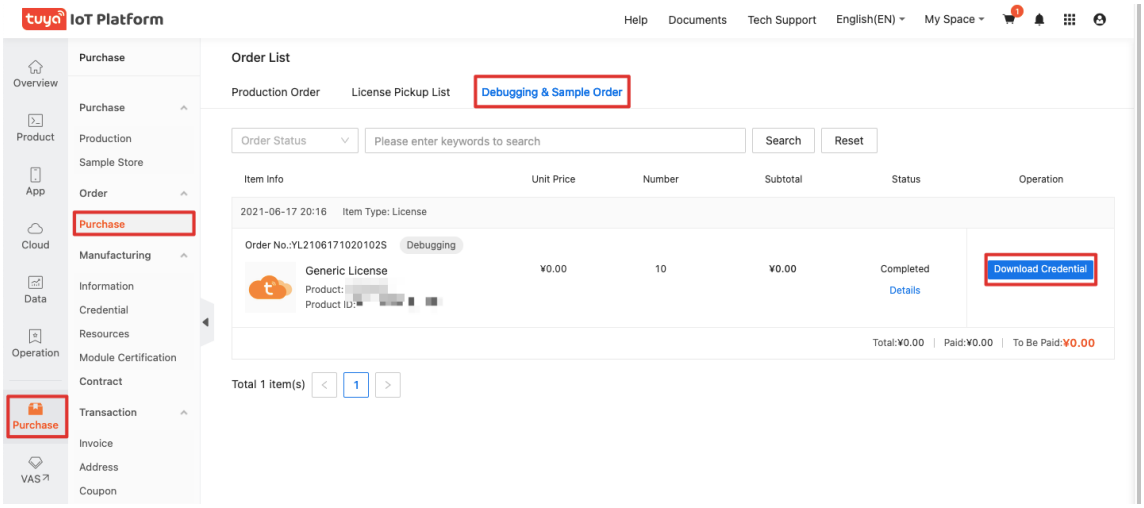
10. Click **Get 2 Free Licenses** to get the license for authorizing a device to connect to the Tuya Developer Platform.



11. Choose a **Delivery Mode**.



12. From the left navigation pane, choose **Purchase** > **Debugging & Sample Order**. Click **Download Credential**.



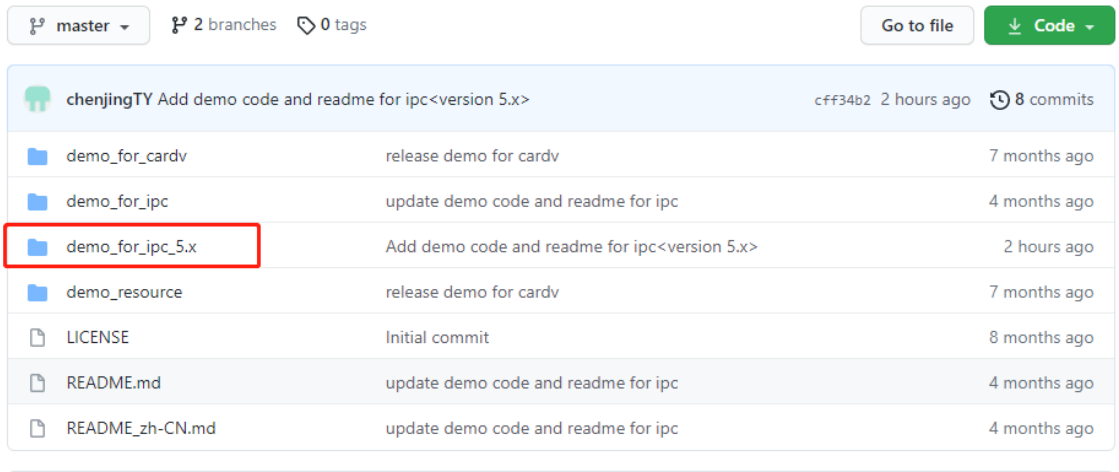
Now, you have created a product and got 2 licenses.

5.2 Step 2: Get IPC SDK

Download the SDK from GitHub. If you cannot find the desired SDK, submit a service ticket.

5.3 Step 3: Run the demo

1. Download the demo.



2. Unzip your SDK and change the video file path.

   Unzip the SDK.

   Copy the SDK folder that includes `include` and `lib` folders to the `demo_for_ipc_5.x` folder.

   Change the video file path

   - Method 1 (without editing code)

     Copy `demo_resource` folder in the demo to `demo_for_ipc_5.x` and rename `demo_resource` as `resource`.

     When you run the code, specify the current folder as the video file path, which follows `-r` in the following code.

     ```
     ./tuya_ipc_demo -m 2 -p PID -u UUID -a AUTHKEY -r "./" -t "
        TOKEN"
     ```

   - Method 2 (editing code)

     Open the `user_main.c` in `demo_src` and go to line 359.

     ```
     strcpy(s_raw_path, "/tmp"); //Path where demo resources
        locates
     ```

     Change the video file path.

The filename defaults to `resource` in the code. You can edit the path directly in the code or change the folder name to `resource` and then change the video file path.

3. Compile the code.

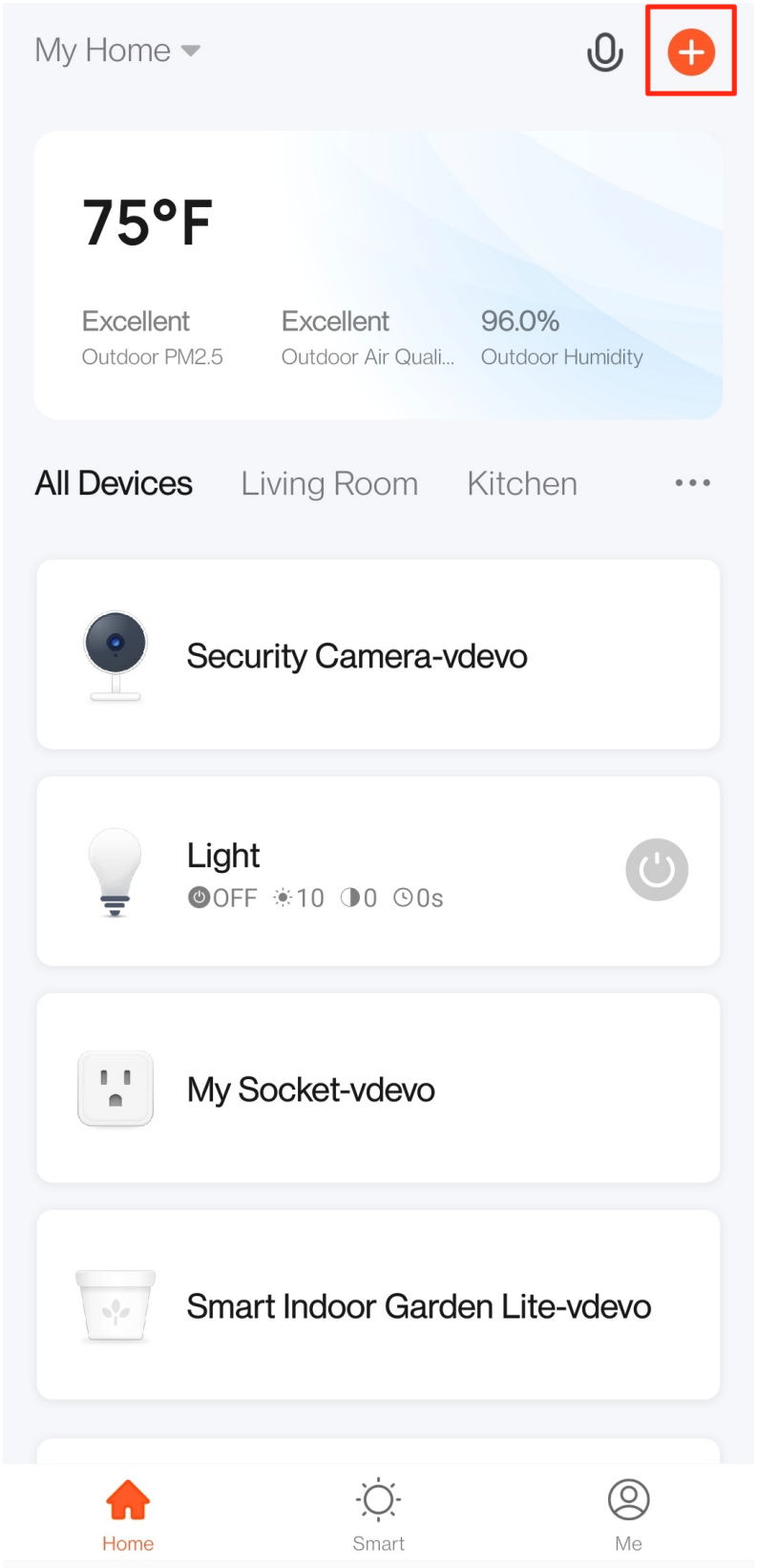
   In the terminal, navigate to the `demo_for_ipc_5.x` folder and run the following code.

   ```
   1 #make APP_NAME=demo
   ```

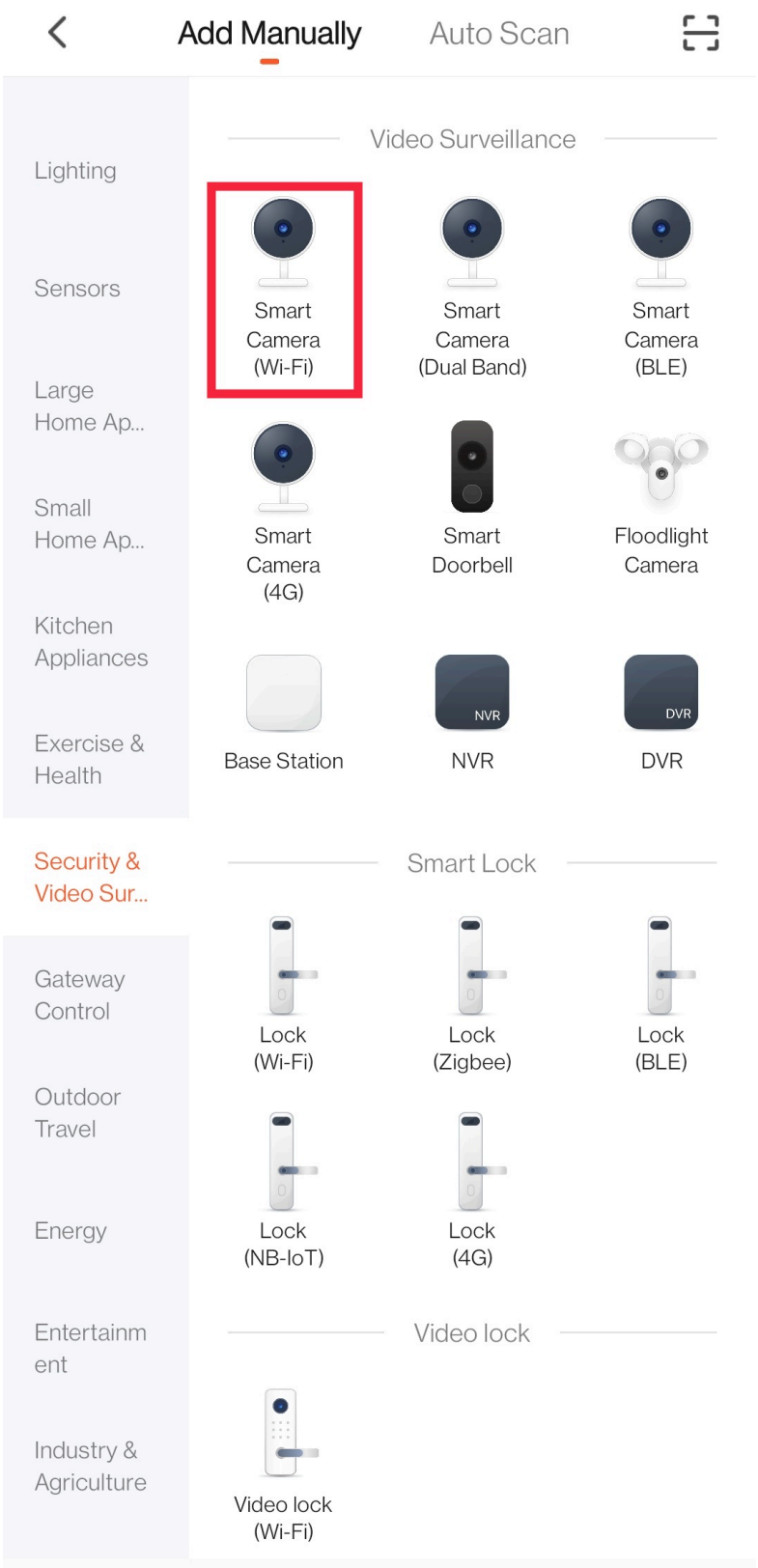
   When the compilation is completed, you will see `[Build APP Finish]`.

   The output is located in `demo_for_ipc_5.x/output/`. `[tuya_ipc_demo]` is the executable file.

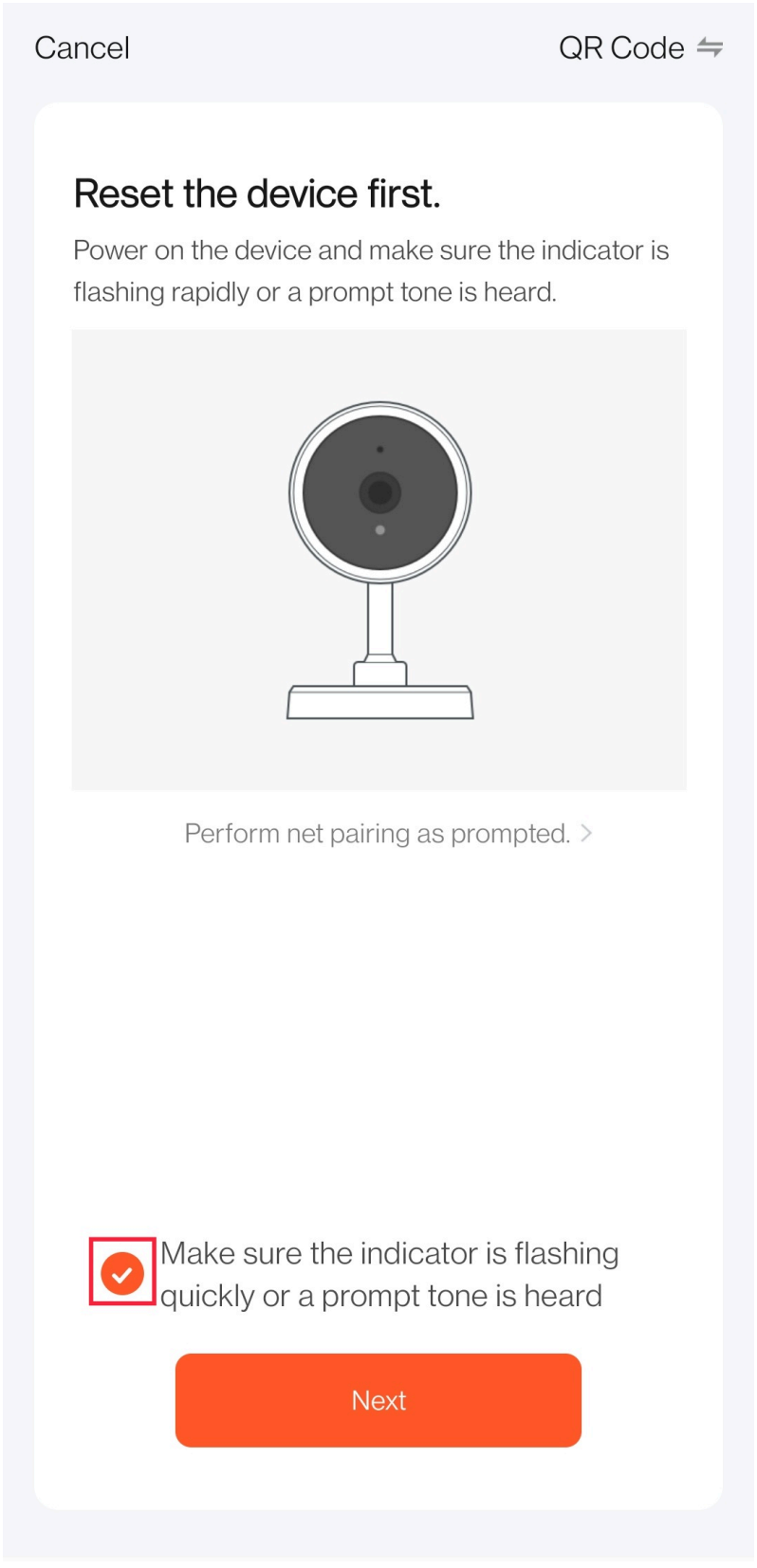
4. Open the Smart Life app and tap the **+** icon in the top right corner to add a device.


My Home
75°F
Excellent
Outdoor PM2.5
Excellent
Outdoor Air Quali...
96.0%
Outdoor Humidity
All Devices
Living Room
Kitchen
Security Camera-vdevo
Light
OFF 10 0 0s
My Socket-vdevo
Smart Indoor Garden Lite-vdevo
Home
Smart
Me

5. Tap **Security & Video Surveillance** and select **Smart Camera**.

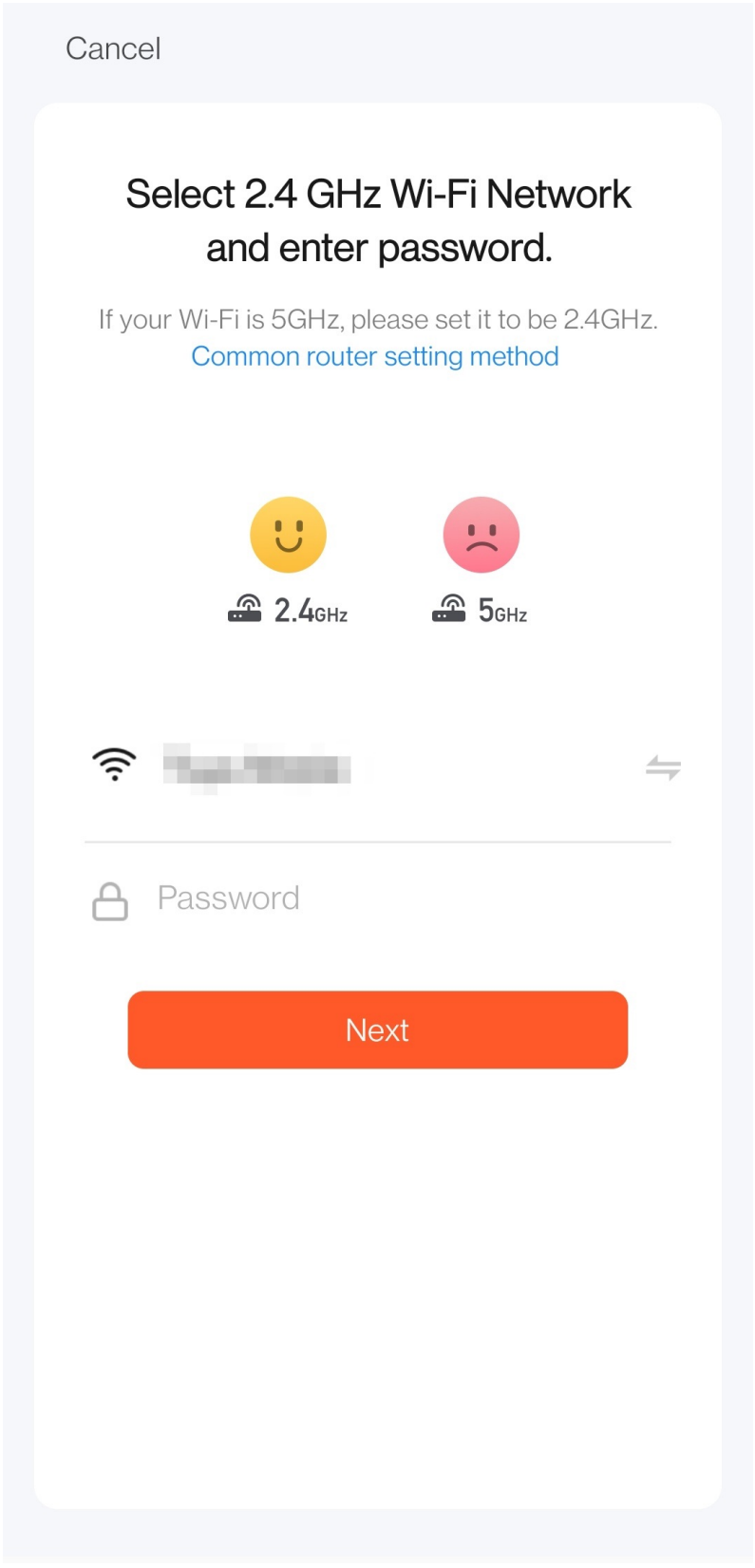

Add Manually
Auto Scan
Lighting
Sensors
Large Home Ap...
Small Home Ap...
Kitchen Appliances
Exercise & Health
Security & Video Sur...
Gateway Control
Outdoor Travel
Energy
Entertainment
Industry & Agriculture
Video Surveillance
Smart Camera (Wi-Fi)
Smart Camera (Dual Band)
Smart Camera (BLE)
Smart Camera (4G)
Smart Doorbell
Floodlight Camera
Base Station
NVR
DVR
Smart Lock
Lock (Wi-Fi)
Lock (Zigbee)
Lock (BLE)
Lock (NB-IoT)
Lock (4G)
Video lock
Video lock (Wi-Fi)

6. Tap **Make sure the indicator is flashing quickly or a prompt tone is heard**.


Cancel
QR Code
Reset the device first.
Power on the device and make sure the indicator is flashing rapidly or a prompt tone is heard.
Perform net pairing as prompted.
Make sure the indicator is flashing quickly or a prompt tone is heard
Next

7. Enter the Wi-Fi name and password.

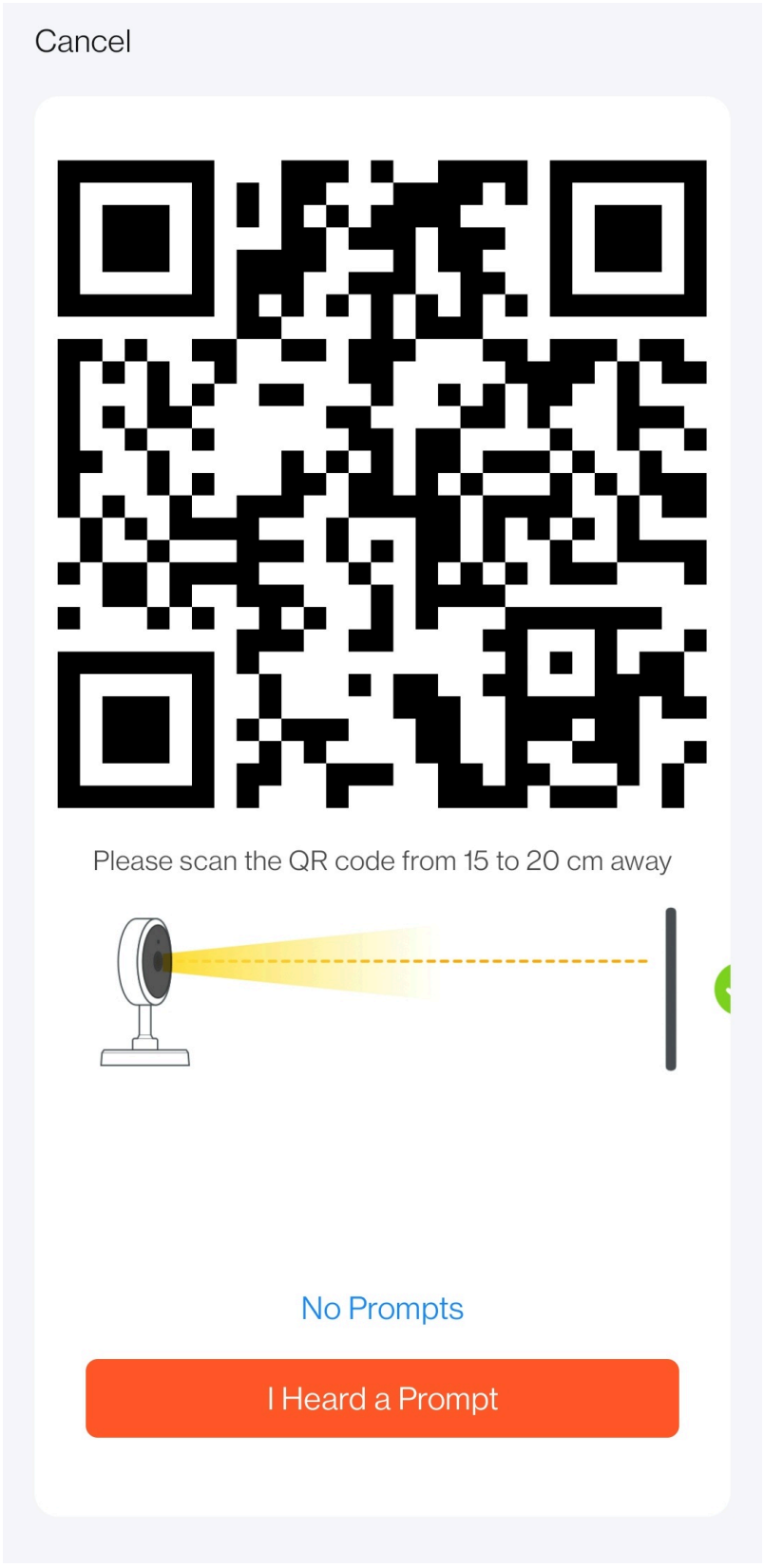
> Make sure to use the 2.4 GHz band to connect to devices.


Cancel
Select 2.4 GHz Wi-Fi Network and enter password.
If your Wi-Fi is 5GHz, please set it to be 2.4GHz.
Common router setting method
2.4GHz
5GHz
Password
Next

8. Generate and scan a QR code.

   Take a screenshot of the QR code, and scan the QR code with a QR code reader.

   > Do not tap **I Heard a Prompt** in this step. Do it in step 11.


Cancel
Please scan the QR code from 15 to 20 cm away
No Prompts
I Heard a Prompt

9. Extract the token from the obtained data.

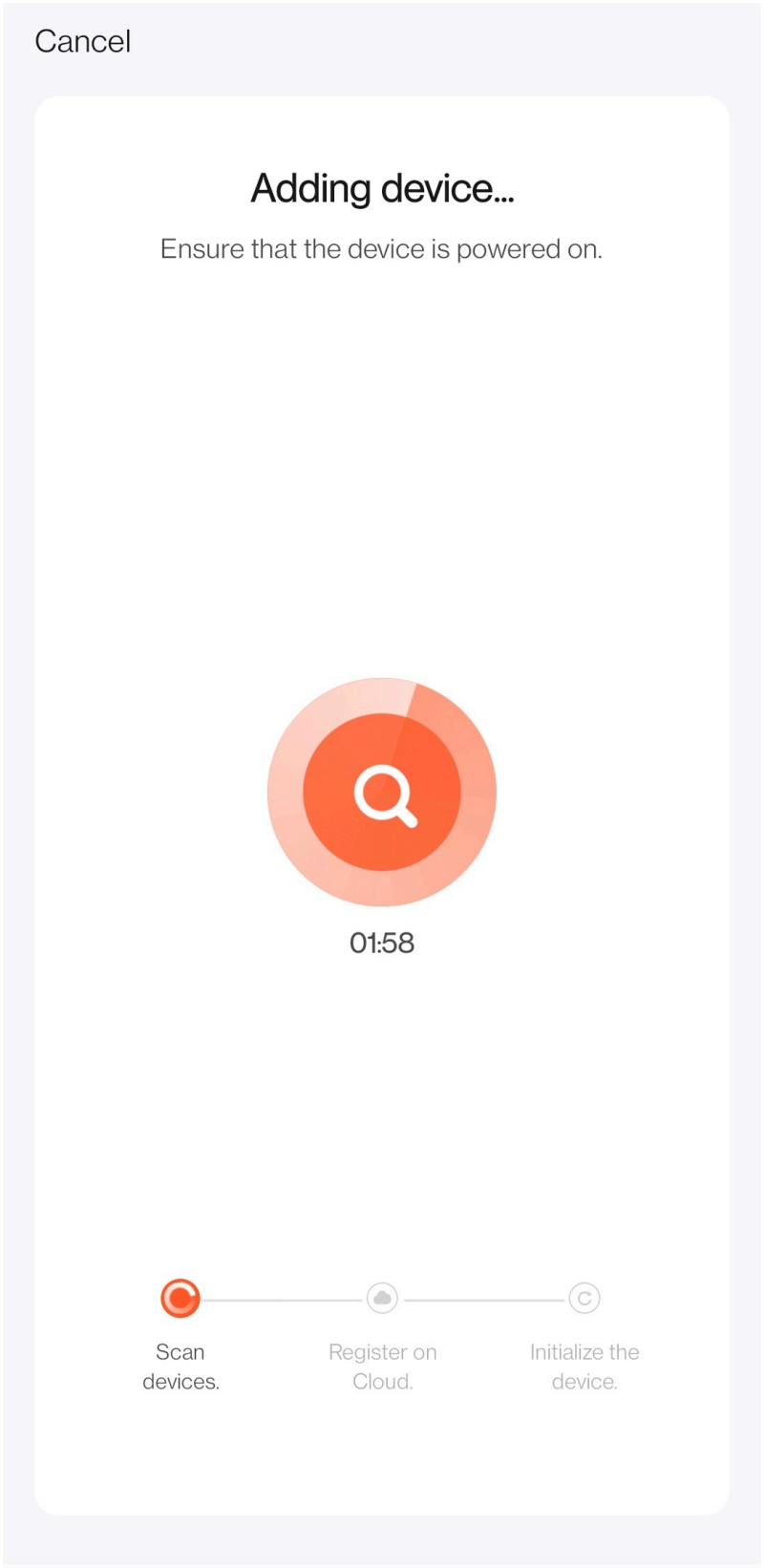
   Data obtained: `{"p":"test","s":"test","t":"AYYBJt2OGv****"}`

   > The value after `t` is the token that is valid for 10 minutes. After the token expired, you need to repeat the process and get a new token.

10. Execute a command to run the executable in the `output/` file.

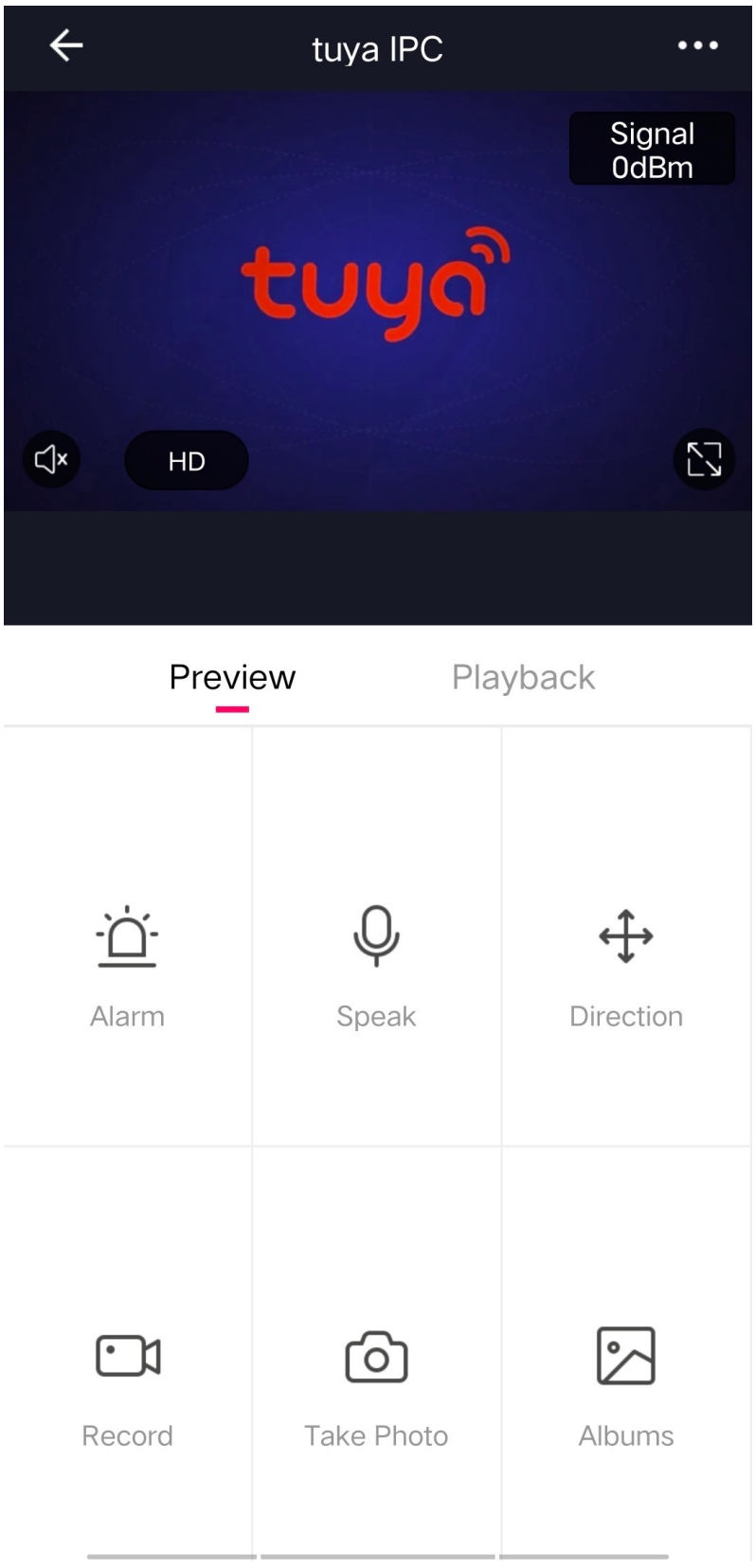
```
# Enter the following code.
#./tuya_ipc_demo -m 2 -p PID -u UUID -a AUTHKEY -r "./" -t "TOKEN"
#Note: `-m` is used to select a pairing mode. It has three valid val
ues from 0 to 2. `0` indicates Wi-Fi Easy Connect (EZ mode), `1` ind
icates access point (AP) mode, and `2` indicates debugging mode. Cha
nge the value of `PID`, `UUID`, and `AUTHKEY` to that of your produc
t that you create on the [Tuya Developer Platform](https://platform.
tuya.com/). `-r` is the video file path. In this code, the value aft
er `-r` indicates the current folder. `token` is obtained from QR co
de scanning in the steps above.
```

11. When the executable is running, you need to tap the button saying **I Heard a Prompt**.


Cancel
Adding device...
Ensure that the device is powered on.
01:58
Scan devices.
Register on Cloud.
Initialize the device.

12. Wait for the process to be completed.

    Tap the added device and view the live video stream. If no video is displayed, it indicates that `demo_resource` is not set correctly. Check the second step **Unzip your SDK and change the video file path**.

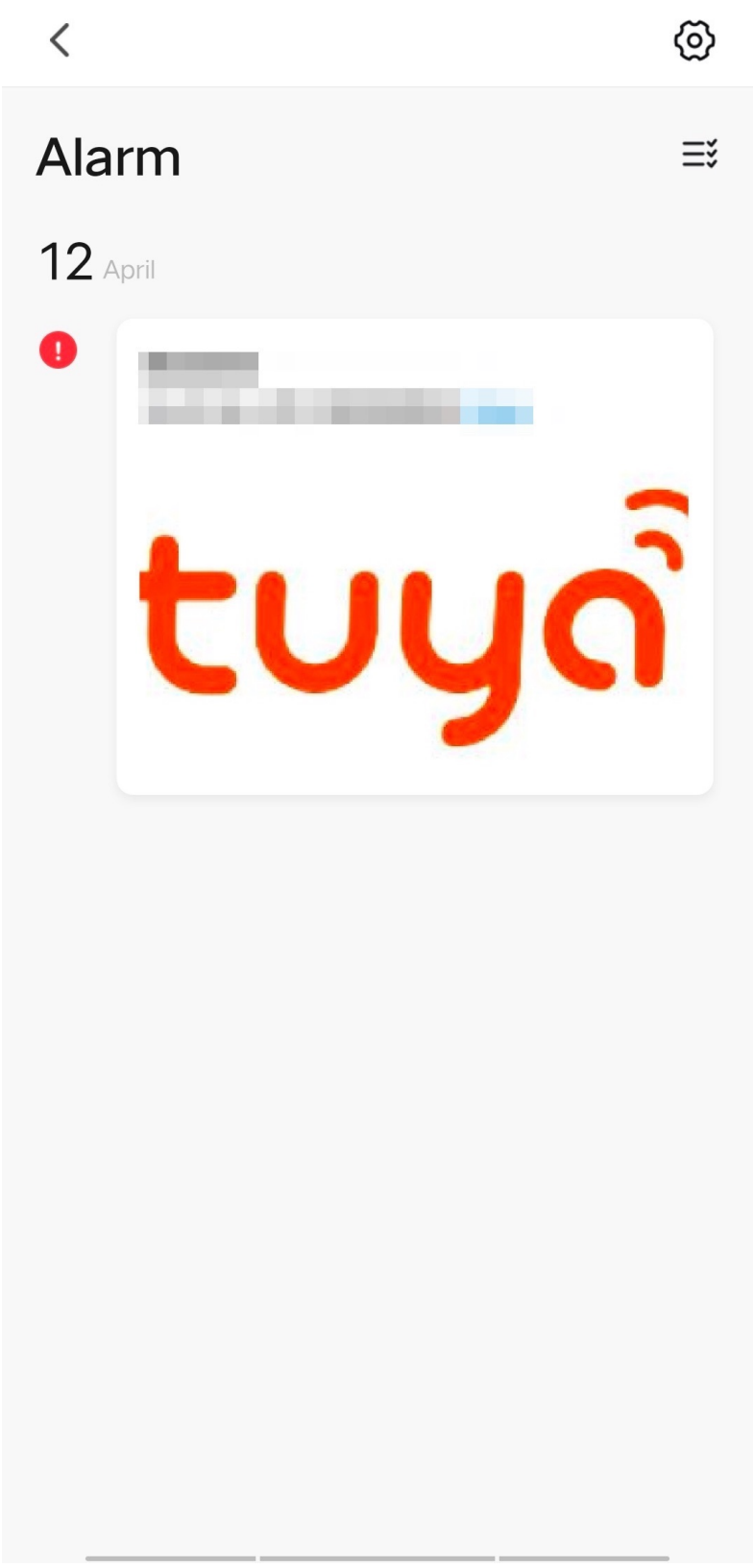

tuya IPC
Signal
0dBm
HD
Preview
Playback
Alarm
Speak
Direction
Record
Take Photo
Albums

The following table lists the commands for simulated operations. You can run a command to test different features.

Command	Description
start	Trigger a motion detection event.
stop	Stop a motion detection event.
status	Get the device’s activation status.
bell	Trigger doorbell alerts.

For example, If you run the following command, you will receive a notification of motion detection in the message center.

```
#start
```


Alarm
12 April

6 Application development

- In the `demo_src/user_main.c` file, change the save path of DB files, OTA files, and local recording files.
  :::info
  The DB file contains the device pairing information. It must be saved in the non-volatile memory.
  :::

  Make sure to check whether or not there is a slash (/) at the end of the path to save the DB file and local media file.

```
#define IPC_APP_STORAGE_PATH "/tmp/"
#define IPC_APP_UPGRADE_FILE "/tmp/upgrade.file"
#define IPC_APP_SD_BASE_PATH "/tmp/"
```

- Specify the `PID`, `UUID`, and `AuthKey`.
  - Method 1 (recommended for demo stage)

```
#./tuya_ipc_demo -m 2 -p PID -u UUID -a AUTHKEY -r "./" -t "TOKEN"
# Change the value of `PID`, `UUID`, and `AUTHKEY`. For the description of other parameters, see step 10 in the section Run the demo.
```

  - Method 2 (recommended for development stage)

```
#CHAR_T s_ipc_pid[64]="tuya_pid";           //pid
#CHAR_T s_ipc_uuid[64]="tuya_uuid";         //uuid
#CHAR_T s_ipc_authkey[64]="tuya_authkey";   //authkey
    # Change the value of `PID`, `UUID`, and `AUTHKEY`. These parameters are written to the flash memory, which will be retained even when the device is powered off.
```

- Change the macro of firmware version number.

  The version number must be set in the format xx.xx.xx, no more than 20 characters.

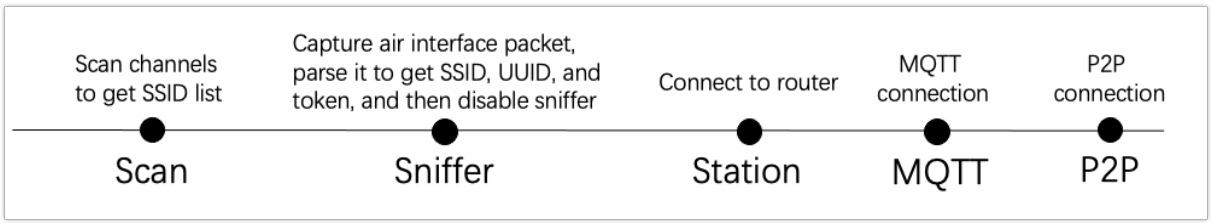
```
#define IPC_APP_VERSION "1.2.3"
```

The basic parameter setting is completed. Next, you can proceed with developing pairing modes.

7 Device pairing

The SDK provides complete framework functions for implementing device pairing guides. This section describes the pairing processes and the purpose of each framework function. You can follow the description of each function and implement specific features accordingly.

7.1 EZ mode (Wi-Fi Easy Connect)



Procedure

- Find the parameter `mode` in the `main` function. Select `WIFI_INIT_AUTO` mode and set the `token` as `NULL`.

```
1   CHAR_T token[30] = {0};  // A token is an identification code
       gene
2 rated by the cloud when users scan the QR code to pair a device. A
     t
3 oken is valid for 10 minutes.
4   WIFI_INIT_MODE_E mode = WIFI_INIT_AUTO;  // Specify a pairing
       mode
5 .
```

- The following interfaces are intended to be implemented by you. For the reference implementation, see the demo.

```
  // The interface to get the Wi-Fi mode, defaulting to sniffer mode
.
  OPERATE_RET tuya_adapter_wifi_get_work_mode(OUT WF_WK_MD_E *mode)
// Scan all the channels and transmit the scanned data to the struct
 `AP_IF_S**ap`.
  OPERATE_RET tuya_adapter_wifi_assign_ap_scan(IN CONST CHAR_T *ssid
,OUT AP_IF_S **ap)
// Set the Wi-Fi mode. When the SDK determines that the input packet
 data is correct, it sets the Wi-Fi status as station mode.
  OPERATE_RET tuya_adapter_wifi_set_work_mode(IN CONST WF_WK_MD_E mo
de)
// Register the sniffer callback.
  int tuya_adapter_wifi_sniffer_set(const bool en, const SNIFFER_CAL
LBACK cb)
// Perform network connection according to the parsed data.
  OPERATE_RET tuya_adapter_wifi_station_connect(IN CONST CHAR_T *ssi
d,IN CONST CHAR_T *passwd)
// Notify the SDK of the obtained IP address on a successful pairing
. This is a frequently called interface.
  OPERATE_RET tuya_adapter_wifi_station_get_status(OUT WF_STATION_ST
AT_E *stat)
```

- Pass in the received packet for Wi-Fi signal strength display.

```
OPERATE_RET tuya_adapter_wifi_station_get_conn_ap_rssi(OUT SCHAR_T *
rssi)
```

If you want to change the interface, consult technical support before any modification is made to avoid any unknown errors.

7.2 QR code mode

Procedure

- In `TUYA_IPC_SDK_START`, select `WIFI_INIT_AUTO` mode and set the `token` as `NULL`.

```
  CHAR_T token[30] = {0};  // A token is an identification code
      gene
rated by the cloud when users scan the QR code to pair a device. A
    t
oken is valid for 10 minutes.
  WIFI_INIT_MODE_E mode = WIFI_INIT_AUTO;  // Specify a pairing
      mode
.
```

- The process of pairing a device by QR code scanning

  1. Capture the QR code.
  2. Pass in the QR code image to the open source ZBar function for recognition. Ensure that the resolution is at least 320 × 240 pixels.
  3. Get the information of SSID, password, and token.
  4. Call `tuya_adapter_wifi_station_connect` to create a MQTT connection.

     The file `demo_src/tuya_ipc_qrcode_demo.c` provides reference implementation. You need to start the thread in the file to call the functions.

- The following interfaces are intended to be implemented by you. For the reference implementation, see the demo.

```
  // Get the Wi-Fi status.
  OPERATE_RET tuya_adapter_wifi_get_work_mode(OUT WF_WK_MD_E *mode
      )
// Get and parse data. Parse the QR code obtained from the camera.
  STATIC CHAR_T* __tuya_linux_get_snap_qrcode(VOID)
// Perform network connection according to the parsed data.
  OPERATE_RET tuya_adapter_wifi_station_connect(IN CONST CHAR_T *
      ssi
d,IN CONST CHAR_T *passwd)
```

If you want to change the interface, consult technical support before any modification is made to avoid any unknown errors.

7.3 Wired mode (scan devices in LAN)

- The device calls `TUYA_IPC_SDK_START` for SDK initialization.

- The following interfaces are intended to be implemented by you. For the reference implementation, see the `demo_src/tuya_ipc_wired_demo.c`.

```
1   // Get the IP address. When the device detects a connected
        router,
2  it returns the IP address.
3   OPERATE_RET tuya_adapter_wifi_get_ip(IN CONST WF_IF_E wf,OUT
        NW_IP
4 _S *ip)
5 // Get the MAC address of a device.
6   OPERATE_RET tuya_adapter_wifi_get_mac(IN CONST WF_IF_E wf,INOUT
        NW
7 _MAC_S *mac)
```

- The SDK broadcasts the IP address and MAC address through the router. When the mobile phone connects to the Wi-Fi network and receives the broadcast, it will request a token from the server. Then, the mobile phone sends the token to the device for pairing.

  If you want to change the interface, consult technical support before any modification is made to avoid any unknown errors.

7.4 Direct connection (QR code scanning)

- Make sure you have downloaded the direct connection supported IPC SDK, and the UUID has been tagged in the background.
- Make sure the device has been connected to the internet when the SDK runs.
- Call `TUYA_IPC_SDK_START` to initialize the SDK. Set `WIFI_INIT_MODE_E` as `WIFI_INIT_NULL` and `token` as `NULL`.
- Tap the scan icon in the top right corner of the mobile app to scan the QR code for device activation.

  If you want to change the interface, consult technical support before any modification is made to avoid any unknown errors.

8 Media-related features

8.1 Live preview

When the SDK is initialized, a 10-second media cache will be created by default. You can feed the media data into the ring buffer. The SDK library already enables data transmission, so related development is not required.

8.1.1 Implement functions

- Confirm video parameters in `TUYA_IPC_SDK_START`.

  We recommend that the size of the group of pictures (GOP) be set to two to three times the frame per second (FPS) and the resolution be 16:9. The bitrate cannot exceed 1.5 Mbit/s. H.264 and H.265 encoding formats are supported.

```
1  // Configure the mainstream as needed.
2  ipc_sdk_run_var.media_info.media_info.channel_enable[
       E_IPC_STREAM_VI
3  DEO_MAIN] = TRUE;   /* Specify whether to use the local HD video
       str
4  eam */
5  ipc_sdk_run_var.media_info.media_info.video_fps[
       E_IPC_STREAM_VIDEO_M
6  AIN] = 30;          /* FPS */
7  ipc_sdk_run_var.media_info.media_info.video_gop[
       E_CHANNEL_VIDEO_MAIN
8  ] = 30;             /* GOP */
9  ipc_sdk_run_var.media_info.media_info.video_bitrate[
       E_CHANNEL_VIDEO_
10 MAIN] = TUYA_VIDEO_BITRATE_1M; /* Rate limiting */
11 ipc_sdk_run_var.media_info.media_info.video_width[
       E_CHANNEL_VIDEO_MA
12 IN] = 640;              /*  Resolution of single frame width */
13 ipc_sdk_run_var.media_info.media_info.video_height[
       E_CHANNEL_VIDEO_M
14 AIN] = 360;          /*  Resolution of single frame height */
15 ipc_sdk_run_var.media_info.media_info.video_freq[
       E_CHANNEL_VIDEO_MAI
16 N] = 90000;         /* Clock rate */
17 ipc_sdk_run_var.media_info.media_info.video_codec[
       E_CHANNEL_VIDEO_MA
18 IN] = TUYA_CODEC_VIDEO_H264; /* Encoding format */
```

```
// Configure the substream as needed.
ipc_sdk_run_var.media_info.media_info.channel_enable[
    E_IPC_STREAM_VI
DEO_SUB] = TRUE;       /* Specify whether to use the local HD
    video st
ream */
ipc_sdk_run_var.media_info.media_info.video_fps[
    E_IPC_STREAM_VIDEO_S
UB] = 30;              /* FPS */
ipc_sdk_run_var.media_info.media_info.video_gop[
    E_IPC_STREAM_VIDEO_S
UB] = 30;              /* GOP */
ipc_sdk_run_var.media_info.media_info.video_bitrate[
    E_IPC_STREAM_VID
EO_SUB] = TUYA_VIDEO_BITRATE_512K; /* Rate limiting */
ipc_sdk_run_var.media_info.media_info.video_width[
    E_IPC_STREAM_VIDEO
_SUB] = 640;      /*  Resolution of single frame width */
ipc_sdk_run_var.media_info.media_info.video_height[
    E_IPC_STREAM_VIDE
O_SUB] = 360;     /*  Resolution of single frame height */
ipc_sdk_run_var.media_info.media_info.video_freq[
    E_IPC_STREAM_VIDEO_
SUB] = 90000;         /* Clock rate */
ipc_sdk_run_var.media_info.media_info.video_codec[
    E_IPC_STREAM_VIDEO
_SUB] = TUYA_CODEC_VIDEO_H264; /* Encoding format */
```

- Confirm audio parameters in `TUYA_IPC_SDK_START`.

  The audio format supports PCM, G.711u, and G.711a. The maximum upload size is 1,400 bytes. The audio sent from the app is fixed to 320 bytes. 8K and 16K audio are supported.

```
ipc_sdk_run_var.media_info.media_info.channel_enable[
    E_IPC_STREAM_AU
DIO_MAIN] = TRUE; /* Specify whether to enable local audio
    collectio
n */
ipc_sdk_run_var.media_info.media_info.audio_codec[
    E_IPC_STREAM_AUDIO
_MAIN] = TUYA_CODEC_AUDIO_PCM;/* Encoding format */
ipc_sdk_run_var.media_info.media_info.audio_sample [
    E_IPC_STREAM_AUD
IO_MAIN]= TUYA_AUDIO_SAMPLE_8K;/* Sampling rate */
ipc_sdk_run_var.media_info.media_info.audio_databits [
    E_IPC_STREAM_A
UDIO_MAIN]= TUYA_AUDIO_DATABITS_16;/* Bit width */
ipc_sdk_run_var.media_info.media_info.audio_channel[
    E_IPC_STREAM_AUD
IO_MAIN]= TUYA_AUDIO_CHANNEL_MONO;/* Channel */
ipc_sdk_run_var.media_info.media_info.audio_fps[
    E_IPC_STREAM_AUDIO_M
AIN] = 25;      /* Frame rate */
```

- Call `tuya_ipc_ring_buffer_open` to start a new session for media transmission.

  All the interfaces related to the ring buff support NVR and IPC. For the IPC, the parameters `device` and `channel` are generally set to `0`.

```
/* Start a new read-write session
[in]device: device serial number (to identify multiple devices)
[in]channel: lens serial number. A single device can have multiple
    l
enses. Set this parameter to 0 if a device only has one lens.
[in]stream: stream type (mainstream, substream)
[in]open_type: the type of the ring buffer of a session
*/
Ring_Buffer_User_Handle_S tuya_ipc_ring_buffer_open(INT_T device,
    IN
T_T channel, IPC_STREAM_E stream, RBUF_OPEN_TYPE_E open_type);
```

- Pass in the collected media to the ring buffer by calling `TUYA_APP_Put_Frame`.

```
/* Send the raw media data to the SDK.
[in] handle: the handle returned by `tuya_ipc_ring_buffer_open`
[in] p_frame: media frame to be transmitted.
*/
OPERATE_RET TUYA_APP_Put_Frame(Ring_Buffer_User_Handle_S handle,IN
    C
ONST MEDIA_FRAME_S *p_frame);
```

- Speaker on/off switch in `TUYA_APP_Enable_Speaker_CB` is intended to be implemented by you.

```
VOID TUYA_APP_Enable_Speaker_CB(BOOL_T enabled)
{
 printf("enable speaker %d \r\n", enabled);
//TODO
/* This function is used to turn on or off the speaker and is
   intend
ed to be implemented by you.
  If you do not need the speaker on/off control, leave this
     function
 empty. */
}
```

- `TUYA_APP_Rev_Audio_CB` implements audio playback.

```
VOID TUYA_APP_Rev_Audio_CB(IN CONST MEDIA_FRAME_S *p_audio_frame,
              TUYA_AUDIO_SAMPLE_E audio_sample,
              TUYA_AUDIO_DATABITS_E audio_databits,
              TUYA_AUDIO_CHANNEL_E audio_channel)
{
 printf("rev audio cb len:%u sample:%d db:%d channel:%d\r\n",
    p_audi
o_frame->size, audio_sample, audio_databits, audio_channel);
//PCM-Format 8K 16Bit MONO
//TODO
/* The audio playback is intended to be implemented by you. */
}
```

- The app sends the audio in G.711u format by default. If the device requires PCM audio, you can call `tuya_g711_decode` to transcode the audio.

````
**Parameters**:
- `1`: `TUYA_G711_MU_LAW`
- `2`: the address of resolved data
- `3`: the length of resolved data
- `4`: the output address after data is resolved
- `5`: the length after data is resolved
```c
int tuya_g711_decode(unsigned char type, unsigned short *src, unsign
ed int srcLen, unsigned char *drc, unsigned int *pOut);
```
````

8.1.2 Development processes

1. Configure the video mainstream and substream and audio parameters in `TUYA_IPC_SDK_START`.
2. Enable the video thread and audio thread in the main function.

```
pthread_t h264_output_thread;
pthread_create(&h264_output_thread, NULL, thread_live_video, NULL)
    ;
pthread_detach(h264_output_thread);
pthread_t pcm_output_thread;
pthread_create(&pcm_output_thread, NULL, thread_live_audio, NULL);
pthread_detach(pcm_output_thread);
```

3. Call `tuya_ipc_ring_buffer_open` in the audio thread and video thread to start a new session.
4. Collect the information of the media files in the thread function and call `TUYA_APP_Put_Frame` to transmit the collected video frame to the ring buffer.

   > You can implement audio transmission to the ring buffer the same way as you do for the video.

5. The P2P thread can automatically transmit data in the ring buffer to the mobile app.

9 Local recording

The SDK library has integrated the local storage feature. You just need to feed the media data into the ring buffer to complete the local storage setting.
This section describes how to implement the local storage of video and related settings.

9.0.1 Implement functions

- Enable the local storage in `TUYA_IPC_SDK_START`.

```
1    ipc_sdk_run_var.local_storage_info.enable = 1;// 1: enable. 0: dis
2 able.
```

  After continuous storage is enabled, the SDK will automatically save the video to the SD card. When the device is started, local storage initialization only needs to be called once. Local storage only supports an SD card in FAT32 format. If the device detects an unsupported SD card, it will report this error to the server through a DP. When users tap the SD card setting on the app, they will get a format error message. The app will guide users to format the SD card.

- If event storage is enabled, the device can call `tuya_ipc_start_storage` to start recording and call `tuya_ipc_stop_storage` to stop recording. Set the storage type to `E_ALARM_SD_STORAGE`.

  The maximum recording time of a single event is 10 minutes. If the device does not call `tuya_ipc_stop_storage` after 10 minutes, the SDK will automatically stop recording.

```
/**
* \fn OPERATE_RET tuya_ipc_start_storage
* \brief Start storage.
* \param[in] storage_type: The storage type.
* \ `E_ALARM_SD_STORAGE` indicates local storage. `E_ALARM_CLOUD_STO
RAGE` indicates cloud storage.
* \return If storage is started, `OPRT_OK` is returned. Other return
ed values indicate failure.
*/
OPERATE_RET tuya_ipc_start_storage(INT_T storage_type);
```

```
  /**
  * \fn OPERATE_RET tuya_ipc_stop_storage
      * \brief  Stop storage.
      * \param[in] storage_type: The storage type.
      * \`E_ALARM_SD_STORAGE` indicates local storage. `E_ALARM_CLOUD_S
TORAGE` indicates cloud storage.
      * \return If storage is stopped, `OPRT_OK` is returned. Other ret
urned values indicate failure.
  */
OPERATE_RET tuya_ipc_stop_storage(INT_T storage_type);
```

- You can implement SD card detection in `tuya_ipc_sd_get_status` as needed.

  When the device detects unsupported SD cards such as SD cards in NTFS format, `abnormal` will be returned. Otherwise, `normal` will be returned.

```
E_SD_STATUS tuya_ipc_sd_get_status(VOID)
{
// This function is intended to be implemented by you.
}
```

  To prevent data loss such as footage, call the function `sync()`; before the system is restarted or enters sleep mode to ensure that all data in the cache can be written to the local storage.

9.0.2 Development processes

1. Enable the local storage in the main function and set parameters.

```
ipc_sdk_run_var.local_storage_info.enable = 1;// 1: enable. 0: disab
le.
ipc_sdk_run_var.local_storage_info.max_event_num_per_day = 500;// Th
e maximum events that can be stored per day.
ipc_sdk_run_var.local_storage_info.skills = 0;// 0 indicates all fea
tures are enabled.
ipc_sdk_run_var.local_storage_info.sd_status_cb = tuya_ipc_sd_status
_upload ;// Update the SD card status.
strcpy(ipc_sdk_run_var.local_storage_info.storage_path, IPC_APP_SD_B
ASE_PATH);// Set the path used for local storage.
```

2. Call `TUYA_IPC_SDK_START` in the main function to run the SDK.
3. If `tuya_ipc_sdk_mqtt_online_proc` detects that local storage is enabled, the SDK will call `tuya_ipc_ss_init`. `tuya_ipc_ss_init` includes a series of features such as local storage initialization, SD card detection, and enabling the storage thread.

10 Data points (DPs)

DPs are used to control different features of a device. You need to set DPs for the device and report the DP status to the server after setting. The `tuya_ipc_dp_utils.h` file specifies the common DPs. You can implement DPs based on the sample code. You can also add and set custom DPs by referring to the existing DP code of the same data type.

You can use five data types including Boolean, value, string, enum, and bitmap (reserved) to set required DPs. For more information, see the struct `TY_OBJ_DP_VALUE_U` .

10.1 Read and write DP data

> You can assign the write and read permission to the DP data as needed.

- `tuya_app_write_INT` and `tuya_app_write_STR` implement DP data storage.

```
/* Configure parameters for the local storage of DP data.
In this sample code, the local storage path is specified as `/tmp`.
You can change it to the required path.
Note that the substituting path must be the same as the one for DP data read/write.*/
STATIC VOID __tuya_app_write_INT(CHAR_T *key, INT_T value)
{
 //TODO
  CHAR_T tmp_cmd[128] = {0};
  snprintf(tmp_cmd, 128, "mkdir -p /tmp/tuya.cfgs/;echo %d > /tmp/tuya.cfgs/%s", value, key);
  printf("write int exc: %s \r\n", tmp_cmd);
  system(tmp_cmd);
}
STATIC VOID __tuya_app_write_STR(CHAR_T *key, CHAR_T *value)
{
 //TODO
 CHAR_T tmp_cmd[256] = {0};
 snprintf(tmp_cmd, 256, "echo %s > /tmp/tuya.cfgs/%s", value, key);
 printf("write STR exc: %s \r\n", tmp_cmd);
 system(tmp_cmd);
}
```

- tuya_app_read_INT and **tuya_app_read_STR** implement DP data read operation. For the full sample code, see the demo.

```
1        /* Configure parameters for reading local DP data. You can
             change
2   the path to the required one.
3        Note that the substituting path must be the same as the one
             for t
4  he local storage of DP data.*/
5    STATIC INT_T __tuya_app_read_INT(CHAR_T *key)
6    {
7     //TODO
8     CHAR_T tmp_file[64] = {0};
9     snprintf(tmp_file, 64, "cat /tmp/tuya.cfgs/%s", key);
10   printf("read int exc: %s \r\n", tmp_file);
11    FILE *p_file = popen(tmp_file, "r");
12   STATIC INT_T __tuya_app_read_STR(CHAR_T *key, CHAR_T *value,
        INT_T
13 value_size)
14   {
15    //TODO
16    memset(value, 0, value_size);
17    CHAR_T tmp_file[64] = {0};
18    snprintf(tmp_file, 64, "cat /tmp/tuya.cfgs/%s", key);
19    printf("read str exc: %s \r\n", tmp_file);
20    FILE *p_file = popen(tmp_file, "r");
21   }
```

10.2 On screen display (OSD) timestamp

- The DP ID of the OSD timestamp is DP 104.
- The DP value sent from the server enables the device to add a timestamp to the video or remove it from the video. On the device, you can set where to place a timestamp on the video.

10.2.1 Development processes

1. Add the following code to tuya_ipc_dp_utils.h.

```
1 #define TUYA_DP_WATERMARK          104
```

2. Add the following code to `s_dp_table[]` in `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_WATERMARK
 {TUYA_DP_WATERMARK,              handle_DP_WATERMARK},
#endif
```

`handle_DP_WATERMARK` is intended to be implemented by you. The sample code is as follows.

```
#ifdef TUYA_DP_WATERMARK
STATIC VOID handle_DP_WATERMARK(IN TY_OBJ_DP_S *p_obj_dp)
{
BOOL_T watermark_on_off = check_dp_bool_invalid(p_obj_dp);
IPC_APP_set_watermark_onoff(watermark_on_off);
watermark_on_off = IPC_APP_get_watermark_onoff();
respone_dp_bool(TUYA_DP_WATERMARK, watermark_on_off);
}
#endif
```

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

11 Motion detection

The SDK provides functions to implement motion detection on/off, sensitivity adjustment, timing, intermittent control, and data reporting. You can call the SDK API to implement motion detection and motion tracking.

11.0.1 Implement functions

- The DP ID of motion alert is DP 134.
- The DP ID of motion sensitivity is DP 106.

Refer to the process of enabling OSD timestamp.

- Call `Tuya_Ipc_Motion_Init` to initialize motion detection.

```
/*
    ******************************************************************
**************
* Init input config.
******************************************************************
**************/
OPERATE_RET Tuya_Ipc_Motion_Init(TUYA_MOTION_TRACKING_CFG mt_cfg);
```

The following describes the initialization parameters of the struct `TUYA_MOTION_TRACKING_CFG` .

```
typedef struct _TUYA_MOTION_TRACKING_CFG
{
  INT_T frame_w;              // The width of the video frame.
  INT_T frame_h;          // The height of the video frame.
  INT_T y_thd;          // The threshold of motion detection, which d
efaults to 30. You can set a threshold from 0 to 1255. The recommend
ed value is 540. Smaller values will trigger motion detection more e
asily.
  INT_T sensitivity;      // Motion detection sensitivity. The value
 is from 1 to 7. The larger the value, the more sensitive the device
.
  TUYA_RPERCENT_ECT roi;  // Select a region of interest (ROI). To s
et this value, see the struct TUYA_RECT.
  TUYA_MULTI_MD_REGION motion_region;// Set multiple ROIs. The ROI ar
ea can be between 1×1 and 5×5.
  INT_T tracking_enable;  // Motion tracking on/off. If you want to
enable motion detection only, set this value to 0. To enable motion
tracking, set it to 1.
}TUYA_MOTION_TRACKING_CFG;
```

The following describes the initialization parameters of the struct TUYA_AI_RECT.

```
typedef struct _TUYA_AI_RECT
{
  INT_T left;     // The percentage of the x-coordinate to the width
 of a video frame in the region coordinate. The value must be rounde
d. For example, if the x-coordinate is 128, the percentage value is
10%.
  INT_T top;      The percentage of the y-coordinate to the height of
 a video frame in the region coordinate. The value must be rounded.
For example, if the x-coordinate is 72, the percentage value is 10%.
  INT_T right;    // The percentage of the width of the selected ROI
 to the width of a video frame in the region coordinate. The value m
ust be rounded. For example, if the width of the selected ROI is 256
, the percentage value is 20%.
  INT_T bottom;    // The percentage of the height of the selected R
OI to the height of a video frame in the region coordinate. The valu
e must be rounded. For example, if the width of the selected ROI is
144, the percentage value is 20%.
}TUYA_AI_RECT;
```

The following figure shows how the struct `TUYA_AI_RECT` works.

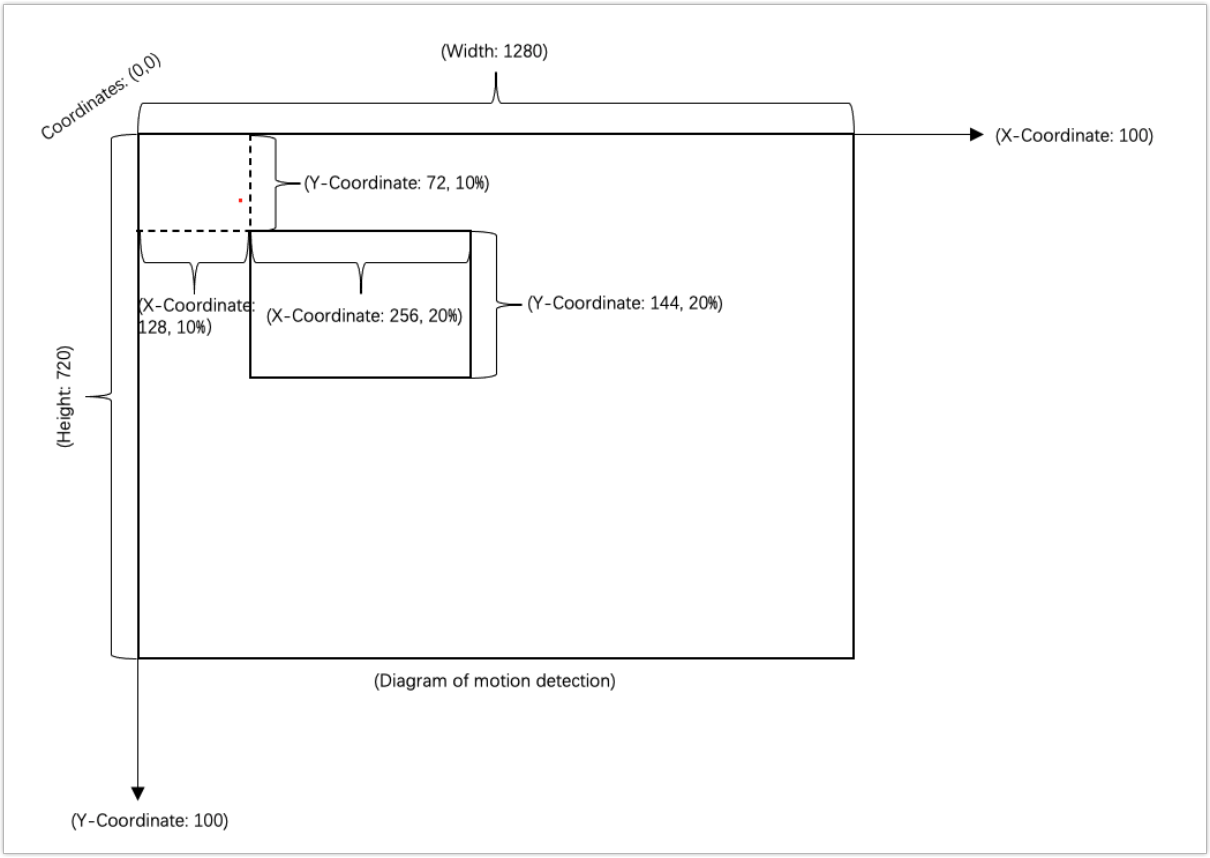


Figure 1: img

- After the initialization function `Tuya_Ipc_Motion_Init` is called, to modify the parameters of motion detection, call `Tuya_Ipc_Set_Motion`.

```
/*
    ********************************************************************
**************
* Set config dynamically.
********************************************************************
**************/
OPERATE_RET Tuya_Ipc_Set_Motion(TUYA_MOTION_TRACKING_CFG mt_cfg);
```

`Tuya_Ipc_Get_Motion` can get the parameters of motion detection.

```
/*
    ************************************************************************
**************
* Get config dynamically.
************************************************************************
**************/
void Tuya_Ipc_Get_Motion(TUYA_MOTION_TRACKING_CFG *mt_cfg);
```

- The device calls the `Tuya_Ipc_Motion` at regular intervals to pass in the collected YUV data. If the SDK tells the difference between two images and triggers an alert, `1` is returned. Otherwise, `0` is returned.

  The image must be in YUV420 format. You can fine-tune values for motion tracking. We recommend that you use the average results of five motion tracking actions as the rotation angle of the servo. This can minimize the occurrence of missed tracking.

  The frequency of calling `Tuya_Ipc_Motion` depends on the fine-tuned values for motion tracking. See the description of parameters in the following sample code.

```
/*
    ************************************************************************
**************
* Execute Motion Detect \ Motion Tracking.
  * in_data      The input image that must be in YUV420 format.
  * motion_flag    Returns the value of motion detection.
        0 indicates no motion. 1 indicates motion is detected.
  * motion_point    If motion tracking is enabled, this parameter
     wi
ll return the coordinates of the center point of the largest
    moving
object.
             If values of x and y coordinates are 0, it indicates
                 no
 motion. Otherwise, motion is detected.
             If `tracking_enable` is 0, (0, 0) is returned.
************************************************************************
**************/
OPERATE_RET Tuya_Ipc_Motion(UCHAR_T *in_data, INT_T *motion_flag,
    TU
YA_POINT * motion_point);
```

If `Tuya_Ipc_Motion` tells the difference between two images and triggers motion tracking, it returns **x and y coordinates** of the current position of the moving object. See the following legend.

With the IPC used as the point of reference, the x coordinate value denotes a location that is relative to the IPC to the east or west, and the y coordinate value denotes a location that is relative to the IPC to the north or south. The device calculates the rotation angle of the servo based on the returned coordinates and controls the servo to move to the decided angle.

- Once the device detects a motion and captures images, it calls `tuya_ipc_notify_alarm` to upload images to the server.

```
/**
* \fn OPERATE_RET tuya_ipc_notify_alarm
  * \ Send alerts to the cloud and the mobile app.
  * \param[in] snap_buffer: The address of the current snapshot.
  * \param[in] snap_size: The size of the snapshot, in bytes.
  * \param[in] name: The editable event type, NOTIFICATION_NAME_E
  * \param[in] is_notify: If it is true, the snapshot is sent to
      the
 in-app message center and the cloud storage. If it is false, the
    sn
apshot is sent to the cloud storage only.
* \return OPERATE_RET
*/
OPERATE_RET tuya_ipc_notify_alarm(CONST CHAR_T *snap_buffer, CONST
    U
INT_T snap_size, CONST NOTIFICATION_NAME_E name, BOOL_T is_notify)
   ;
```

- During the OTA update, to release memory, call `Tuya_Ipc_Motion_Release` to exit motion detection and motion tracking.

```
void Tuya_Ipc_Motion_Release();
```

11.0.2 Development processes

> Refer to the `thread_md_proc` in the demo.

1. Add the DP 134 of motion alert to enable the motion detection feature.
2. Enable the motion detection thread, as shown below.

```
pthread_create(&motion_detect_thread, NULL, thread_md_proc, NULL);
```

3. Call the time sync function in the motion detection thread.

```
// Get the UTC time.
OPERATE_RET tuya_ipc_get_utc_time(OUT TIME_T *time_utc);
```

4. Call `get_motion_status` in the motion detection thread to determine whether an object is moving.

> With `Tuya_Ipc_Motion`, you can implement the function used to determine whether an object is moving according to your chip platform.

```
int fake_md_status = 0;
int get_motion_status()
{
  // If motion is detected, 1 is returned.
  // Otherwise, 0 is returned. The returned value is stored in the p
arameter of `fake_md_status`.
 return fake_md_status;
}
```

5. If the object is moving, the device calls `tuya_ipc_start_storage` to enable local storage and cloud storage and calls `get_motion_snapshot` to capture images.

```
tuya_ipc_start_storage(E_ALARM_SD_STORAGE);      // Enable local storag
e.
tuya_ipc_start_storage(E_ALARM_CLOUD_STORAGE);  // Enable cloud stor
age.
```

```
// You can implement an image capture function according to your chi
p platform.
void get_motion_snapshot(char *snap_addr, int *snap_size)
{
 //we use file to simulate
 char snapfile[128];
 *snap_size = 0;
 extern char s_raw_path[];
 printf("get one motion snapshot\n");
 snprintf(snapfile,64,"%s/resource/media/demo_snapshot.jpg",s_raw_pa
th);
 FILE*fp = fopen(snapfile,"r+");
 //...
}
```

6. Call `tuya_ipc_notify_alarm` to send an alert to the cloud.

```
/**
* \fn OPERATE_RET tuya_ipc_notify_alarm
* \ Send alerts to the cloud and the mobile app.
* \param[in] snap_buffer: The address of the current snapshot.
* \param[in] snap_size: The size of the snapshot, in bytes.
* \param[in] name: The editable event type, NOTIFICATION_NAME_E
* \param[in] is_notify: If it is true, the snapshot is sent to the i
n-app message center and the cloud storage. If it is false, the snap
shot is sent to the cloud storage only.
* \return OPERATE_RET
*/
OPERATE_RET tuya_ipc_notify_alarm(CONST CHAR_T *snap_buffer, CONST U
INT_T snap_size, CONST NOTIFICATION_NAME_E name, BOOL_T is_notify);
```

7. If the object does not move and more than 10 seconds have passed since the last motion detection event, the device calls `tuya_ipc_stop_storage` to stop local storage and cloud storage.

```
tuya_ipc_stop_storage(E_ALARM_SD_STORAGE);   // Stop local storage.
tuya_ipc_stop_storage(E_ALARM_CLOUD_STORAGE);     // Stop cloud storage
.
```

12 Image scaling (SDK integration)

Image scaling refers to the resizing of captured images to fit the required resolution.

12.1 Development processes

- Call `Tuya_Ipc_Img_Resize` to pass in the image, parameter, and image output address.

  Parameters:

  - `in_data`: The address of the input image.
  - `paras`: The parameter of the input image.
  - `out_data`: The image output address. The input image must be in YUV420 format.

```
```c
/*********************************************************  ********
**************
* YUV420 image scale interface
```

- in_data `in_data`: The input image that must be in YUV420 format.
- paras The scaling struct.
- out_data The output image that must be in YUV420 format.
  **********/
  OPERATE_RET Tuya_Ipc_Img_Resize(UCHAR_T _in_data, TUYA_IMG_RESIZE_PARA paras, UCHAR_T _out_data);

```
```
```

- The following code describes the parameters of the struct `TUYA_IMG_RESIZE_PARA`.

```
typedef struct _TUYA_IMG_RESIZE_PARA
{
      INT_T srcWidth;      // The width of the input image.
      INT_T srcHeight;     // The height of the input image.
      INT_T dstWidth;      // The width of the output image.
      INT_T dstHeight;     // The height of the output image.
      IMG_TYPE img_type;  // The scaling type.
      IMG_RESIZE_TYPE resize_type;
}TUYA_IMG_RESIZE_PARA;
```

- The following code describes the parameters of the encrypted struct `IMG_RESIZE_TYPE`.

```
typedef enum
{
     LINEAR,  // The scaling process is quick, but the image
        quality
 is poor.
     CUBIC,    CUBIC, // The scaling process is slow, but the
        image
quality is high.
}IMG_RESIZE_TYPE;
```

13 Features of PTZ cameras

The DP 119 is used to start rotating, and DP 116 is used to stop rotating.

```
#define TUYA_DP_PTZ_CONTROL                 119
/* Pan-tilt control is the enum type. 0: up  1: upper right  2: righ
t  3: lower right 4: down  5: lower left 6: left  7: upper left
* In SDK v4.0, the enum values are 0: upper right 1: right 2: lower
right 3: down 4: lower left 5: left 6: upper left 7: up
*/
#define TUYA_DP_PTZ_STOP                    116         /* Stop rotat
ing. Boolean type */
```

The device receives the command from the server and then controls the PTZ camera to rotate to the specified angle. The following code shows the typical data format of DP 119.

```
{"range":["1","2","3","4","5","6","7","0"],"type":"enum"}
```

13.1 General features

- The device receives specified DP values from the server and then controls the camera to rotate to the specified angle. For example, the device starts rotating with DP 119 and stops rotating with DP 116. The table below lists the DPs for general controls.

DP ID	Description
119	The pan-tilt motor rotates the camera.
	0: upper right
	1: right
	2: lower right
	3: down
	4: lower left
	5: left

DP ID	Description
	6: upper left
	7: up
116	The pan-tilt motor stops rotating the camera. This DP is of the boolean type.
161	Enable or disable motion tracking. This DP is of the boolean type.
	`true`: enable
	`false`: disable
178	Add or delete preset points. This DP is of the string type. The string differs depending on operation types.
	`type 1`: add
	`type 2`: delete

- You can implement the control logic of rotation in `handle_DP_PTZ_CONTROL` and `handle_DP_PTZ_STOP` respectively.

```
STATIC VOID handle_DP_PTZ_CONTROL(IN TY_OBJ_DP_S *p_obj_dp)
{
  if( (p_obj_dp == NULL) || (p_obj_dp->type != PROP_ENUM) )
  {
      printf("Error!! type invalid %d \r\n", p_obj_dp->type);
      return;
  }
        // DP 119 format: {"range
            ":["1","2","3","4","5","6","7","0"],"t
ype":"enum"}
  UINT_T dp_directions[8] = {1,2,3,4,5,6,7,0};
  UINT_T direction = dp_directions[p_obj_dp->value.dp_enum];
}
```

```
#ifdef TUYA_DP_PTZ_STOP
STATIC VOID handle_DP_PTZ_STOP(IN TY_OBJ_DP_S *p_obj_dp)
{
  IPC_APP_ptz_stop_move();
  respone_dp_bool(TUYA_DP_PTZ_STOP, TRUE);
}
#endif
```

Development processes

1. Define DPs in `tuya_ipc_dp_utils.h` to enable the PTZ feature.

```
#define TUYA_DP_PTZ_CONTROL                 119         /* Start
    rota
ting. Enum type */
#define TUYA_DP_PTZ_STOP                    116         /* Stop
    rotat
ing. Boolean type */
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_PTZ_CONTROL
     {TUYA_DP_PTZ_CONTROL,          handle_DP_PTZ_CONTROL},
#endif
#ifdef TUYA_DP_PTZ_STOP
     {TUYA_DP_PTZ_STOP,             handle_DP_PTZ_STOP},
#endif
```

`handle_DP_PTZ_CONTROL` and `handle_DP_PTZ_STOP` correspond to DP 119 and DP 116. These two functions are intended to be implemented by you.

```
#ifdef TUYA_DP_PTZ_CONTROL
STATIC VOID handle_DP_PTZ_CONTROL(IN TY_OBJ_DP_S *p_obj_dp)
{
if( (p_obj_dp == NULL) || (p_obj_dp->type != PROP_ENUM) )
{
 printf("Error!! type invalid %d \r\n", p_obj_dp->type);
 return;
}
// DP 119 format: {"range":["1","2","3","4","5","6","7","0"],"type
    ":
"enum"}
UINT_T dp_directions[8] = {1,2,3,4,5,6,7,0};
UINT_T direction = dp_directions[p_obj_dp->value.dp_enum];
CHAR_T tmp_str[2] = {0};
snprintf(tmp_str,2,"%d",direction);
IPC_APP_ptz_start_move(tmp_str);
respone_dp_enum(TUYA_DP_PTZ_CONTROL,tmp_str);
}
#endif
```

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and deter-

mine whether to trigger the feature function of this DP. IPC_APP_handle_dp_cmd_objs has been called in TUYA_IPC_SDK_START.

13.2 Preset points

The DP 178 is for preset points.

```
#define TUYA_DP_PRESET_SET                    178  /* Add or delete p
reset points. This DP is of the string type. The string differs depe
nding on operation types. `type 1`: add. `type 2`: delete.  */
```

13.3 Implement functions

- To add a preset point, the device must report the non-cruise status of DP 179.
- When the device receives DP data from the server, it adds the function tuya_ipc_preset_add.

```
/**
* \fn OPERATE_RET tuya_ipc_preset_add(S_PRESET_POSITION*
    preset_pos)
  * \brief  Add a preset point.
  * \param[in]  The specified preset point.
* \return OPERATE_RET
*/
OPERATE_RET tuya_ipc_preset_add(S_PRESET_POSITION* preset_pos);
```

- Struct S_PRESET_POSITION

  :::info

  - id[32]: The ID value from the server. You can leave it as is.
  - name[32]: The name of an added preset point, which can be stored.
  - mpId: Specify a serial number for an added preset point, starting with one.
  - ptz: Specify the pan and tilt coordinates of an added preset point. For fixed-focus cameras, set the coordinate of zoom to 0.

    :::

```
typedef struct
{
     CHAR_T id[32];        // The server ID.
     CHAR_T name[32];       // The name of a preset point.
     INT_T mpId;            // The index ID.
     S_PRESET_PTZ ptz;    // The position of a preset point.
} S_PRESET_POSITION;
typedef struct
{
      INT_T pan;        // Landscape
      INT_T tilt;       // Portrait
      INT_T zoom;       // Set this value to 0 for fixed-focus
     cameras.
}S_PRESET_PTZ;
```

- Call `tuya_ipc_preset_add_pic` to pass in the address and size of an image.

```
/*
\fn OPERATE_RET tuya_ipc_preset_add_pic(CHAR_T *addr, UINT_T size)
  \brief  Upload the snapshot of the current preset point.
  \param[in] addr/size: The address and size of the snapshot.
\return OPERATE_RET
*/
OPERATE_RET tuya_ipc_preset_add_pic(CHAR_T *addr, UINT_T size);
```

- After the device is powered on, call `tuya_ipc_preset_get` to synchronize the existing preset points with the server.

```
/*
\fn OPERATE_RET tuya_ipc_preset_get(S_PRESET_CFG *preset_cfg)
  \brief  Get all the preset points stored on the server.
\param[in out] preset_cfg
\return OPERATE_RET
*/
OPERATE_RET tuya_ipc_preset_get(S_PRESET_CFG *preset_cfg);
```

- The server sends a command to delete a preset point. After executing the command, the device calls `tuya_ipc_preset_del` to return the ID contained in the command.

```
  // Delete preset points. The error alternates between 0 and 1.
if(tmp == 0)
{
 tmp = 1;
}
else if(tmp == 1)
{
 tmp = 0;
}
tuya_ipc_preset_del(devId->valuestring);
snprintf(respond_del,128,"{\"type\":%d,\"data\":{\"error\":%d}}",
   typ
e->valueint,tmp);
```

13.4 Development processes

1. Define DPs in `tuya_ipc_dp_utils.h` to enable the preset point feature.

```
#define TUYA_DP_PRESET_SET                  178
/* Add or delete preset points. This DP is of the string type. The
    s
tring differs depending on operation types. `type 1`: add  `type
   2`:
 delete */
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_PRESET_SET
 {TUYA_DP_PRESET_SET,              handle_DP_SET_PRESET},
#endif
```

3. In `IPC_APP_set_preset` of `handle_DP_SET_PRESET`, determine the operation on a preset point, such as adding, deleting, and calling. Then, call `tuya_ipc_preset_add` to add a preset point, `tuya_ipc_preset_add_pic` to upload the image of a preset point, `tuya_ipc_preset_del` to delete a preset point, and `tuya_ipc_preset_go` to call a preset point. Sample code

   > Before you call `tuya_ipc_preset_add`, you must get the current position of the PTZ.

```
/*1: add a preset point. 2: delete a preset point. 3: call a
   preset
point. */
if(type->valueint == 1)
{
 tuya_ipc_preset_add(data);
}
else if(type->valueint == 2)
{
 tuya_ipc_preset_del(data);
}
else if(type->valueint == 3)
{
 tuya_ipc_preset_go(data);
}
```

4. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

14 Log reporting

- Before the device starts an application, it must determine whether the flag information exists in the SD card. If the flag exists, the output log of the application is redirected to the SD card. Otherwise, the output log is redirected to the directory `/tmp/log.txt`.
  :::info
  - When you use Tuya' s SDK to develop devices, it is forbidden to connect to third-party or private servers due to the risk of non-compliance with local regulations.
  - To trigger the device to report logs to the cloud, user permission is required first.
  - The DNS on the device must be obtained automatically to avoid networking exceptions.
    :::

- Call `tuya_ipc_set_log_attr` to control the log output. The default level of the log output is four. The smaller the value is, the less the log is output. The trace level is five. The following code shows the functions that are called.

```
1  OPERATE_RET tuya_ipc_sdk_start(IN CONST TUYA_IPC_SDK_RUN_VAR_S *
       pRu
2  nInfo)
3  {
4    //...
5    s_ipc_sdk_run_handler.sdk_run_info = *pRunInfo;
6      //TODO loglevel
7      tuya_ipc_set_log_attr(pRunInfo->debug_info.log_level,NULL);
8            //...
9        }
```

To enable the SDK to send the log file of the MCU to the cloud, you can register the callback `cloud_operation_set_log_path_cb`. Make sure to set the log path and file size.

```
typedef VOID (*GW_APP_LOG_PATH_CB)(OUT CHAR_T *path, IN CONST
    INT_T
len);
VOID cloud_operation_set_log_path_cb(GW_APP_LOG_PATH_CB cb)
{
    s_log_cb = cb;
}
```

15 Doorbell feature

This section describes how to develop features of a doorbell, including push notification of doorbell press alerts, video talk, and video messages.

15.0.1 Implement functions

- Call `TUYA_APP_Enable_DOORBELL` to initialize the doorbell.
- Call `ty_timer_create` to create a timer.

```
  // You need to implement the timer interface according to your chi
p platform.
static int ty_timer_create(TIMER_CB cb, timer_t *p_timer_id)
{
  struct sigevent evp;
  memset(&evp, 0, sizeof(struct sigevent));
  evp.sigev_notify = SIGEV_THREAD;
  evp.sigev_notify_function = cb;
  if(timer_create(CLOCK_REALTIME, &evp, p_timer_id) == -1) {
      printf(" fail to timer create\n");
      return -1;
  }
  return 0;
}
```

- Call `get_snapshot` on a doorbell press event to capture images used for push notifications.

```
  // You need to implement the image capture function according to y
our chip platform.
void get_snapshot(char *snap_addr, int *snap_size)
{
  //we use file to simulate
  char snapfile[128];
  *snap_size = 0;
  extern char s_raw_path[];
  printf("get one motion snapshot\n");
}
```

- After the image capture, call `tuya_ipc_door_bell_press` to send the image to the Tuya Developer Platform and the mobile app.

```
/**
* \fn OPERATE_RET tuya_ipc_door_bell_press
  * \brief  Send a message of a doorbell press event to the Tuya Dev
eloper Platform and the mobile app.
  * \param[in] doorbell_type: The doorbell type. `DOORBELL_NORMAL` i
ndicates the battery-powered doorbell that pushes images. `DOORBELL_
AC` indicates the wired doorbell that pushes images via P2P.
  * \param[in] snap_buffer: The address of the snapshot.
  * \param[in] snap_size: The size of the snapshot.
  * \param[in] type: The image format, JPEG or PNG.
* \return OPERATE_RET
*/
OPERATE_RET tuya_ipc_door_bell_press(IN CONST DOORBELL_TYPE_E doorbe
ll_type, IN CONST CHAR_T *snap_buffer, IN CONST UINT_T snap_size, IN
 CONST NOTIFICATION_CONTENT_TYPE_E type);
```

Call `tuya_ipc_leave_video_msg` to enable leaving a video message.

```
/**
* \fn OPERATE_RET tuya_ipc_leave_video_msg
  * \brief  Leave a video message.
  * \param[in] extra_data  The buffer address of the image.
  * \param[in] data_len  The buffer size.
* \return
*/
OPERATE_RET tuya_ipc_leave_video_msg(CHAR_T *extra_data, INT_T data_
len);
```

15.0.2 Development processes

1. Enable the doorbell feature and set parameters.

```
ipc_sdk_run_var.video_msg_info.enable = 1;      // 1: enable  0: disa
ble
ipc_sdk_run_var.video_msg_info.type = MSG_BOTH;     // The supported m
essage type. See the struct `MESSAGE_E`.
ipc_sdk_run_var.video_msg_info.msg_duration = 10;// The maximum leng
th of a video message, in seconds.
```

2. Call `TUYA_IPC_SDK_START` to run the SDK. The SDK will call `tuya_ipc_video_msg_init` to initialize the doorbell feature.
3. `doorbell_handler` will be called when the doorbell is pressed.
4. `doorbell_handler` is used to determine the doorbell status. If the doorbell is in `DOORBELL_LISTEN`, the device calls `get_snapshot` to capture an image and calls `tuya_ipc_door_bell_press` to send the doorbell press event to the Tuya Developer Platform and the mobile app.
5. If the doorbell is in `DOORBELL_RECORD`, the device calls `tuya_ipc_leave_video_msg` to start leaving a video message.

16 Sleep mode and wake-up

16.1 Implement functions

- The DP 149 is for the low-power mode.

  DP 149 has two valid values. `0` indicates the device enters sleep mode and `1` indicates the device is woken up.

```
#define TUYA_DP_DOOR_SLEEP          "149"    /* A DP of boolean type
to enable or disable sleep mode. `true`: The device is in sleep mod
e. `false`: The device is woken up.  */
```

- Call `tuya_ipc_low_power_server_get` to get the server ID and port.

```
/**
* \fn OPERATE_RET tuya_ipc_low_power_server_addr_get
  * \brief  Get the information of the low-power server.
  * \return  `OPRT_OK` indicates the IP and port are obtained. Other
 values indicate failure.
*/
OPERATE_RET tuya_ipc_low_power_server_get(OUT UINT_T *ip,OUT UINT_T
*port);
```

- Call `tuya_ipc_device_id_get` to get the device ID.

```
/**
* \fn OPERATE_RET tuya_ipc_device_id_get
  * \brief  Get the device ID.
  * \return  `OPRT_OK` indicates the device ID is obtained. Other va
lues indicate failure.
*/
OPERATE_RET tuya_ipc_device_id_get(IN OUT CHAR_T *devid, IN OUT INT_
T * id_len);
```

- Call `tuya_ipc_local_key_get` to get the local key.

```
/**
* \fn OPERATE_RET tuya_ipc_local_key_get
  * \brief   Get the signature key from the IPC SDK.
* \ make destKeyBuf len >=17;
  * \return  `OPRT_OK` indicates the signature key is obtained.
*/
OPERATE_RET tuya_ipc_local_key_get(OUT CHAR_T * destKeybuf,OUT UINT_
T * len);
```

- Call `tuya_ipc_low_power_server_connect` to connect to the server.

```
/**
* \fn OPERATE_RET tuya_ipc_lowpower_server_connect
  * \brief  Connect to the lower-power server.
  * \[in] serverIp: The server ID.
  * \[in] port: The server port.
  * \[in] pdevId: The device ID.
  * \[in] idLen: The length of the device ID.
  * \[in] pkey: The local key.
  * \[in] keyLen: The length of the local key.
* \return OPERATE_RET 0:success.other:failed
*/
OPERATE_RET tuya_ipc_low_power_server_connect(UNW_IP_ADDR_T serverIp
,INT_T port,CHAR_T* pdevId, INT_T idLen, CHAR_T* pkey, INT_T keyLen)
;
```

- Call `tuya_ipc_low_power_socket_fd_get` to get the socket file descriptor (FD).

```
/**
* \fn OPERATE_RET tuya_ipc_low_power_socket_fd_get
  * \brief  Get the the TCP handler of the low-power mode.
* \return OPERATE_RET 0:success.other:failed
*/
OPERATE_RET tuya_ipc_low_power_socket_fd_get();
```

- Call `tuya_ipc_low_power_wakeup_data_get` to get the data for waking up the device from the mobile app.

```
/**
* \fn OPERATE_RET tuya_ipc_lowpower_server_connect
  * \brief  Get the data for waking up the device.
* \return OPERATE_RET 0:success.other:failed
*/
OPERATE_RET tuya_ipc_low_power_wakeup_data_get(OUT CHAR_T* pdata,
    OU
T UINT_T* plen);
```

- Call `tuya_ipc_low_power_heart_beat_get` to get the heartbeat.

```
/**
* \fn OPERATE_RET tuya_ipc_lowpower_server_connect
  * \brief  Get the heartbeat.
* \return OPERATE_RET 0:success.other:failed
*/
OPERATE_RET tuya_ipc_low_power_heart_beat_get(OUT CHAR_T * pdata,
    OUT
 UINT_T *plen);
```

16.2 Development processes

1. To enable the low-power mode, call `tuya_ipc_low_power_server_get` to get the server ID and port.
2. Call `tuya_ipc_device_id_get` to get the device ID.
3. Call `tuya_ipc_local_key_get` to get the local key.
4. With the above obtained information, the device calls `TUYA_APP_LOW_POWER_START` to enter low-power mode.

```
OPERATE_RET TUYA_APP_LOW_POWER_START(char * devbuf,char *keybuf,
    int
ip,int port)
```

5. Call `tuya_ipc_low_power_server_connect` in `TUYA_APP_LOW_POWER_START`to connect to the server.
6. Call `tuya_ipc_low_power_socket_fd_get` in `TUYA_APP_LOW_POWER_START` to get the socket FD used to send heartbeats.
7. Call `tuya_ipc_low_power_wakeup_data_get` in `TUYA_APP_LOW_POWER_START` to get the data for waking up the device from the mobile app.

8. Create a loop to keep sending heartbeats to notify the server of the device state and receiving the wake-up data.

```
send(low_power_socket, heart_beat, heart_beat_len, 0);// Send heartb
eats.
```

9. When the module receives commands that require the device to be woken up, startup operations should be executed accordingly.

> The startup operation is intended to be implemented by you.

```
if (FD_ISSET(low_power_socket, &rfds))
{
 bzero(recBuf, MAXBUF+1);
 printf("============recv data==============\n");
 len = recv(low_power_socket, recBuf, MAXBUF, 0);
 if (len > 0)
 {
     printf("Successfully received the message: is {");
     for(i=0;i<len;i++)
         printf("0x%02x ",recBuf[i]);
     printf("}\n");
     if(strncmp(recBuf,wakeData,wake_data_len)==0)
     {
         // Start the SDK, which is intended to be implemented by you.
         printf("recve data is wake up\n");
     }
 }
}
```

16.3 Startup optimization

- The MQTT connection callback `IPC_APP_Get_Net_Status_cb` can use `STAT_MQTT_ONLINE` to determine whether MQTT is connected. If MQTT is connected, `IPC_APP_Get_Net_Status_cb` will proceed with the subsequent operations.
- After the Wi-Fi network is connected to the internet, a separate thread can be started for P2P initialization to optimize device access to the internet.
- After the device successfully detects the SD card, the initialization of local recording can be started. The SDK can automatically detect and fix the data

with incorrect timestamps.

- When the device uses the local storage of events, after calling `tuya_ipc_ss_stop_event`, the device waits for two to three seconds until the recording data has been saved to the SD card and then enters sleep mode.
- During local storage of events, after the device calls `tuya_ipc_ss_stop_event`, it calls `tuya_ipc_ss_get_status` to get the status of the local recording. When the status is `E_STORAGE_STOP`, the device can then enter sleep mode. `tuya_ipc_ss_get_status` can be used only to get status but not to set status.
- When cloud storage is used, the device calls `tuya_ipc_cloud_storage_get_event_status_by_id` after calling `tuya_ipc_cloud_storage_event_delete` to query the status of data uploading. The device can enter sleep mode if the returned status is not `EVENT_ ONGOING` or `EVENT_READY`. Otherwise, data loss might occur.

```
typedef enum
{
  EVENT_NONE, // No cloud storage event occurred, or data is success
fully uploaded to the server.
  EVENT_ONGOING, // Cloud storage event is in progress.
  EVENT_READY,   // The critical point when the cloud storage event
occurs, that is, when `tuya_ipc_cloud_storage_event_add` has just be
en called.
  EVENT_INVALID  // Cloud storage fails to be initialized.
}EVENT_STATUS_E;
```

- Cloud storage can call `tuya_ipc_cloud_storage_set_pre_record_time` to set the pre-recording time to start the video recording a few seconds before the event occurs.
- Increase the priority of threads of P2P and local storage. This might expose a risk to other threads. Make sure that the basic functionality of other threads on the device will not be affected.

17 OTA update feature

This section describes how to implement the OTA update, a popular method to update the firmware. When the SDK detects firmware updates, it will notify the device of OTA updates through a callback.

17.1 Development processes

- The OTA callback `IPC_APP_Upgrade_Inform_cb()` passes in the update download URL and the size of the update file.
- To release memory, call `tuya_ipc_ss_set_write_mode()` to disable local recording. And then, call `tuya_ipc_ss_uninit()` and `tuya_ipc_tranfser_close()`. The `uninit` and `close` functions are used for deinitialization of the local storage and P2P feature to release memory as much as possible.

  Release memory before `tuya_ipc_upgrade_sdk` is called.

```
1  VOID IPC_APP_Upgrade_Inform_cb(IN CONST FW_UG_S *fw)
2  {
3    PR_DEBUG("Rev Upgrade Info");
4    PR_DEBUG("fw->fw_url:%s", fw->fw_url);
5    PR_DEBUG("fw->sw_ver:%s", fw->sw_ver);
6    PR_DEBUG("fw->file_size:%u", fw->file_size);
7    FILE *p_upgrade_fd = fopen(s_mgr_info.upgrade_file_path, "w+b");
8  // Implement releasing memory.
9    tuya_ipc_upgrade_sdk(fw, __IPC_APP_get_file_data_cb,
       __IPC_APP_upg
10 rade_notify_cb, p_upgrade_fd);
11 }
```

  The first callback `IPC_APP_get_file_data_cb` in `tuya_ipc_upgrade_sdk` is used to get the sharding data. This function has input and output parameters. The input parameters can be used for your purpose. After you get the output parameters, you need to return whether the data has been written. Return `0` on successful writing.

  The second callback `IPC_APP_upgrade_notify_cb` is used to display the update progress.

- Replace the old firmware with the new one in `IPC_APP_upgrade_notify_cb`. For

more information about the reference implementation, see tuya_ipc_sdk_upgrade_demo.c.

After the firmware is downloaded, you need to implement firmware replacement and restart the device in the specified function.

Before replacement, back up the DB file. After replacement, compare the MD5 hash of the DB file before and after replacement. If the MD5 hash is identical, restart the device. Otherwise, replace the current DB file with the backup DB file and restart the device.

```
VOID __IPC_APP_upgrade_notify_cb(IN CONST FW_UG_S *fw, IN CONST INT_
T download_result, IN PVOID_T pri_data)
```

- If the OTA update fails, add the restart operation in IPC_APP_upgrade_notify_cb.

  Deinitialization is not reversible. If the OTA update fails, the device must be restarted.

17.2 Custom OTA update progress

- The device downloads the updates through a URL and calls tuya_ipc_upgrade_progress_report to report download progress.

  The progress value reported is recommended to be less than 99%.

```
/*
\fn OPERATE_RET tuya_ipc_upgrade_progress_report
  \brief  Sends the update progress to the cloud and the mobile app.
  \param[in] percent: the update progress in percentage, with valid
value from 0 to 100.
\return SUCCESS - OPERATE_RET , FAIL - COMM ERROR
*/
OPERATE_RET tuya_ipc_upgrade_progress_report(IN UINT_T percent);
```

- When the firmware is being updated, the progress displayed on the app will pause at 98%. At this time, the firmware has been downloaded, and the app is

waiting for the device to be restarted and send the new firmware version number. After receiving the new version number, the app will display a **successful update**.

- Generally, the app waits one minute for the device to replace the firmware and report the new version number. If the waiting time of your device exceeds one minute, you can contact the project manager to configure the PID in the backend.

18 Cloud storage feature

The SDK library has integrated cloud storage. After the cloud storage service is subscribed, the SDK will send the media in the ring buffer to the cloud for storage. The retention time depends on the subscribed cloud storage service.

- The SDK uses software encryption for data by default. The encryption API `OpensslAES_CBC128_encrypt` uses the AES in CBC mode. If software encryption puts a large load on the MCU, you can use hardware encryption and the SoC encryption channel with PKCS#5 and PKCS#7 padding.

Padding type	Description
Zero padding	In the encrypted mode, some blocks do not need to be padded. Or the plain texts are integer multiples of the block length.
ISO10126	If the data with 64-bit block size is `FF FF FF FF FF FF FF FF FF`, the padded data is `FF FF FF FF FF FF FF FF FF 7D 2A 75 EF F8 EF 07`.
PKCS#5	If the data with 64-bit block size is `FF FF FF FF FF FF FF FF FF`, the padded data is `FF FF FF FF FF FF FF FF FF 07 07 07 07 07 07 07`.
	PKCS#5 padding is defined for 8-byte block sizes, which will be padded to 16 bytes. PKCS#7 padding works for any block size from 1 to 255 bytes.
OAEP/PKCS#1	It provides the basic definitions of and recommendations for implementing the RSA algorithm for public-key cryptography.

18.1 Procedure

- 24-hour rolling cloud storage

  1. Specify `ipc_sdk_run_var.cloud_storage_info.enable = 1`.

     For the details of `ipc_sdk_run_var`, see the struct `TUYA_IPC_SDK_RUN_VAR_S`. `enable=1` indicates cloud storage is enabled. `enable=0` indicates cloud storage is disabled.

  2. Call `AES_CBC_init`, `AES_CBC_encrypt`, or `AES_CBC_destory` to initialize the encryption method as needed.

```
ipc_sdk_run_var.aes_hw_info.aes_fun.init = AES_CBC_init;
ipc_sdk_run_var.aes_hw_info.aes_fun.encrypt =AES_CBC_encrypt;
ipc_sdk_run_var.aes_hw_info.aes_fun.destory = AES_CBC_destory;
```

3. Call `TUYA_IPC_SDK_START`. In this function, the `tuya_ipc_sdk_mqtt_online_proc` thread calls `tuya_ipc_cloud_storage_init` that has integrated with cloud storage features.

   For more information about the parameters of `media_setting`, see the struct `IPC_MEDIA_INFO_S;`.

```
/**
* \fn OPERATE_RET tuya_ipc_cloud_storage_init
   * \brief   Initialize the cloud storage and allocate memory
      .
   * aes_func  The encryption method.
   * \param[in]  Specify the required memory and storage.
* \return OPERATE_RET
*/
OPERATE_RET tuya_ipc_cloud_storage_init(IN IPC_MEDIA_INFO_S *
   media_s
etting, IN AES_HW_CBC_FUNC *aes_func);
```

- Event-based cloud storage

1. When the device detects an event that needs to be reported, it calls `tuya_ipc_start_storage` to store this event.

```
/**
* \fn OPERATE_RET tuya_ipc_start_storage
   * \brief Start storage.
   * \param[in] storage_type: The storage type. `
      E_ALARM_SD_STORAGE` i
ndicates local storage. `E_ALARM_CLOUD_STORAGE` indicates
   cloud stor
age.
   * \return If storage is started, `OPRT_OK` is returned.
      Other retur
ned values indicate failure.
*/
OPERATE_RET tuya_ipc_start_storage(INT_T storage_type);
```

2. The cloud storage thread in the SDK will automatically send the new event

to the cloud.

3. When the event ends, the device calls `tuya_ipc_stop_storage` to stop storage. Sample code

```
/**
* \fn OPERATE_RET tuya_ipc_stop_storage
   * \brief  Stop storage.
   * \param[in] storage_type: The storage type. `
      E_ALARM_SD_STORAGE` i
ndicates local storage. `E_ALARM_CLOUD_STORAGE` indicates
   cloud stor
age.
   * \return If storage is stopped, `OPRT_OK` is returned.
      Other retur
ned values indicate failure.
*/
OPERATE_RET tuya_ipc_stop_storage(INT_T storage_type);
```

19 Remove and reset devices

This section describes how to remove devices from the app and reset devices by button press.

19.1 Remove devices

- When users tap **Remove Device** on the mobile app, the SDK will clear the pairing information in the DB file and execute the callback `IPC_APP_Reset_System_CB` to restart the device.

  In this function, you need to implement device restart and DP file reset. Do not delete the DB file.

```
VOID IPC_APP_Reset_System_CB(GW_RESET_TYPE_E type)
{
  printf("reset ipc success. please restart the ipc %d\n", type);
  IPC_APP_Notify_LED_Sound_Status_CB(IPC_RESET_SUCCESS);
    //TODO
  /* Developers need to restart IPC operations */
}
```

19.2 Reset devices

- When the device detects that the reset button is pressed, three DB files `tuya_user.db_bak`, `tuya_user.db`, and `tuya_enckey.db` will be deleted.
- The DP data files are reset, and the device is restarted.

  For more information, see IPC Specification for Tone and Light.

20 Access point (AP) mode on/off

This section describes how to implement turning on or off the (AP) mode of the device by using the mobile app.

The device runs in station mode on successful pairing. After the device switches to the AP mode, the mobile app can connect to the hotspot broadcast by the device. This allows users to preview videos and set up the device without using additional internet access.

- DP 231 is for AP mode query.

```
#define TUYA_DP_AP_MODE                 231    /* The mobile app sends a
 NULL query. The device returns { "is_ap": %d, ap_ssid: "%s", passwo
rd:"%s" } */
```

The device returns the current Wi-Fi mode to the mobile app.

  - `is_ap` indicates whether the device runs in AP mode.
    - `1`: in AP mode.
    - `0`: not in AP mode.
  - `app_ssid` indicates the hotspot's SSID in AP mode.
  - `password` indicates the password of the hotspot, which can be empty.

- DP 232 is for AP mode on/off.

```
#define TUYA_DP_AP_SWITCH               232    /*  The mobile app sends
data: { "ap_enable" : %d, "ap_ssid" : "%s", "ap_pwd" : "%s" }. The d
evice returns data: { "ap_enable" : 1, "errcode" : 0 } */
```

After the AP mode is turned off, the device will automatically enter station mode one minute later and connect to the Wi-Fi network.

  - `ap_enable` indicates turning on or off AP mode.
    - `1`: turn on.
    - `0`: turn off.
  - `errorcode`: error code.

  - `0`: success.
  - Other values: failure.

- DP 233 is for time sync in AP mode.

```
#define TUYA_DP_AP_TIME_SYNC          233     /* The mobile app sends d
ata: { "%s" }*/
```

A device in AP mode cannot get the time from the internet. Therefore, it uses the time from the mobile app to set the time. A Unix timestamp in string: `{ "1629808340"}`

- DP 234 is for time zone sync in AP mode.

```
#define TUYA_DP_AP_TIME_ZONE          234      /* The mobile app sends
data: { "%s" }*/
```

The device sets the time zone according to the DP data from the mobile app. The format is in string format, such as `{ "+8"}`.

20.1 Implement functions

- `tuya_adapter_wifi_ap_start` is used to turn on AP mode, which is intended to be implemented by you.

```
OPERATE_RET tuya_adapter_wifi_ap_start(CONST WF_AP_CFG_IF_S *cfg);
```

- `WF_AP_CFG_IF_S` struct

```
/* tuya sdk definition of ap config info */
typedef struct {
    unsigned char ssid[WIFI_SSID_LEN+1];        ///< ssid
    unsigned char s_len;                        ///< len of ssid
    unsigned char passwd[WIFI_PASSWD_LEN+1];    ///< passwd
    unsigned char p_len;                        ///< len of passwd
    unsigned char chan;                         ///< channel.
        default
:6
    WF_AP_AUTH_MODE_E md;                       ///< encryption
        type
    unsigned char ssid_hidden;                  ///< ssid hidden
        def
ault:0
    unsigned char max_conn;                     ///< max sta
        connect
nums default:3
    unsigned short ms_interval;                 ///< broadcast
        interv
al default:100
    NW_IP_S ip;                                 ///< ip info for ap
         m
ode
}WF_AP_CFG_IF_S;
/* tuya sdk definition of wifi encryption type */
typedef enum
{
    WAAM_OPEN = 0,      ///< Open
    WAAM_WEP,           ///< WEP
    WAAM_WPA_PSK,       // WPA-PSK
    WAAM_WPA2_PSK,      // WPA2-PSK
    WAAM_WPA_WPA2_PSK,  // WPA/WPA2
    WAAM_UNKNOWN,       // Unknown
}WF_AP_AUTH_MODE_E;
```

- `tuya_adapter_wifi_ap_stop` is used to turn off AP mode, which is intended to be implemented by you.

```
OPERATE_RET tuya_adapter_wifi_ap_stop(VOID);
```

20.2 Development processes

- AP mode query

  1. Define the DP in `tuya_ipc_dp_utils.h` to enable the AP mode query feature.

```
#define TUYA_DP_AP_MODE 231
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_AP_MODE
    {TUYA_DP_AP_MODE,                handle_DP_AP_MODE},
#endif
```

`handle_DP_AP_MODE` corresponds to the DP 231.

```
#ifdef TUYA_DP_AP_MODE
  STATIC VOID handle_DP_AP_MODE(IN TY_OBJ_DP_S *p_dp_json)
  {
      if(p_dp_json == NULL )
      {
          printf("Error!! type invalid\r\n");
          return;
      }
      respone_dp_str(TUYA_DP_AP_MODE, IPC_APP_get_ap_mode());
  }
  #endif
```

`IPC_APP_get_ap_mode` assembles a response according to the current Wi-Fi mode: `{ "is_ap": %d, ap_ssid: "%s", password:"%s"}`.

Call `tuya_adapter_wifi_get_work_mode` to get the current Wi-Fi mode. `is_ap` indicates whether the device runs in AP mode.

Call `tuya_adapter_wifi_get_mac` to get the MAC address of the Wi-Fi. Assemble an SSID value in the format `TUYA_IPC-xxxx` as the return value of the parameter `ap_ssid`. You can also customize the format of SSID.

Call `tuya_app_read_STR` to get the configuration items of `tuya_ap_passwd` as the password of the hotspot in AP mode. You can also customize the password.

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

- Turn on or off AP mode

1. Define DPs in `tuya_ipc_dp_utils.h` to enable the AP mode on/off control.

```
#define TUYA_DP_AP_SWITCH 232
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_AP_SWITCH
    {TUYA_DP_AP_SWITCH,             handle_DP_AP_SWITCH},
#endif
```

`handle_DP_AP_SWITCH` corresponds to the DP 232.

```
#ifdef TUYA_DP_AP_SWITCH
STATIC VOID handle_DP_AP_SWITCH(IN TY_OBJ_DP_S *p_dp_json)
{
    CHAR_T resp[32] = {0};
    INT_T ap_enable = IPC_APP_set_ap_mode((cJSON *)p_dp_json->value.dp_
str);
    if(ap_enable < 0)
    {
        snprintf(resp, 32, "{\\\"ap_enable\\\":0,\\\"errcode\\\":0}");
    }
    else
    {
        snprintf(resp, 32, "{\\\"ap_enable\\\":%d,\\\"errcode\\\":0}",ap_e
nable);
    }
    respone_dp_str(TUYA_DP_AP_SWITCH, resp);
    respone_dp_str(TUYA_DP_AP_MODE, IPC_APP_update_ap_mode(ap_enable));
 //switch ap mode, need update ap mode value
    if(ap_enable >= 0)
    {
        change_ap_process();
    }
}
#endif
```

`IPC_APP_set_ap_mode` is used to parse the received DP data and assert Wi-Fi mode.

The device needs to respond to DP 231 according to its response to DP 232 to sync the status of DP 231 with the mobile app.

```
respone_dp_str(TUYA_DP_AP_SWITCH, resp);
respone_dp_str(TUYA_DP_AP_MODE, IPC_APP_update_ap_mode(
    ap_enable));
//switch ap mode, need update ap mode value
```

After switching to AP mode, the device will be disconnected from the current network. Therefore, the device must respond to DPs and then switch to AP mode. The switching operation must be completed in a new thread in `change_ap_process`.

```
VOID change_ap_process()
{
    pthread_t ap_change_thread;
    int ret = pthread_create(&ap_change_thread, NULL,
        __ap_change_threa
d, NULL);
    if(ret != 0)
    {
        printf("ap_change_thread ,create fail! ret:%d\n",ret);
        return;
    }
    pthread_detach(ap_change_thread);
}
```

In the demo, the device turns on AP mode in `ap_change_thread`. Before calling `tuya_adapter_wifi_ap_start` to turn on AP mode, the device should call `tuya_devos_netlink_monitor_disable` to turn off monitoring of Wi-Fi connection.

After calling `tuya_adapter_wifi_ap_stop` to turn off AP mode, the device calls `tuya_devos_netlink_monitor_enable` to turn on monitoring of Wi-Fi connection.

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

- Time sync in AP mode

1. Define the DP in `tuya_ipc_dp_utils.h` to enable time sync in AP mode.

```
#define TUYA_DP_AP_TIME_SYNC                233
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_AP_TIME_SYNC
    {TUYA_DP_AP_TIME_SYNC,          handle_DP_AP_TIME_SYNC},
#endif
```

For more information about the reference implementation of `handle_DP_AP_TIME_SYNC`, see the demo.

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

- Time zone sync in AP mode

1. Define the DP in `tuya_ipc_dp_utils.h` to enable time zone sync in AP mode.

```
#define TUYA_DP_AP_TIME_ZONE              234
```

2. Add the following code in `s_dp_table[]` of `tuya_ipc_dp_utils.c`.

```
#ifdef TUYA_DP_AP_TIME_ZONE
    {TUYA_DP_AP_TIME_ZONE,          handle_DP_AP_TIME_ZONE},
#endif
```

For more information about the reference implementation of `handle_DP_AP_TIME_ZONE`, see the demo.

3. Call `IPC_APP_handle_dp_cmd_objs` to process the DP data from the server and determine whether to trigger the feature function of this DP. `IPC_APP_handle_dp_cmd_objs` has been called in `TUYA_IPC_SDK_START`.

- Enter AP mode on startup

1. If you turn on starting device in AP mode, you need to save the AP mode configuration on the device to retain the AP state after shutdown, so that the device can turn on AP mode with the saved configuration on startup.
2. In the demo, with the `tuya_ipc_sdk_mqtt_online_proc`, local storage and P2P are initialized after the internet is connected and time is synced. However, if you turn on starting device in AP mode, this initialization logic is not feasible. You need to implement an additional initialization process to deal

with this situation, such as the reference `tuya_ipc_ap_mode_start_proc` in the demo.

3. Call `TUYA_IPC_SDK_START` in the `main` function of `user_main.c`, call `IPC_APP_get_ap_mode_config` to get the AP mode configuration, and then call `IPC_APP_switch_ap_mode` to turn on the AP mode.

   Since the paired device enters station mode on startup by default, to start a device in AP mode, you need to call `TUYA_IPC_SDK_START` first and then call `IPC_APP_switch_ap_mode` to turn on the AP mode.

21 API adaptations

For more information, see IPC SDK APIs.

22 FAQs