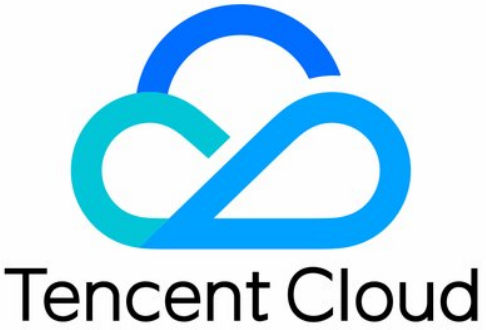
IoT Explorer

Device Development Guide



Copyright Notice

Trademark Notice

Service Notice

This document provides an overview of the as-is details of Tencent Cloud's products and services in their entirety or part. The descriptions of certain products and services may be subject to adjustments from time to time.

The commercial contract concluded by you and Tencent Cloud will provide the specific types of Tencent Cloud products and services you purchase and the service standards. Unless otherwise agreed upon by both parties, Tencent Cloud does not make any explicit or implied commitments or warranties regarding the content of this document.

Contact Us

We are committed to providing personalized pre-sales consultation and technical after-sale support. Don't hesitate to contact us at 4009100100 or 95716 for any inquiries or concerns.

Contents

Device Development Guide Developer Guide

Last updated: 2025-04-27 17:48:43

Tencent Cloud IoT development platform provides device-side SDKs in multiple languages and oriented to various scenarios. To help you use the correct SDK integration and utilize the IoT development platform more efficiently, please read the following content carefully.

Developer Guide

SDK Overview

SDK	Description	Application Scenario	Reference Documentation
qcloud-iot-explorer-sdk-embedded-c	Device-side C Language SDK	Provide adaptation guides for multiple integrations and usages of IoT Explorer for platforms developed based on the C language.	C SDK Usage Reference
qcloud-iot-explorer-5G-sdk-embedded	Device-side C Language SDK 5G	Oriented to development platforms based on the C language, introduce 5G and edge computing features on the basis of IoT Explorer.	C SDK 5G Usage Reference
iot-device-java	Device-side Java Language SDK	Provide adaptation guides for platforms such as Android to access and use IoT Explorer for platforms developed based on the Java language.	• Android SDK Usage Reference • Java SDK Usage Reference
iot-device-python	Device-side Python Language SDK	For platforms developed based on the Python language, support the access of microcontrollers or embedded devices running MicroPython to IoT Explorer.	Python SDK Usage Reference

qcloud-iot-sdk-tencent-at-based	Device-side AT Module SDK	For platforms developed based on the customization of AT Modules by Tencent Cloud, provide an adaptation guide for the access of MCU+ Tencent Cloud customized AT Modules to IoT Explorer.	AT SDK Usage Reference
qcloud-iot-esp-wifi	Device-side ESP8266 SDK	For platforms developed based on ESP8266, provide the access process for Tencent Cloud ESP8266 custom firmware, as well as the adaptation guide for multiple networking protocols such as SoftAp and SmartConfig to access Tencent Lianlian Mini Program.	ESP8266 SDK Usage Reference

Besides using the above SDK to integrate with IoT Explorer, you can also use TencentOS tiny to quickly integrate with IoT Explorer by porting the C SDK, and introduce features such as low-power, low resource occupation, modularization, and security and reliability to your application. For details, see Usage Reference of SDK Based on TencentOS Tiny.

Development Process

The development process is divided into the following three steps:

1. Determine application scenarios, among them the application scenarios include:
   - Directly connected device.
   - Gateway and subdevice.
   - Bluetooth device.
   - Device network configuration.
2. Select the appropriate SDK according to the application scenario.
3. Reference documentation and examples to implement the functionality.

Relevant Guidelines

Directly Connected Device

Direct device access types are divided into **resource-rich type** and **resource-constrained type**. For details, see direct device access type description.

- Resource-rich devices

Development Platform	SDK	Reference Documentation
Linux	qcloud-iot-explorer-sdk-embedded-c	Linux Platform Access Guide
Windows	qcloud-iot-explorer-sdk-embedded-c	Windows Platform Access Guide
Android	iot-device-java	Android Platform Access Guide
Java	iot-device-java	Java Platform Access Guide
FreeRTOS+lwIP	qcloud-iot-explorer-sdk-embedded-c	FreeRTOS+lwIP Platform Access Guide
other platforms	qcloud-iot-explorer-sdk-embedded-c	C SDK Porting Access Guide

- Resource-constrained devices

Development Platform	SDK	Reference Documentation
MCU+ customized AT Module (cellular type)	qcloud-iot-sdk-tencent-at-based	MCU+ Custom MQTT AT Module (Cellular Type) Integration Guide
MCU+ customized AT Module (Wi-Fi type)	qcloud-iot-sdk-tencent-at-based	MCU+ Custom MQTT AT Module (Wi-Fi Type) Access Guide
MCU+ General AT Module + FreeRTOS	qcloud-iot-explorer-sdk-embedded-c	MCU+ General TCP AT Module (FreeRTOS) Integration Guide
MCU+ General AT Module + nonOS	qcloud-iot-explorer-sdk-embedded-c	MCU+ General TCP AT Module (nonOS) Access Guide

Bluetooth Device

For details, see Bluetooth device development.

Gateway and Subdevice

For details, see Gateway and Sub-device Development.

Device Network Configuration

Distribution Network Protocol	SDK	Reference Documentation
AirKiss	qcloud-iot-esp-wifi	AirKiss Distribution Network Development
SmartConfig	qcloud-iot-esp-wifi	SmartConfig Distribution Network Development
softAP	qcloud-iot-esp-wifi	softAP Distribution Network Development

Best Practice

Practice Item	Introduction
Wi-Fi Configuration Practice	This practice mainly introduces how to port Tencent Cloud IoT C SDK to Espressif ESP8266 RTOS platform, and provides a runnable Demo. Meanwhile, it introduces how to use Wi-Fi distribution network API at code level, and can combine with Tencent Lianlian Mini Program to perform Wi-Fi distribution network and device binding in SoftAP mode.
MCU+ Custom AT Module Practice	This practice is oriented for device developers who use modules (2G/3G/4G/5G, NB, Wi-Fi, etc.) that support Tencent AT commands to integrate with the Tencent IoT Platform. It provides a porting example of using the Tencent AT-SDK on the mcu side. It shows how to achieve the HAL layer porting in a software and hardware environment based on the STM32F103 mcu and FreeRTOS.
MCU+ General TCP Module (I-Cube) Practice	This practice implements porting examples of STM32+esp8266+FreeRTOS and STM32+esp8266+without RTOS based on STM32 I-Cube.
TencentOS-tiny	This practice realizes the integration with Tencent Cloud IoT Explorer based on TencentOS-tiny, a real-time operating system developed by Tencent for the Internet of Things domain.

Device Identity Authentication Introduction

Last updated: 2025-04-27 17:48:56

Overview

The IoT explorer platform assigns a unique identifier, ProductID, to each created product. Users can customize the Devicename to identify devices. The legitimacy of devices is verified using the product identification + device identification + device certificate/key. When creating a product, users need to select the device authentication method. When devices are connected, they need to report product, equipment information and corresponding key information according to the specified method. Only after authentication can they connect to the IoT explorer platform. Since the resource and security level requirements of devices of different users are all different, the platform provides two authentication methods to meet different use cases.

Provide the following two authentication methods:

- Certificate authentication (device-level): Assign certificates and private keys to each device, use asymmetric encryption for authentication during integration, and users need to burn different configuration information for each device.
- Key authentication (device-level): Assign a device key to each device, authenticate the access using symmetric encryption, and the user needs to burn different configuration information for each device.

The two solutions have their own pros and cons in terms of ease of use, security and device resource requirements. You can comprehensively evaluate and select based on your business scenario. The solution comparison is as follows:

Feature	Certificate Authentication	Key Authentication
Device burning information	Product ID, Device name, Device certificate, Device private key.	ProductId, Devicename, device key.
Whether to pre-create devices	Must.	Required.
Security	High.	Normal.
Use Limits	Can create up to 200,000 devices under a single product.	Create up to 200,000 devices under a single product.

Device Resource Requirements	Relatively high. Need to support X509 certificate parsing.	Lower.

Note:
Only enterprise instances support certificate authentication.

Device Authentication Info Burning Method

Local Burning

Users pre-create devices through the IoT Explorer console or TencentCloud API, and obtain the product ID, device name, and device key (or device certificate, device private key) required for device authentication. During production, burn the configuration information required for device authentication into the Device Flash for long-term storage. This information is used for subsequent device-to-IoT Explorer connections. Only after passing authentication can the device be activated and launched to complete device activation and go online, enabling data interaction with the cloud to fulfill business requirements.

Directions

Need to burn different firmware for each device. There is certain realize cost in production line application, but it has higher security. Recommended for use.

Air Delivery

Devices can obtain device authentication information required for burning through dynamic retrieval. In this method, users only need to enable the device dynamic registration switch in the console to burn the same configuration firmware (ProductID + ProductSecret) for all devices under the same product. Users pre-create devices in the IoT Explorer console or TencentCloud API, and then obtain the configuration information required for device authentication through a dynamic registration request, burn it into the Device Flash for long-term storage, and use it for subsequent devices to initiate connection with the IoT Explorer. Only after passing authentication, complete device activation and go live, it can interact with the cloud for data exchange to realize business requirements.

> **Note:**
> If you need to use this burning method, you need to manually enable the dynamic registration feature of the product on the product details page on the console first.

Operation Process

Introduction to Equipment Communication Protocol Device MQTT Access Protocol MQTT-Based Device Connection over TCP

Last updated: 2025-04-27 17:49:33

MQTT Protocol Description

Currently, the IoT development platform supports MQTT standard protocol access (compatible with v3.1.1). For more information, please see MQTT Version 3.1.1 protocol documentation.

Difference From Standard MQTT

1. Support MQTT messages such as PUB, SUB, PING, PONG, CONNECT, DISCONNECT, UNSUB.
2. cleanSession is supported.
3. Not support will, retain msg.
4. Not support QOS2.

MQTT Channel, Security Level

Support TLSV1, TLSV1.1, and TLSV1.2 protocol versions to establish a secure connection with a high security level.

TOPIC Specification

By default, after a product is created, all devices under the product have the following topic category permissions:

- Subscribe to ${productId}/${deviceName}/control.
- `${productId}/${deviceName}/event` released.
- `${productId}/${deviceName}/data` subscription and publication.
- `$shadow/operation/${productId}/${deviceName}` published. Distinguish by the type inside the package body: update/get, which correspond to the update and retrieval operations of the Device Shadow Document, respectively.

- `$shadow/operation/result/${productId}/${deviceName}` subscription. Distinguish by the type inside the package body: update/get/delta. Type update/get correspond to the results of the update and retrieval operations of the Device Shadow Document, respectively; when users modify the Device Shadow Document through the restAPI, the server will publish messages through this topic, where the type is delta.
- `$ota/report/${productID}/${deviceName}` for publishing. The device reports the version number and the download/upgrade progress to the cloud.
- `$ota/update/${productID}/${deviceName}` subscription. The device receives upgrade messages from the cloud.

MQTT Access

The MQTT protocol supports two methods to integrate with the IoT platform: device certificate and key signature. Based on your own scenario, you only need to choose one method to integrate. The access parameters are as follows:

Access Authentication Method	Connection Domain Name and Port	Connect Message Parameters
Certificate authentication	MQTT server connection address, devices in Guangzhou fill in: ${productId}.iotcloud.tencentdevices.com, here ${productId} is a variable parameter, and the user must enter the product ID automatically generated when creating the product. For example, 1A17RZR3XX.iotcloud.tencentdevices.com; Port: 8883	• KeepAlive: The duration of keep-alive. The value range is 0 – 900 s. If the Internet of Things Platform still does not receive the data of the client after exceeding 1.5 times the KeepAlive duration, the platform will disconnect from the client. • ClientId: ${productId}${deviceName}, a combined string of product ID and device name. • UserName: `${productId}${deviceName};${sdkappid};${connid};${expiry}`. For details, see the username part in the MQTT-based signature authentication access guide below; • PassWord: password (you can assign any value).
Key Authentication	The MQTT server connection address	• KeepAlive: time to keep the connection alive. Value range: 0-900s; • ClientId:${productId}${deviceName};

	matches the certificate authentication; Port: 1883.	• UserName: `${productId}${deviceName};${sdkappid};${connid};${expiry}`. For details, see the username part in the MQTT-based signature authentication access guide below; • PassWord: password. For details, see the access guide based on MQTT signature authentication in the password section below.

Note:
When a device using certificate authentication is connected, the filled-in PassWord part will not be verified. Any value can be filled in the PassWord part during certificate authentication.

Certificate Authentication Device Connectivity Guide

The IoT platform uses TLS encryption to guarantee the security of device-transmitted data. When a certificate device is connected, after obtaining the certificate, key, and CA certificate file of the certificate device, set KeepAlive, ClientId, UserName, PassWord, etc. (Devices connected via the Tencent Cloud Device SDK do not require setting; the SDK can automatically generate based on device information). The device uploads the authentication file to the URL (connection domain name and port) corresponding to the certificate authentication. After passing, send the MqttConnect message to complete the certificate device's TCP-based MQTT access.

Key Authentication Device Connectivity Guide

The IoT platform supports generating digest signatures based on the device key using methods such as HMAC-SHA256 and HMAC-SHA1. The process of accessing the IoT platform through the signature method is as follows:

1. Log in to the IoT Explorer console. You can create products, add devices, and obtain device keys in the console.
2. Generate the username field according to the constraints of IoT Explorer. The format of the username field is as follows:

```
The format of the username field is:
${productId}${deviceName};${sdkappid};${connid};${expiry}
Note: ${} indicates a variable, not a specific concatenation symbol.
```

The meanings of each field are as follows:

- productId: product ID.
- deviceName: device name.
- sdkappid: fixed fill 12010126.
- connid: a random string.
- expiry: indicates the signature validity period, in unix timestamp format, such as 1704363215. expiry should be set to a time far exceeding the actual lifecycle of the device, or by adding a large integer each time the device side gets the current system time. If the value of expiry is less than the current system time, MQTT identity verification will fail.

3. Perform base64 decoding on the device key to obtain the original key raw_key.
4. Use the raw_key generated in step 3 to generate a hash value for the username through the HMAC-SHA1 or HMAC-SHA256 algorithm, abbreviated as Token.
5. Generate the password field according to the constraints of IoT Explorer. The format of the password field is:

```
The format of the password field is as follows:
${token}; hmac signature method
Among them, fill in the hmac signature method field with the digest
algorithm used in step 3. The optional values are hmacsha256 and
hmacsha1.
```

As a reference, Python, Java, Node.js, JavaScript, and C code samples for user-generated signatures are as follows:

Python code:

```
#!/usr/bin/python
# -*- coding: UTF-8 -*-
import base64
import hashlib
import hmac
import random
import string
import time
import sys
# Generate a random string of the specified length
def RandomConnid(length):
    return  ''.join(random.choice(string.ascii_uppercase +
string.digits) for _ in range(length))
# Generate parameters required for IoT platform integration
```

```
def IotHmac(productID, devicename, devicePsk):
    # 1. Generate connid as a random string to facilitate issue
identification in the backend
    connid   = RandomConnid(5)
    # 2. Generate expiration time, which indicates the signature's
expiration time, as a UTF8 string representing the number of seconds
from the epoch January 1, 1970, 00:00:00 UTC to the present
    expiry   = int(time.time()) + 60 * 60
    # 3. Generate the clientid part of MQTT in the format of
${productid}${devicename}
    clientid = "{}{}".format(productID, devicename)
    # 4. Generate the username part of MQTT in the format of
${clientid};${sdkappid};${connid};${expiry}
    username = "{};12010126;{};{}".format(clientid, connid, expiry)
    # 5. Sign the username and generate a token
    secret_key = devicePsk.encode('utf-8')  # convert to bytes
    data_to_sign = username.encode('utf-8')  # convert to bytes
    secret_key = base64.b64decode(secret_key)  # this is still bytes
    token = hmac.new(secret_key, data_to_sign,
digestmod=hashlib.sha256).hexdigest()
    # 6. Generate the password field according to IoT platform rules
    password = "{};{}".format(token, "hmacsha256")
    return {
        "clientid" : clientid,
        "username" : username,
        "password" : password
    }
if __name__ == '__main__':
    print(IotHmac(sys.argv[1], sys.argv[2], sys.argv[3]))
```

Save the above code to IotHmac.py, and execute the following command just. Here, "YOUR_PRODUCTID", "YOUR_DEVICENAME", and "YOUR_PSK" are fill in your actual product ID, device name, and device key of the created device.

```
python3 IotHmac.py "YOUR_PRODUCTID" "YOUR_DEVICENAME" "YOUR_PSK"
```

Java code:

```
package com.tencent.iot.hub.device.java.core.sign;

import org.junit.Test;
```

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import java.util.*;

import static junit.framework.TestCase.fail;
import static org.junit.Assert.assertTrue;

public class SignForMqttTest {

    @Test
    public void testMqttSign() {
        try {

System.out.println(SignForMqttTest("YourProductId","YourDeviceName","Y
ourPsk"));
            assertTrue(true);
        } catch (Exception e) {
            e.printStackTrace();
            fail();
        }
    }

    public static Map<String, String> SignForMqttTest(String
productID, String devicename, String
            devicePsk) throws Exception {
        final Base64.Decoder decoder = Base64.getDecoder();
        //1. Generate connid as a random string to facilitate issue
identification in the backend
        String connid = HMACSHA256.getConnectId(5);
        //2. Generate expiration time, which indicates the signature's
expiration time, as a UTF8 string representing the number of seconds
from the epoch January 1, 1970, 00:00:00 UTC to now
        Long expiry = Calendar.getInstance().getTimeInMillis()/1000 +
600;
        //3. Generate the MQTT clientid part in the format of
${productid}${devicename}
        String clientid = productID+devicename;
        //4. Generate the MQTT username part in the format of
${clientid};${sdkappid};${connid};${expiry}
        String username = clientid+";"+"12010126;"+connid+";"+expiry;
        //5. Sign the username, generate a token, and generate the
password field according to IoT platform rules
```

```
        String password = HMACSHA256.getSignature(username.getBytes(),
decoder.decode(devicePsk)) + ";hmacsha256";
        Map<String,String> map = new HashMap<>();
        map.put("clientid",clientid);
        map.put("username",username);
        map.put("password",password);
        return map;
    }
    public static class HMACSHA256 {
        private static final String HMAC_SHA256 = "HmacSHA256";
        /**
         * Generate signature data
         *
         * @param data The data to be encrypted
         * @param key The key used for encryption
         * @return Generate a hexadecimal encoded string
         */
        public static String getSignature(byte[] data, byte[] key)  {
            try {
                SecretKeySpec signingKey = new SecretKeySpec(key,
HMAC_SHA256);
                Mac mac = Mac.getInstance(HMAC_SHA256);
                mac.init(signingKey);
                byte[] rawHmac = mac.doFinal(data);
                return bytesToHexString(rawHmac);
            }catch (Exception e) {
                e.printStackTrace();
            }
            return null;
        }
        /**
         * Convert byte[] array to a hexadecimal string
         *
         * @param bytes The byte array to be switched
         * @return Converted result
         */
        private static String bytesToHexString(byte[] bytes) {
            StringBuilder sb = new StringBuilder();
            for (int i = 0; i < bytes.length; i++) {
                String hex = Integer.toHexString(0xFF & bytes[i]);
                if (hex.length() == 1) {
                    sb.append('0');
                }
```

```
                sb.append(hex);
            }
            return sb.toString();
        }


        /**
         * Get connection ID (a random alphanumeric string with a
length of 5)
         */
        public static String getConnectId(int length) {
            StringBuffer connectId = new StringBuffer();
            for (int i = 0; i < length; i++) {
                int flag = (int) (Math.random() * Integer.MAX_VALUE) %
3;
                int randNum = (int) (Math.random() *
Integer.MAX_VALUE);
                switch (flag) {
                    case 0:
                        connectId.append((char) (randNum % 26 + 'a'));
                        break;
                    case 1:
                        connectId.append((char) (randNum % 26 + 'A'));
                        break;
                    case 2:
                        connectId.append((char) (randNum % 10 + '0'));
                        break;
                }
            }

            return connectId.toString();
        }
    }

}
```

Nodejs and JavaScript code are:

```
// The following is the introduction method for node. For browsers,
use the corresponding way to introduce the crypto-js library.
const crypto = require('crypto-js')
```

```
// Function to generate a random number
const randomString = (len) => {
    len = len || 32;
    var chars = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
    var maxPos = chars.length;
    var pwd = '';
    for (let i = 0; i < len; i++) {
        pwd += chars.charAt(Math.floor(Math.random() * maxPos));
    }
    return pwd;
}
// product id, device name and device key are required
const productId = 'YOUR_PRODUCTID';
const deviceName = 'YOUR_DEVICENAME';
const devicePsk = 'YOUR_PSK';

// 1. Generate connid as a random string to facilitate issue
identification in the backend
const connid =  randomString(5);
// 2. Generate the expiration time, which indicates the signature's
expiration time, as a UTF8 string representing the number of seconds
from the epoch January 1, 1970, 00:00:00 UTC to now
const expiry = Math.round(new Date().getTime() / 1000) + 3600 * 24;
// 3. Generate the MQTT clientid part in the format of
${productid}${devicename}
const clientId = productId + deviceName;
// 4. Generate the MQTT username part in the format of
${clientid};${sdkappid};${connid};${expiry}
const userName = ${clientId};12010126;${connid};${expiry};
//5. Sign the username, generate a token, and generate the password
field according to IoT platform rules
const rawKey = crypto.enc.Base64.parse(devicePsk);      // Decode the
device key in base64
const token =  crypto.HmacSHA256(userName, rawKey);
const password = token.toString(crypto.enc.Hex) + ";hmacsha256";
console.log(userName:${userName}\npassword:${password});
```

The C code is as follows:

> **Note:**
> If you want more information about the C code, please see project download.

```
#include "limits.h"
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>

#include "HAL_Platform.h"
#include "utils_base64.h"
#include "utils_hmac.h"

/* Max size of base64 encoded PSK = 64, after decode: 64/4*3 = 48*/
#define DECODE_PSK_LENGTH 48

/* MAX valid time when connect to MQTT server. 0: always valid */
/* Use this only if the device has accurate UTC time. Otherwise, set
to 0 */
#define MAX_ACCESS_EXPIRE_TIMEOUT (0)

/* Max size of conn Id  */
#define MAX_CONN_ID_LEN (6)

/* IoT C-SDK APPID */
#define QCLOUD_IOT_DEVICE_SDK_APPID     "21****06"
#define QCLOUD_IOT_DEVICE_SDK_APPID_LEN
(sizeof(QCLOUD_IOT_DEVICE_SDK_APPID) - 1)

static void HexDump(char *pData, uint16_t len)
{
    int i;

    for (i = 0; i < len; i++) {
        HAL_Printf("0x%02.2x ", (unsigned char)pData[i]);
    }
    HAL_Printf("\n");
}

static void get_next_conn_id(char *conn_id)
{
    int i;
    srand((unsigned)HAL_GetTimeMs());
    for (i = 0; i < MAX_CONN_ID_LEN - 1; i++) {
        int flag = rand() % 3;
        switch (flag) {
            case 0:
```

```
                conn_id[i] = (rand() % 26) + 'a';
                break;
            case 1:
                conn_id[i] = (rand() % 26) + 'A';
                break;
            case 2:
                conn_id[i] = (rand() % 10) + '0';
                break;
        }
    }

    conn_id[MAX_CONN_ID_LEN - 1] = '\0';
}

int main(int argc, char **argv)
{
    char *product_id    = NULL;
    char *device_name   = NULL;
    char *device_secret = NULL;

    char *username     = NULL;
    int   username_len = 0;
    char  conn_id[MAX_CONN_ID_LEN];

    char password[51]      = {0};
    char username_sign[41] = {0};

    char   psk_base64decode[DECODE_PSK_LENGTH];
    size_t psk_base64decode_len = 0;

    long cur_timestamp = 0;

    if (argc != 4) {
        HAL_Printf("please ./qcloud-mqtt-sign product_id device_name
device_secret\r\n");
        return -1;
    }

    product_id    = argv[1];
    device_name   = argv[2];
    device_secret = argv[3];

    /* first device_secret base64 decode */
```

```
    qcloud_iot_utils_base64decode((unsigned char *)psk_base64decode,
DECODE_PSK_LENGTH, &psk_base64decode_len,
                                  (unsigned char *)device_secret,
strlen(device_secret));
    HAL_Printf("device_secret base64 decode:");
    HexDump(psk_base64decode, psk_base64decode_len);

    /* second create mqtt username
     * [productdevicename;appid;randomconnid;timestamp] */
    cur_timestamp = HAL_Timer_current_sec() +
MAX_ACCESS_EXPIRE_TIMEOUT / 1000;
    if (cur_timestamp <= 0 || MAX_ACCESS_EXPIRE_TIMEOUT <= 0) {
        cur_timestamp = LONG_MAX;
    }

    // 20 for timestampe length & delimiter
    username_len = strlen(product_id) + strlen(device_name) +
QCLOUD_IOT_DEVICE_SDK_APPID_LEN + MAX_CONN_ID_LEN + 20;
    username     = (char *)HAL_Malloc(username_len);
    if (username == NULL) {
        HAL_Printf("malloc username failed!\r\n");
        return -1;
    }

    get_next_conn_id(conn_id);
    HAL_Snprintf(username, username_len, "%s%s;%s;%s;%ld", product_id,
device_name, QCLOUD_IOT_DEVICE_SDK_APPID,
                 conn_id, cur_timestamp);

    /* third use psk_base64decode hamc_sha1 calc mqtt username sign
crate mqtt
     * password */
    utils_hmac_sha1(username, strlen(username), username_sign,
psk_base64decode, psk_base64decode_len);
    HAL_Printf("username sign: %s\r\n", username_sign);
    HAL_Snprintf(password, 51, "%s;hmacsha1", username_sign);

    HAL_Printf("Client ID: %s%s\r\n", product_id, device_name);
    HAL_Printf("username : %s\r\n", username);
    HAL_Printf("password : %s\r\n", password);

    HAL_Free(username);
```

```
    return 0;
}
```

6. Finally, fill in the parameters generated above into the corresponding MQTT connect message.
7. Enter the clientid into the clientid field of the MQTT protocol.
8. Enter the username into the username field of MQTT.
9. Enter the password into the password field of MQTT and send MqttConnect information to the domain name and port of key authentication to integrate with the IoT platform.

Equipment'S MQTT Access Based on WebSocket

Last updated: 2025-04-27 17:49:49

MQTT-WebSocket Overview

The IoT platform supports MQTT communication based on WebSocket. Devices can use the MQTT protocol to transmit messages on the basis of the WebSocket protocol. Thus causing browser-based applications to achieve data communication with the platform and devices connected to the platform. Meanwhile, WebSocket uses ports 443/80, allowing message transmission to pass through most firewalls.

MQTT-WebSocket Integration

Since both the MQTT-WebSocket protocol and the MQTT-TCP protocol ultimately transmit messages based on MQTT, the access parameters for MQTT are identical for these two protocols. The main difference lies in the protocol and port used to connect to the MQTT platform. Devices with key authentication use WS for access, while devices with certificate authentication use WSS for access, that is, WS+TLS.

Certificate Authentication Device Access Guide

1. Log in to the IoT Explorer console and enter the target product page. In the corresponding device information section, download files such as the certificate and device private key.
2. Connection Domain Name: Devices in Guangzhou need to connect to `${ProductId}.ap-guangzhou.iothub.tencentdevices.com:443`, where ${ProductId} is a variable parameter for product ID.
3. MQTT Connection Parameter Settings:
   The connection parameter settings are consistent with those when using MQTT-TCP access. For details, see the MQTT Access chapter in the Device-Based TCP MQTT Access document.

```
UserName:${productid}${devicename};${sdkappid};${connid};${expiry}
PassWord: PassWord (setting any value).
ClientId:${ProductId}${DeviceName}
KeepAlive: time to keep the connection alive. Value range: 0-900s
```

Key Authentication Device Access Guide

1. Log in to the IoT Explorer console and enter the target product page. In the corresponding device information section, obtain the device key.
2. Connection Domain Name: The device in Guangzhou needs to connect to `${ProductId}.ap-guangzhou.iothub.tencentdevices.com:80`, where ${ProductId} is a variable parameter product ID.
3. MQTT Connection Parameter Settings:
   The connection parameter settings are consistent with those when using MQTT-TCP access. For details, see the Key Device Access Guide chapter in the Device-Based TCP MQTT Access document.

```
UserName:${productid}${devicename};${sdkappid};${connid};${expiry}
PassWord:${token};hmac signature method
ClientId:${ProductId}${DeviceName}
KeepAlive: time to keep the connection alive. Value range: 0-900s
```

MQTT Persistent Session

Last updated: 2025-04-27 17:50:03

IoT Explorer supports MQTT protocol version 3.1.1. It simultaneously supports QOS0 and QOS1 service quality grades (QOS2 is not supported). Use MQTT persistent session to can save the device's subscription status and the messages not received by the device. When a device goes offline and comes online again, it can recover to the previous session and receive the subscription messages not received when offline.

Create an MQTT Persistent Session on the Device Side

When a device connects to the IoT platform, the CleanSession flag bit in the variable header of the Connect message can be set to 0. The IoT platform will determine the session status of the device based on the client identifier ClientId during the device connection. If there is currently no session, a new persistent session will be created. If an existing session exists, communication will be performed based on the existing session process.

IoT Platform Response Description

After the device sends a Connect message, the Internet of Things (IoT) Hub will return a Connack Message. In the connection acknowledgment flag SessionPresent of the message, it indicates whether the Internet of Things (IoT) Hub has already created the session status corresponding to the client identifier when the device is connected. If SessionPresent is 0, it means a persistent session has not been created, and the device needs to reestablish the session status. If SessionPresent is 1, it indicates a persistent session has been created.

- After the device successfully connects, if it enters an existing persistent session, IoT Hub will send the stored QOS1 messages and unconfirmed QOS1 messages to the device.
- After the device successfully connects, if a new persistent session is created, IoT Hub will save the subscription status of the device and store the QOS1 messages (excluding QOS0) subscribed by the device when the device is offline. When the device goes live again, the stored QOS1 messages and unconfirmed QOS1 messages will be sent to the device.

Note:

- The Internet of Things Platform sends stored QOS1 messages sequentially at intervals of 500 ms.
- Only QOS1 messages are stored in a persistent session. A maximum of 150 messages can be stored per device, with a maximum storage duration of 24*7 hours.

Disable MQTT Persistent Session

You can close the MQTT persistent session in the following two ways.

- When a device connects to the IoT platform, set the CleanSession flag bit in the variable header of the Connect message to 1.
- The time when the device is disconnected **exceeds 24 hours**, the persistent session will be automatically closed.

> **Note:**
> Device disconnection includes the device sending a disconnect message and disconnection caused by device communication timeout.

Dynamic Registration Protocol

Last updated: 2025-04-27 17:50:16

Parameter Description

When performing dynamic device registration, carry ProductId and DeviceName to initiate an `http/https` request to the platform. The request API and parameters are as follows:

- Requested URL:
  `https://ap-guangzhou.gateway.tencentdevices.com/device/register`
  `http://ap-guangzhou.gateway.tencentdevices.com/device/register`
- Request method: Post

Request Parameter

Parameter Name	Required	Type	Description
ProductId	Yes	string	Product Id.
DeviceName	Yes	string	Device name.

Note:
The API only supports application/json format.

Signature Generation

Use the HMAC-sha256 algorithm to sign the request message. For details, see Signature Method.

Platform Response Parameters

Parameter Name	Type	Description
RequestId	String	Request Id.
Len	Int64	The length of the returned Payload.
Payload	String	Returned device registration information. The data is returned after encryption and needs to be decrypted and processed by the device side.

Note:

> The encryption process is to convert the original JSON format Payload into a string and then perform AES encryption, followed by base64 encryption. The AES encryption algorithm is in CBC mode, with a key length of 128 bits, using the first 16 bits of productSecret, and an offset of a 16-character string "0".

Original payload content description:

key	value	Description
encryptionType	1	Encryption type. • Certificate authentication. • 2 indicates key authentication.
psk	1239466501	Device key. This parameter is present when the product certification type is signature authentication.
clientCert	-	Device certificate file in string format. This parameter is present when the product certification type is certificate authentication.
clientKey	-	Device private key file in string format. This parameter is present when the product certification type is certificate authentication.

Example Code

Request Packet

```
POST https://ap-guangzhou.gateway.tencentdevices.com/device/register
Content-Type: application/json
Host: ap-guangzhou.gateway.tencentdevices.com
X-TC-Algorithm: HmacSha256
X-TC-Timestamp: 1551****65
X-TC-Nonce: 5456
X-TC-Signature:
2230eefd229f582d8b1b891af7107b91597****07d778ab3738f756258d7652c
{"ProductId":"ASJ****GX","DeviceName":"xyz"}
```

Response Package

```
{
  "Response": {
    "Len": 53,
    "Payload":
"031T01DWAoqFePDt71VuZXuLzkUzbIhGOnvMzpAFtNgOjagyFNHVSostNl9ztvhOuRx0dMM
/DMoWAXQCfL7jyA==",
    "RequestId": "f4da4f1f-d72e-40f1-****-349fc0072ba0"
  }
}
```


Payload Data Parsing Example

> **Note:**
> The following data is provided only for your testing use. Ensure that your information is not leaked when you use it officially.

1. Payload

```
s6FB3a1BA/YYbcmSE12XpeDVmQNDcf1QgVD141RRbmmAnFwQfp1ECAu5O016mCOvYlJJ6V
59yM4OqQSiWphfTg==
```

2. Base64 decoded:

```
b3a141ddad4103f6186dc992135d97a5e0d599034371fd508150f5e354516e69809c5c
107e9d44080bb93b4d7a9823af625249e95e7dc8ce0ea904a25a985f4e
```

3. AES decryption.

   Product key: `hzvf5LF9S0isvBhDSauWMaIk`
   Decrypted data: `{"encryptionType":2,"psk":"lDZ6Uqt+I9E0wW7rvDUs7Q=="}`

Thing Model Protocol

Last updated: 2025-04-27 17:50:31

Overview

The Thing Model can digitally define the features of devices in the physical world, making it easy for applications to conveniently manage devices. The platform provides users with a business protocol based on the Thing Model, which can meet the needs of smart life scenarios as well as those of various vertical industry applications in the IoT.

- Smart life scenario: Based on the Thing Model Protocol, after users submit device-related attributes, events, etc., to the cloud, they can seamlessly use Tencent Lianlian Mini Program or Chinese domestic brand mini programs and apps. There is no need to handle the communication details between the cloud and the mini program or app, enhancing the Application Development Efficiency of users in smart life scenarios.
- Vertical industry application scenarios: Based on the Thing Model Protocol, there is no need for users to parse device data. Users can use the data analysis, Alarm, and storage services of IoT Explorer and related Cloud services of Tencent Cloud to enhance the development efficiency of vertical industry applications.

Thing Model Protocol

Overview

After the product defines the Thing Model, the device can report attributes and events according to the definitions in the Thing Model, and control commands can be issued to the device. For details on Thing Model management, see Product Definition. The Thing Model Protocol includes the following parts.

- Attribute reporting of devices: The device end submits the defined attributes to the cloud according to the business logic of the device end.
- Remote control of devices: Issue control commands from the cloud to the device end, that is, set the writable attributes of the device on the cloud.
- Obtain the latest reported information of the device: Obtain the latest reported data of the device.
- Device event reporting: The device can report alarms, faults and other event information according to the protocol of device event reporting when an event defined in the Thing Model is triggered.
- Device behavior invocation: The cloud can notify the device to perform a certain action via RPC, suitable for scenarios where the application needs to obtain the device's execution result in real time.

- Submit initial device information: Submit initial information when the device connects to the platform, making it easy for mini programs or Apps to display device details, such as the device MAC address, IMEI number.
- User deletes a device: A notification message sent from the cloud to the device when a user deletes a device in the Tencent Lianlian Mini Program or the user's own brand mini program, making it easy for the device to reset or for gateway devices to clear subdevice data.

Equipment Attribute Reporting

1. When the equipment needs to report changes in the device running status to the cloud, notify the application side mini program, App to display in real time or the cloud business system to receive the attribute data reported by the equipment. The IoT Explorer has set a default Topic for the equipment:
   - Device property uplink request Topic: `$thing/up/property/{ProductID}/{DeviceName}`
   - `$thing/down/property/{ProductID}/{DeviceName}` : Device property downlink response Topic
2. Request.
   - Request message example on the device side

```
{
    "method":"report",
    "clientToken":"123",
    "timestamp":1628646783,
    "params":{
        "power_switch":1,
        "color":1,
        "brightness":32
       }
}
```

   - request parameter description

Parameter	Type	Description
method	String	report indicates device property reporting.
clientToken	String	Used for uplink and downlink message pairing identifier.
timestamp	Integer	The time of attribute reporting, format: UNIX system timestamp. Not filling in this field means it defaults to

		the current system time. Unit: seconds.
params	JSON	The JSON structure contains attribute values reported by the device.
params.power_switch	Boolean	The value of a Boolean attribute is generally 0 or 1.
params.color	Enum	Enumerating integer attribute values as integer values, a 406 response code appears if the numerical type is filled in incorrectly or exceeds the defined range of enumeration items, indicating a format validation error of the Thing Model.
params.brightness	Integer	The value of an integer attribute is an integer value. A 406 response code will appear if the numerical type is filled in incorrectly or exceeds the value range, indicating a format validation error of the Thing Model.

3. Respond.
   - Message example returned by the cloud to the device side

```
{
    "method":"report_reply",
    "clientToken":"123",
    "code":0,
    "status":"some message where error"
}
```

   - response parameter description

Parameter	Type	Description
method	String	report_reply indicates the HTTP response message from the cloud after receiving device reporting.
clientToken	String	Used for uplink and downlink message pairing identifier.
code	Integer	0 means the cloud has successfully received the attributes reported by the device.
status	String	prompt error message when code is not 0.

Remote Control of Equipment

1. Devices using the Thing Model Protocol need to subscribe to the published Topic to receive cloud instructions when the cloud needs to control the devices:
   - Send Topic: `$thing/down/property/{ProductID}/{DeviceName}`
   - `$thing/up/property/{ProductID}/{DeviceName}` : Response Topic
2. Request.
   - Remote control request message format:

```
{
    "method": "control",
    "clientToken": "123",
    "params": {
        "power_switch": 1,
        "color": 1,
        "brightness": 66
    }
}
```

   - request parameter description

Parameter	Type	Description
method	String	Control means the cloud initiates a control request to the device.
clientToken	String	Used for uplink and downlink message pairing identifier.
params	JSON	The JSON structure contains set values of device properties. Only writable attribute values can be controlled successfully.

3. Respond.
   - Equipment response to remote control request message format:

```
{
    "method":"control_reply",
    "clientToken":"123",
    "code":0,
    "status":"some message where error"
```

```
}
```

- response parameter description

Parameter	Type	Description
method	String	Request response indicating the control command sent by the device to the cloud.
clientToken	String	Used for uplink and downlink message pairing identifier.
code	Integer	0 means the device has successfully received the control command sent by the cloud.
status	String	prompt error message when code is not 0.

Obtain the Latest Reported Information of the Device

1. Topic used by devices to receive the latest messages from the cloud:
   - Request Topic: `$thing/up/property/{ProductID}/{DeviceName}`
   - `$thing/down/property/{ProductID}/{DeviceName}` : Response Topic
2. Request.
   - Request message format:

```
{
    "method": "get_status",
    "clientToken": "123",
    "type" : "report",
    "showmeta": 0
}
```

   - request parameter description

Parameter	Type	Description
method	String	get_status: Obtain the latest reported information of the device.
clientToken	String	Message Id. The replied message will return this data for comparison with the request response message.
type	String	Indicates what kind of information is obtained. report

		indicates the information reported by the device.
showmeta	Integer	Flag whether the reply message carries metadata. Default to 0 means not returning metadata.

3. Respond.
   - Response message format:

```
{
    "method": "get_status_reply",
    "code": 0,
    "clientToken": "123",
    "type": "report",
    "data": {
        "reported": {
            "power_switch": 1,
            "color": 1,
            "brightness": 66
        }
    }
}
```

   - response parameter description

Parameter	Type	Description
method	String	reply message indicating the latest reported information obtained from the device.
code	Integer	0. The cloud successfully received the attributes reported by the device.
clientToken	String	Message Id. The replied message will return this data for comparison with the request response message.
type	String	Indicates what kind of information is obtained. report indicates the information reported by the device.
data	JSON	Return the latest reported data content of the specific device.

Device Event Reporting

1. When a device needs to report events to the cloud, such as reporting device failures and alarm data, the platform sets a default Topic for the device:

- Device event uplink request Topic: `$thing/up/event/{ProductID}/{DeviceName}`
- `$thing/down/event/{ProductID}/{DeviceName}` : Device event downlink response Topic

2. Request.
   - Request message example on the device side

```
{
    "method":"event_post",
    "clientToken":"123",
    "version":"1.0",
    "eventId":"PowerAlarm",
    "type":"fault",
    "timestamp":1212121221,
    "params":{
        "Voltage":2.8,
        "Percent":20
     }
}
```

   - request parameter description

Parameter	Type	Description
method	String	event_post means event reporting.
clientToken	String	Message ID. The replied message will return this data for comparison with the request response message.
version	String	Protocol version, default is 1.0.
eventId	String	Event Id, defined in the Thing Model event.
type	String	Event type. • info: information. • alert: Alarm. • fault: Failure.
params	String	Parameters of the event, defined in the Thing Model event.
timestamp	Integer	The time of event reporting. Not filling in this field means it defaults to the current system time. Unit: seconds.

3. Respond.

- Response message format:

```
{
    "method": "event_reply",
    "clientToken": "123",
    "version": "1.0",
    "code": 0,
    "status": "some message where error",
    "data": {}
}
```

- response parameter description

Parameter	Type	Description
method	String	event_reply indicates the response returned by the cloud to the device.
clientToken	String	Message Id. The replied message will return this data for comparison with the request response message.
version	String	Protocol version, default is 1.0.
code	Integer	Event report result, 0 indicates success.
status	String	Event report result description.
data	JSON	The content returned by the event report.

Device Behavior Invocation

1. When the application initiates a certain behavior invocation of the device through the cloud, the platform has set a default Topic for the handling of the device behavior:
   - `$thing/down/action/{ProductID}/{DeviceName}` : Application call device behavior Topic
   - Equipment response behavior execution result Topic: `$thing/up/action/{ProductID}/{DeviceName}`
2. Request.
   - Message example of device behavior invocation initiated by the application side:

```
{
    "method": "action",
    "clientToken": "20a4ccfd-d308-****-86c6-5254008a4f10",
```

```
    "actionId": "openDoor",
    "timestamp": 1212121221,
    "params": {
        "userid": "323343"
    }
}
```

- request parameter description

Parameter	Type	Description
method	String	action indicates calling a certain behavior of the device.
clientToken	String	Message Id. The replied message will return this data for comparison with the request response message.
actionId	String	actionId is a behavior identifier in the Thing Model, defined by the developer manually according to the device's application scenario.
timestamp	Integer	The current time of the behavior invocation. If not specified, it defaults to the current system time of the behavior invocation. Unit: seconds.
params	String	Calling parameters of the behavior, defined in the behavior of the Thing Model.

3. Respond.
   - Response message format:

```
{
    "method": "action_reply",
    "clientToken": "20a4ccfd-d308-11e9-86c6-5254008a4f10",
    "code": 0,
    "status": "some message where error",
    "response": {
        "Code":  0
     }
}
```

   - response parameter

Parameter	Type	Description
method	String	action_reply is the response that the device end sends to the cloud after executing the specified behavior.
clientToken	String	Message Id. The replied message will return this data for comparison with the request response message.
code	Integer	Action execution result, 0 indicates success.
status	String	Error information description after action execution failure.
response	JSON	The response parameters defined in the device behavior are returned to the cloud after the device behavior is executed successfully.

Reporting Basic Information of Devices

1. When a mini program or App displays device details, it generally shows the device's MAC address, IMEI number, time zone and other basic information. Topic used for device information reporting:
   - Uplink request Topic: `$thing/up/property/{ProductID}/{DeviceName}`
   - Downlink response Topic: `$thing/down/property/{ProductID}/{DeviceName}`
2. Request.
   - Request message example on the device side

```
{
    "method": "report_info",
    "clientToken": "1234567",
    "params": {
        "name": "dev001",
        "imei": "ddd",
        "module_hardinfo": "ddd",
        "mac": "ddd",
        "device_label": {
            "append_info": "ddddd"
        }
    }
}
```

   - Request Parameter Description

Parameter	Type	Description
method	String	report_info indicates device basic information reporting.
clientToken	String	Used for uplink and downlink message pairing identifier.
imei	String	IMEI number information of the device, optional field.
mac	String	MAC information of the device, optional field.
module_hardinfo	String	Specific hardware model of the module.
append_info	String	Custom basic product information of the equipment vendor, submitted in KV format.

3. Respond.
   - Message example returned by the cloud to the device side

```
{
    "method":"report_info_reply",
    "clientToken":"1234567",
    "code":0,
    "status":"success"
}
```

   - response parameter description

Parameter	Type	Description
method	String	report_reply indicates the HTTP response message from the cloud after receiving device reporting.
clientToken	String	Used for uplink and downlink message pairing identifier.
code	Integer	0 means the cloud has successfully received the attributes reported by the device.
status	String	prompt error message when code is not 0.

User Deletion of Device

1. When a user deletes a bound device in a mini program or App, the platform will send a notification of user deletion of the device to the device. After receiving it, the device can handle it based on business needs. For example, if a gateway device receives a notification that a subdevice has been deleted.

   Send user deletion Device Topic: `$thing/down/service/{ProductID}/{DeviceName}`
2. Request.
   - Message example of user deletion of device initiated by the application side:

```
{
    "method": "unbind_device",
    "clientToken": "20a4ccfd-****-11e9-86c6-5254008a4f10",
    "timestamp": 1212121221
}
```

   - Request Parameter Description

Parameter	Type	Description
method	String	unbind_device means that the user removes or unbinds a certain device in the mini program or App.
timestamp	Integer	The system timestamp when the user deletes the device.

User Binding Device Notification Message

1. After a user successfully binds a device in a mini program or App, the platform will send a notification message that the device has been bound by the user to the device. After receiving it, the device can handle it based on business needs. This message is used to notify the device end from the mobile application end, and there is no need for the device end to reply.

   Send user binding device notification message Topic: `$thing/down/service/{ProductID}/{DeviceName}`
2. Request.
   - Message example of user device binding notification initiated by the application side:

```
{
    "method": "bind_device",
    "clientToken": "20a4ccfd-****-11e9-86c6-5254008a4f10",
```

```
    "timestamp": 1212121221
}
```

- request parameter description

Parameter	Type	Description
method	String	bind_device means that the user binds a certain device in the mini program or App.
clientToken	String	Used for uplink and downlink message pairing identifier.
timestamp	Integer	System timestamp for user to bind device.

Location Service Fence Alarm Message Delivery

1. When a user creates and associates a geographic electronic fence for a device in the Console **Location Service** feature, mini program, or App, the platform will send a fence alarm message notification to the device when the device triggers the fence alarm condition. After receiving it, the device can handle it by itself based on business needs. For example, if the device receives a fence alarm message, it will voice broadcast to remind the user or manager using the device to pay attention to safety.
   - `$thing/down/service/{ProductID}/{DeviceName}` : Downlink Fence Alarm Message Topic
   - Device response reply Topic: `$thing/up/service/{ProductID}/{DeviceName}`
2. The cloud sends down fence alarm messages.
   - Message example of alarm message sent by the cloud

```
{
    "method":"alert_fence_event",
    "clientToken":"xx",
    "timestamp": 0,
    "data": {
        "alert_type":"xx", //Event, In Out InOrOut
        "alert_condition":"xx", //Trigger condition for device
binding fence In Out InOrOut
        "alarm_time": 0, // Alarm time
        "fence_name":"xx", // Fence name
        "long": 0,
        "lat": 0
    }
```

```
}
```

- request parameter description

Parameter	Type	Description
method	String	alert_fence_event indicates a fence alarm event.
clientToken	String	Used for uplink and downlink message pairing identifier.
timestamp	Integer	Timestamp, in seconds.
data.alert_type	String	alarm event type: In, Out, InOrOut.
data.alert_condition	String	Trigger conditions for device-bound fence: In, Out, InOrOut.
data.alarm_time	Int	Alarm time.
data.fence_name	String	Fence name.
data.long	Float	Longitude.
data.lat	Float	Latitude.

3. Device end replies.
   - Message example returned by the device side to the cloud

```
{
    "method":"alert_fence_event_reply",
    "clientToken":"xx",
    "timestamp": 0,
    "code":0,
    "status":"success"
}
```

   - response parameter description

Parameter	Type	Description
method	String	alert_fence_event_reply indicates a reply to a fence alarm.

clientToken	String	Used for uplink and downlink message pairing identifier.
timestamp	Int	Timestamp.
code	Int	0 indicates it has been correctly handled.
status	String	Prompt error message when code is not 0.

Error Code

code	Description
400	The message format is not JSON format.
403	Incorrect method identifier or property, event, behavior identifier inconsistent with the identifiers defined in the Thing Model.
405	Timestamp error. The difference between the current time and the reporting time is 24 hours. Note that the timestamp is in seconds.
406	Data type error of the Thing Model parameter input value or the data exceeds the defined range.
503	Internal system error.

Firmware Upgrade Protocol

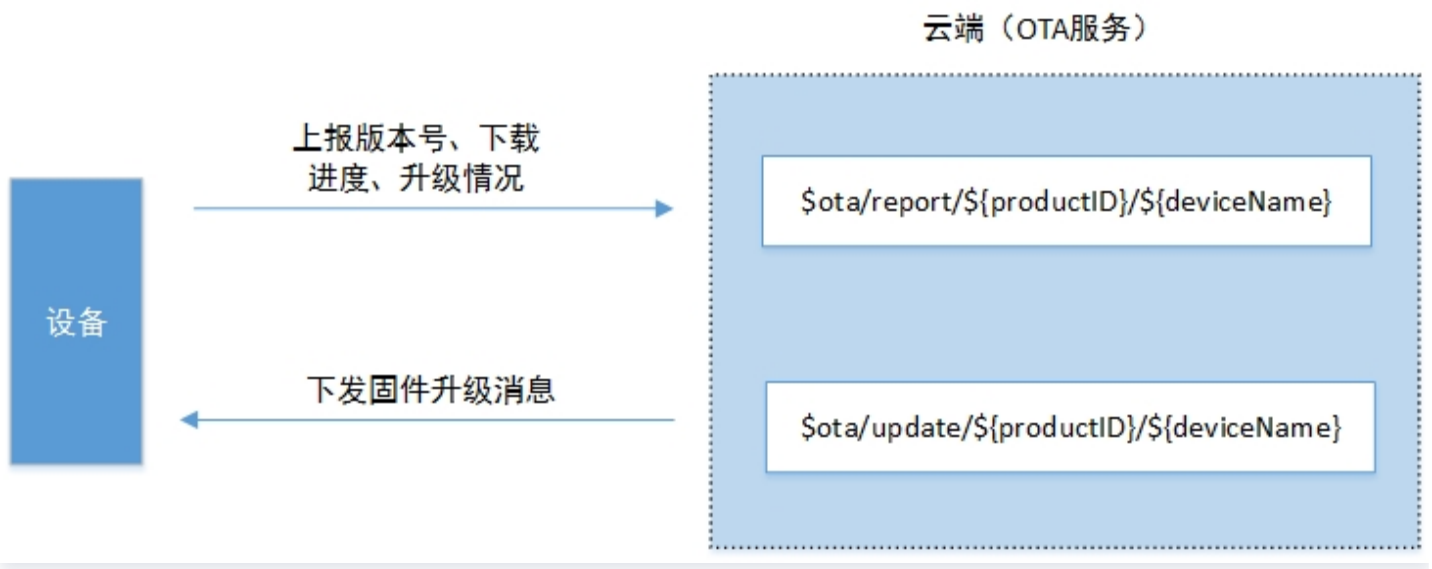
Last updated: 2025-04-27 17:50:47

Overview

Device firmware upgrade, also known as OTA, is a crucial part of the Internet of Things (IoT) Hub service. When IoT devices have new features or need to fix vulnerabilities, devices can quickly perform firmware upgrades through the OTA service.

Implementation Principles

During the firmware upgrade process, device subscription to the following two topics is required to achieve communication with the cloud, as shown in the figure below:

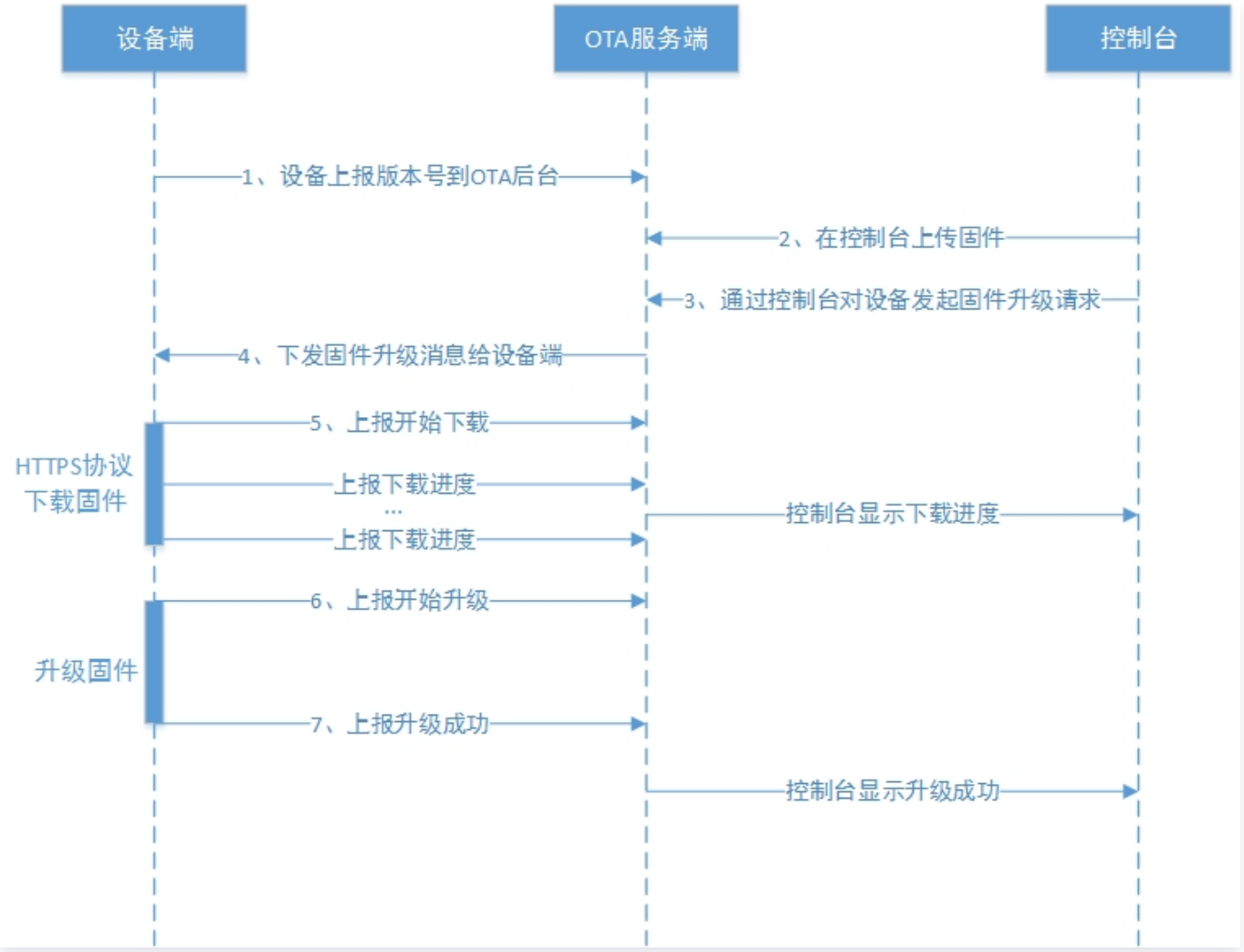


Example as follows:

```
$ota/report/${productID}/${deviceName}
Used to publish (uplink) messages, device reporting version number and
download, upgrade progress to the cloud
$ota/update/${productID}/${deviceName}
Used to subscribe (downlink) messages, device receives upgrade messages
from the cloud
```

Operation Process

The upgrade procedure of the equipment is as follows:



1. The device submits the current version number. The device publishes a message to Topic `$ota/report/${productID}/${deviceName}` via the MQTT protocol to report the version number. The message is in json format, with the following content:

```
{
    "type": "report_version",
    "report":{
        "version": "0.1"
    }
}
// type: message type
// version: The submitted version number
```

2. Then you can upload the firmware through the console.
3. Upgrade the specified device to the specified version on the console.

4. After triggering the firmware upgrade operation, the device will receive firmware upgrade messages through the subscribed Topic `$ota/update/${productID}/${deviceName}`, with the following content:

```
{
    "file_size": 708482,
    "md5sum": "36eb5951179db14a63**********22a2",
    "type": "update_firmware",
    "url": "https://ota-125*****90.cos.ap-guangzhou.myqcloud.com",
    "version": "0.2"
}
// type: the message type is update_firmware
// version: Upgrade version
// url: The URL for downloading firmware
// md5sum: The MD5 value of the firmware
// file_size: firmware size in bytes
```

5. After receiving the firmware upgrade message, the device downloads the firmware according to the URL. During the download process, the Device SDK continuously reports the download progress through the Topic `$ota/report/${productID}/${deviceName}`. The reported content is as follows:

```
{
    "type": "report_progress",
    "report":{
        "progress":{
            "state":"downloading",
            "percent":"10",
            "result_code":"0",
            "result_msg":""
        },
    "version": "0.2"
    }
}
// type: message type
// state: status is downloading
// percent: current download progress, percentage
```

6. When the device finishes downloading the firmware, it needs to submit a message to start the upgrade through the Topic `$ota/report/${productID}/${deviceName}`. The content is as follows:

```
{
    "type": "report_progress",
    "report":{
        "progress":{
            "state":"burning",
            "result_code":"0",
            "result_msg":""
        },
    "version": "0.2"
    }
}
// type: message type
// state: status is burning in progress
```

7. After the device firmware upgrade is completed, submit an upgrade success message to the Topic `$ota/report/${productID}/${deviceName}`. The content is as follows:

```
{
    "type": "report_progress",
    "report":{
        "progress":{
        "state":"done",
        "result_code":"0",
        "result_msg":""
        },
    "version": "0.2"
    }
}
// type: message type
// state: status is completed
```

During the process of downloading or upgrading firmware, if it fails, submit a firmware upgrade failure message through Topic `$ota/report/${productID}/${deviceName}`. The content is as follows:

```
{
    "type": "report_progress",
    "report":{
        "progress":{
        "state":"fail",
        "result_code":"-1",
```

```
        "result_msg":"time_out"
        },
    "version": "0.2"
    }
}
// state: status is failed
// result_code: error code, -1: Download Timeout; -2: File does not
exist; -3: Signature expired; -4: MD5 mismatch; -5: firmware update
failed
// result_msg: error message
```

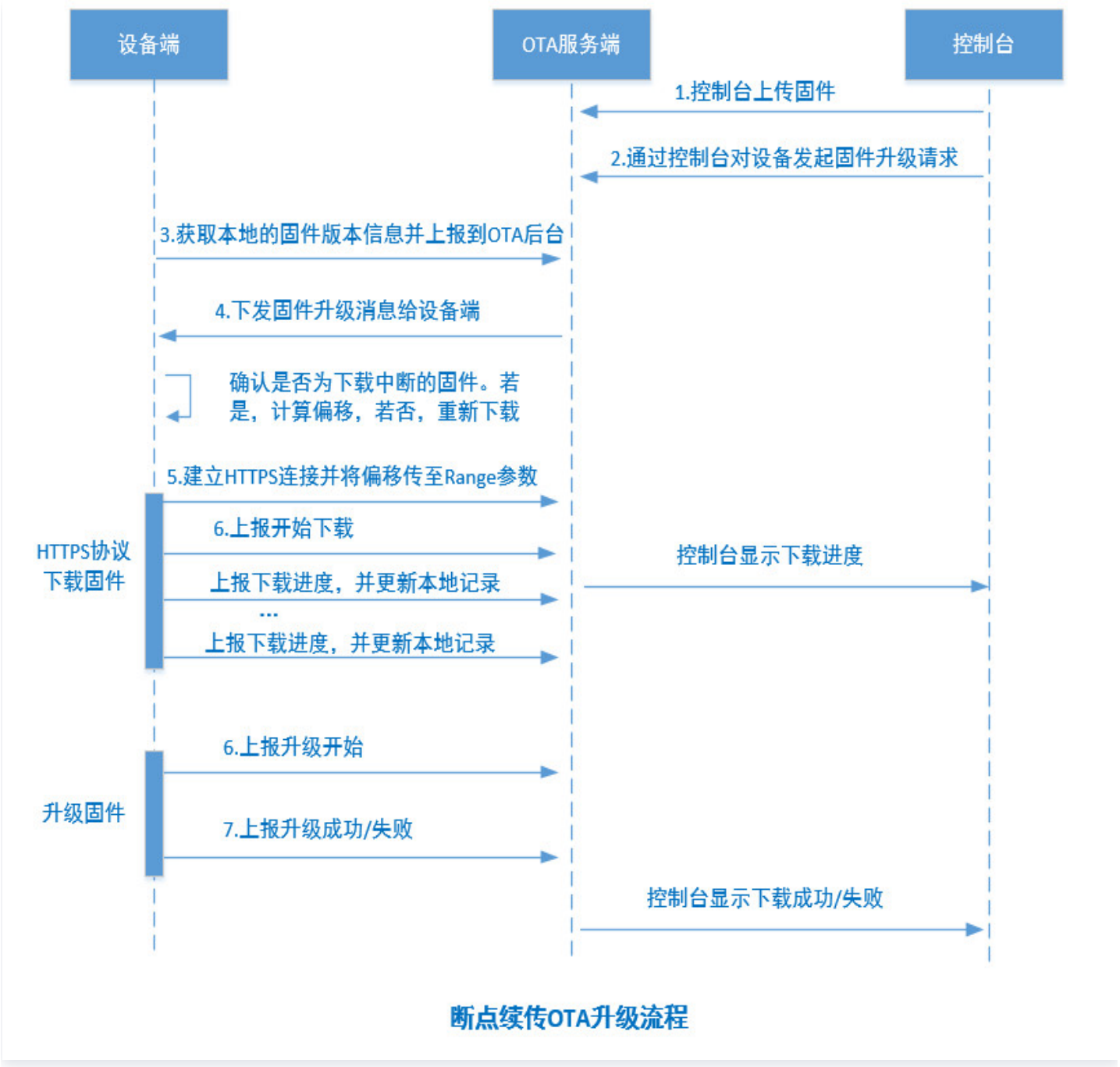
OTA Breakpoint Resume

IoT devices are in a weak network environment in some scenarios. In this scenario, the connection will be unstable, and the firmware download may be interrupted. If the firmware always starts to download from an offset of 0 each time, it is possible that the firmware download cannot be completed in a weak network environment. Therefore, the breakpoint resume function of the firmware is especially necessary.

- Resumable uploading refers to re-downloading or uploading from the last interruption of a file. To implement the resumable uploading feature, the device needs to record the interruption position of the firmware download, and record the MD5, individual file size, and version information of the downloaded firmware.
- For scenarios of OTA interruption, the platform reports the device version on the device side. If the reported version number is inconsistent with the target version number to be upgraded, the platform will send the firmware upgrade message again. After the device obtains the information of the target firmware to be upgraded and compares it with the interrupted firmware information recorded locally, if it is confirmed to be the same firmware, it will continue to download based on the breakpoint.

The OTA upgrade process with breakpoint resume is as follows:

> **Note:**
> Steps 3 to 6 may be executed multiple times in a weak network environment. If step 7 is not executed and step 3 is executed, the device will receive the messages of step 4 each time.



断点续传OTA升级流程

Gateway Subdevice Feature Overview

Last updated: 2025-04-27 17:51:12

Device Classification

The IoT development platform divides devices into the following three categories (i.e., node categories) according to their functionality:

- **Ordinary device**: this type of device can directly access the IoT development platform and has no mounted subdevices.
- **Gateway device**: This type of device can directly access the IoT development platform and can accept subdevices to join the LAN.
- **subdevice**: This type of device must rely on a gateway device to communicate with IoT Explorer, such as Zigbee, Bluetooth, RF433 devices.

Overview

- For devices that don't have direct access to the Internet, they can be connected to the network of the local gateway device first and then access the IoT development platform through the communication feature between the gateway device and the cloud.
- For subdevices that join or exit the network in the LAN, the gateway device can perform bind or unbind operations on them on the platform and report the topological relationship with the subdevices to achieve the management of the entire LAN subdevices by the platform.

Access Method

- Gateway devices can be connected to the IoT development platform in the same way as ordinary devices. After a gateway device is connected, it can connect/disconnect subdevices in the same LAN to/from the platform, proxy device-reported data of subdevices, proxy the receipt of data sent from the cloud to subdevices, and manage the topological relationships between them.
- The access of subdevices needs to be completed through gateway devices. After the authentication of the identity of subdevices is completed through gateway devices, they can successfully access the cloud. The authentication methods are divided into the following two:
  - **Device-level key method**
    The gateway obtains the device certificate or key of the sub-device and generates a sub-device binding signature string. The gateway reports the sub-device binding

signature string information to the platform and acts as a proxy for the sub-device to complete identity verification.

- **Product-level key method**
  The gateway obtains the ProductSecret (product key) of the subdevice and generates a signature. The gateway sends a dynamic registration request to the platform. If the verification is successful, the platform will return the DeviceCert or DeviceSecret of the subdevice. The gateway device will generate a signature string for sub-device binding based on this and report the information of the sub-device binding signature string to the platform. After the verification is successful, the access of the subdevice is completed.

Topology Relationship Management

Last updated: 2025-04-27 17:51:26

Feature Overview

For devices of the gateway type, they can bind and unbind their subdevices through data communication with the cloud. To implement such functionality, the following two topics need to be leveraged:

- Data upstream Topic (for publishing): `$gateway/operation/${productid}/${devicename}`
- Data downstream Topic (for subscription): `$gateway/operation/result/${productid}/${devicename}`

Binding Device

Devices of the gateway type can use the data upstream Topic to request adding the topological relationship between them and their subdevices to implement binding sub-devices. After successful request, the cloud returns the binding result information of the subdevice through the data downstream Topic.
Data format of the gateway binding sub-device request:

```
{
  "type": "bind",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subdeviceaaaa",
        "signature": "signature",
        "random": 121213,
        "timestamp": 1589786839,
        "signmethod": "hmacsha256",
        "authtype": "psk"
      }
    ]
  }
}
```

Response data format for binding sub-device by gateway:

```
{
```

```
  "type": "bind",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subaaa",
        "result": -1
      }
    ]
  }
}
```

Request Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. The value for binding sub-device is: `bind`.
payload.devices	Array	The list of sub-devices that should be bound.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.
signature	String	Sub-device binding signature string. Signature algorithm: 1. Signature original string, concatenate the product ID, device name, random number, and timestamp: text=${product_id}${device_name};${random};${timestamp} 2. Use the device Psk key or the Sha1 abstract of the certificate for signing: `base64_encode(hmac_Sha1(device_secret, text))`
random	Int	Random number.
timestamp	Int	Timestamp, in seconds.
signmethod	String	Signature algorithm. Support hmacsha1, hmacsha256.
authtype	String	Signature type. • psk: Signing using device psk.

		• certificate: Signing using the device public key certificate.

Response Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. The value for binding sub-device is: `bind`.
payload.devices	Array	List of sub-devices that should be bound.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.
result	Int	Sub-device binding result. See Error Code in the table below.

Unbinding Device

Devices of the gateway type can use the data upstream Topic to request unbinding the topological relationship between them and their sub-devices. After successful request, the cloud returns the unbinding information of the sub-devices through the data downstream Topic.
Request data format for unbinding sub-device by gateway:

```
{
  "type": "unbind",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subaaa"
      }
    ]
  }
}
```

Response data format for gateway unbinding sub-device:

```
{
  "type": "unbind",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subaaa",
        "result": -1
      }
    ]
  }
}
```

Request Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. The value for unbinding sub-device is: `unbind`.
payload.devices	Array	The list of sub-devices that need to be unbound.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.

Response Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. The value for unbinding sub-device is: `unbind`.
payload.devices	Array	The list of sub-devices that need to be unbound.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.
result	Int	Sub-device binding result. See Error Code in the table below.

Enable the Search State of the Notification Gateway

When the application side mini program or app needs to enter a certain sub-device binding process, the platform will notify the Notification Gateway to enable and disable the sub-device search feature. The protocol is as follows:

- Data upstream Topic (for publishing): `$gateway/operation/${productid}/${devicename}`
- Data downstream Topic (for subscription): `$gateway/operation/result/${productid}/${devicename}`

Data Format Delivered by the Platform

```
{
    "type": "search_devices",
    "payload": {
    "status": 0 //0-off 1-on
    }
}
```

Request Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. Notification Gateway enables search state, value is: search_devices.
status	Int	Gateway search state: • 0: Disable. • 1: Enable.

Data Format of Gateway Reply

```
{
    "type": "search_devices",
    "payload": {
    "status": 0, //0-off 1-on
    "result": 0
    }
}
```

Field	Type	Description
type	String	Gateway Message Type. Notification Gateway enables search state, value is: search_devices.

status	Int	Gateway search state: • 0: Off. • 1: Enable.
result	Int	Gateway Response Handling Result • 0: Success. • 1: Failure.

Topology Relationship Query

Devices of the gateway type can use this Topic for an upstream request to query the topological relationship of subdevices.
Gateway query sub-device topology relationship request data format:

```
{
    "type": "describe_sub_devices"
}
```

Request parameter description:

Parameter	Type	Description
type	String	Gateway Message Type. The value for querying sub-devices is: `describe_sub_devices`.

Gateway query sub-device topology relationship response data format:

```
{
  "type": "describe_sub_devices",
  "payload": {
    "devices": [
      {
        "product_id": "XKFA****LX",
        "device_name": "2OGDy7Ws8mG****YUe"
      },
      {
        "product_id": "XKFA****LX",
        "device_name": "5gcEHg3Yuvm****2p8"
      },
      {
        "product_id": "XKFA****LX",
```

```
        "device_name": "hmIjq0gEFcf****F5X"
      },
      {
        "product_id": "XKFA****LX",
        "device_name": "x9pVpmdRmET****mkM"
      },
      {
        "product_id": "XKFA****LX",
        "device_name": "zmHv6o6n4G3****Bgh"
      }
    ]
  }
}
```

Response Parameter Description:

Parameter	Type	Description
type	String	Gateway Message Type. The value for querying sub-devices is: `describe_sub_devices`.
payload.devices	Array	The list of sub-devices bound to the gateway.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.

Topology Change

Devices of the gateway type can subscribe to the platform through this data downlink Topic to get topology changes of subdevices.
When a sub-device is bound or unbound, the gateway will receive topology changes of the sub-device. The data format is as follows:

```
{
  "type": "change",
  "payload": {
  "status": 0,  //0-unbind 1-bind
  "devices": [
      {
        "product_id": "CFCS****G7",
        "device_name": "****ev",
```

```
        }
      ]
    }
}
```

Request Parameter Description:

Parameter	Type	Description
type	String	Gateway Message Type. Topology change value is: change.
status	Int	Topology change status. • 0: Unbind. • 1: Bind.
payload.devices	Array	The list of sub-devices bound to the gateway.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.

Gateway response, data format as follows:

```
{
  "type": "change",
  "result": 0
}
```

Response Parameter Description:

Parameter	Type	Description
type	String	Gateway Message Type. Topology change value is: change.
result	Int	Gateway Response Handling Result.

Error Codes

Error Code	Description

0	Succeeded.
-1	Gateway device not bound to subdevice.
-2	System error. Subdevice online or offline failure.
801	Request parameter error.
802	Invalid device name or device does not exist.
803	Signature verification failed.
804	The signature method is not supported.
805	The signed request has expired.
806	The device has been bound.
807	Non-standard equipment cannot be bound.
808	The operation is not allowed.
809	Repeat binding.
810	Unsupported subdevice.

Proxy Sub-Device Online and Offline

Last updated: 2025-04-27 17:51:41

Feature Overview

Devices of the gateway type can perform online and offline operations for their subdevices through data communication with the cloud. The topics used by this functionality match those for gateway subdevice topology management:

- Data upstream Topic (for publishing): `$gateway/operation/${productid}/${devicename}`
- Data downstream Topic (for subscription): `$gateway/operation/result/${productid}/${devicename}`

Proxy Sub-Device Online

Devices of the gateway type can use the data upstream Topic to proxy subdevices to go live. After the request is successful, the cloud returns the subdevice online result information through the data downstream Topic.
Request data format for gateway proxying sub-device online

```
{
  "type": "online",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subdeviceaaaa"
      }
    ]
  }
}
```

Response data format for proxying sub-device online

```
{
  "type": "online",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subdeviceaaaa",
        "result":0
```

```
        }
      ]
    }
}
```

Request parameter description:

Field	Type	Description
type	String	Gateway message type. The value for agent sub-device online is: `online`.
payload.devices	Array	List of sub-devices to be online.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.

Response Parameter Description:

Field	Type	Description
type	String	Gateway message type. The value for agent sub-device online is: `online`.
payload.devices	Array	List of sub-devices to be online.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.
result	Int	Subdevice Online Result. See Error Code in the table below.

Agent Subdevice Offline

Devices of the gateway type can use the data upstream Topic to proxy subdevices to go offline. After the request is successful, the cloud returns the offline information of the successful subdevice through the data downstream Topic.
Request data format for gateway proxying sub-device offline

```
{
  "type": "offline",
  "payload": {
    "devices": [
```

```
        {
          "product_id": "CFC******AG7",
          "device_name": "subdeviceaaaa"
        }
      ]
    }
}
```

Response data format for gateway proxying sub-device offline

```
{
  "type": "offline",
  "payload": {
    "devices": [
      {
        "product_id": "CFC******AG7",
        "device_name": "subdeviceaaaa",
        "result":-1
      }
    ]
  }
}
```

Request parameter description:

Field	Type	Description
type	String	Gateway Message Type. The value for agent sub-device offline is: `offline`.
payload.devices	Array	List of sub-devices to be offline via agent.
product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.

Response Parameter Description:

Field	Type	Description
type	String	Gateway Message Type. The value for agent sub-device offline is: `offline`.
payload.devices	Array	List of sub-devices to be offline via agent.

product_id	String	Subdevice Product ID.
device_name	String	Subdevice name.
result	Int	Subdevice Offline Result. Specific Error Codes are in the table below.

Error Code

Error Code	Description
0	Succeeded.
-1	Gateway device not bound to subdevice.
-2	System error. Subdevice online or offline failure.
801	Request parameter error.
802	Invalid device name, or device does not exist.
810	Unsupported subdevice.

Proxied Subdevice Publish and Subscribe

Last updated: 2025-04-27 17:51:54

Feature Overview

A gateway device can publish and subscribe to messages on behalf of its subdevices through data communication with the cloud.

Prerequisites

Before publishing and subscribing to messages, please see gateway device integration and sub-device online and offline to integrate gateway devices and subdevices online.

Publish and Subscribe to Messages

After the gateway product establishes a binding with the subproduct and obtains the Topic permissions of the subdevice, the gateway device can use the subdevice Topic proxy to send/receive messages. You can also view the communication information in Device Debugging - Device Log.

Sub-Device Firmware Upgrade

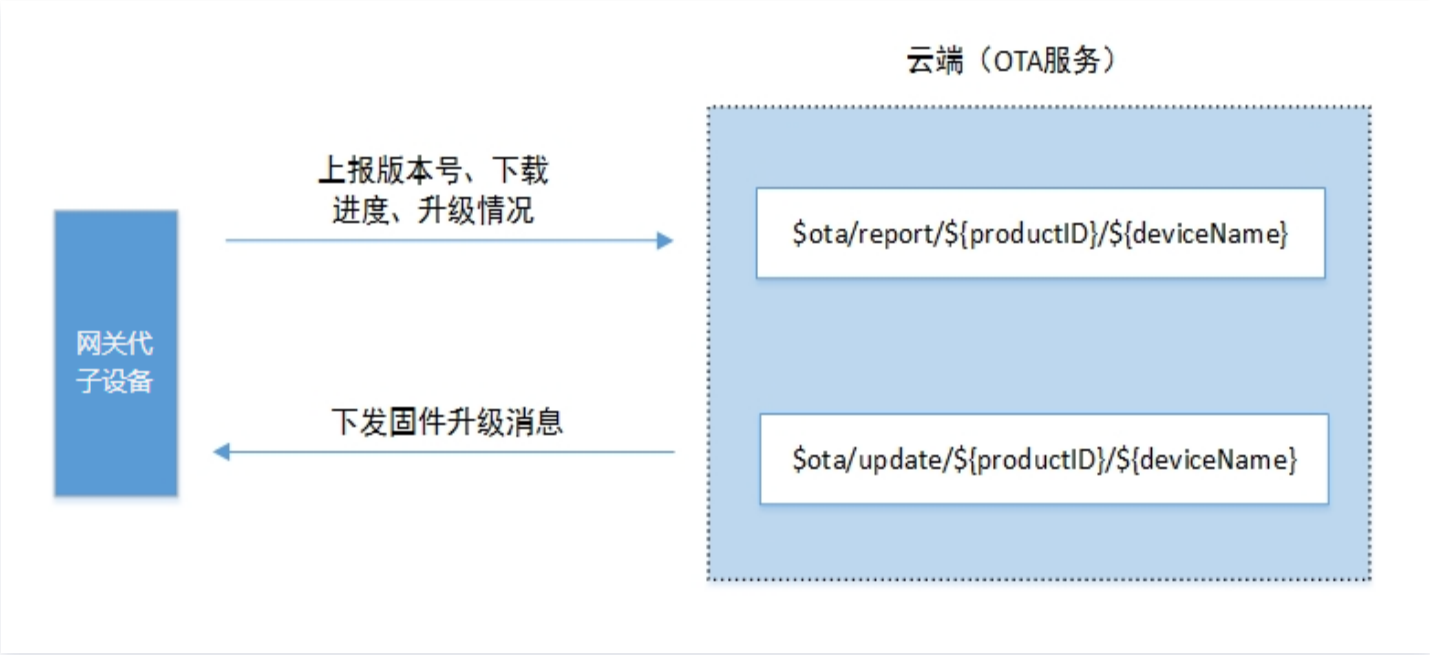
Last updated: 2025-04-27 17:52:08

Overview

When a gateway subdevice has a new feature or needs to fix a vulnerability, the subdevice can perform a firmware upgrade quickly through the device firmware upgrade service.

Implementation Principles

During the firmware upgrade process, the gateway needs to use the following two topics on behalf of the sub-device to communicate with the cloud, as shown below:



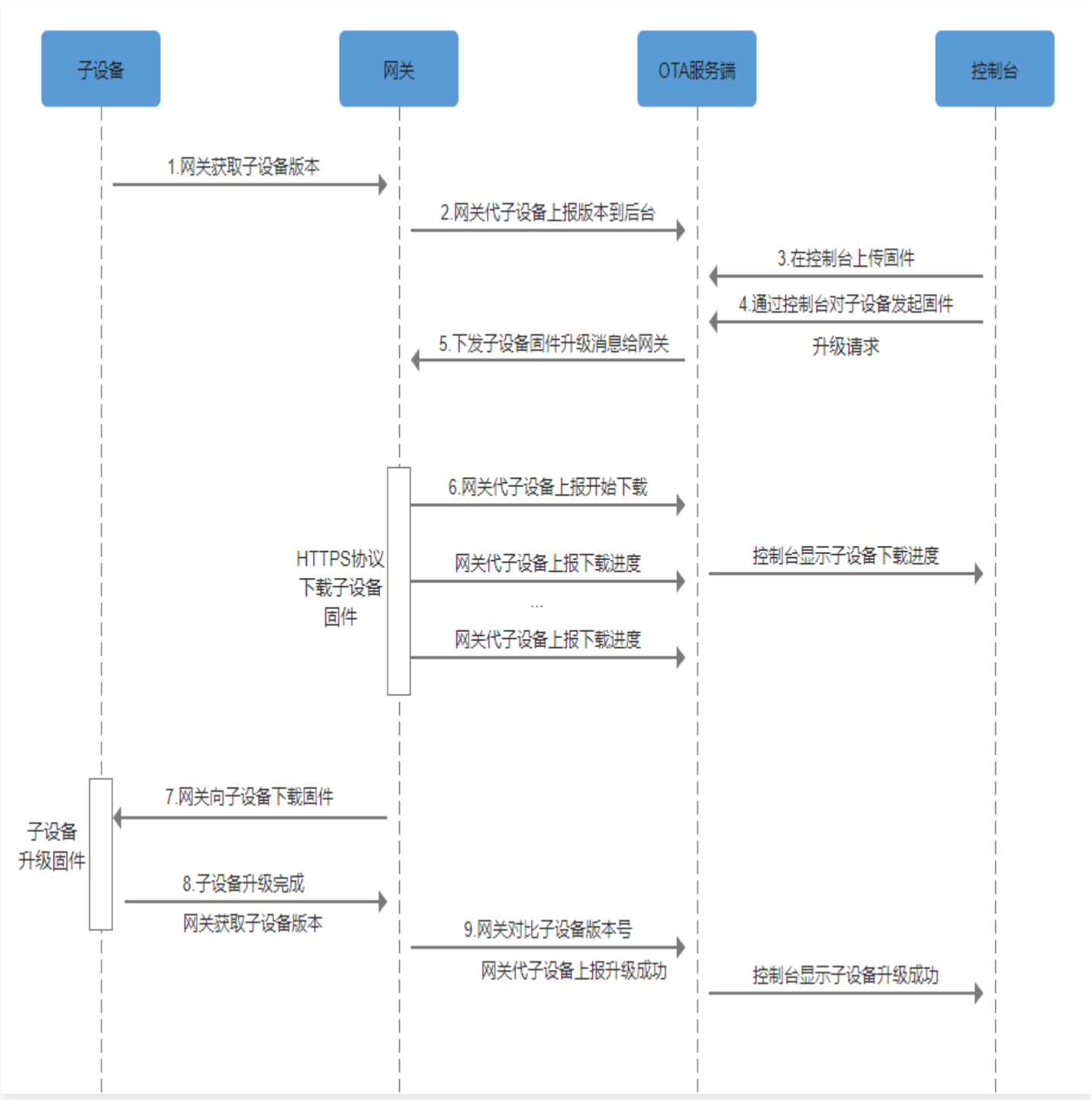
Sample code is as follows:

```
$ota/report/${productID}/${deviceName}
Used to publish (uplink) messages, the device reports the version number
and the download and upgrade progress to the cloud.
$ota/update/${productID}/${deviceName}
Used to subscribe (downlink) messages, the device receives upgrade
messages from the cloud.
```

Operation Process

Take MQTT as an example. The upgrade process diagram of the subdevice is as follows.

Note:
Specific operation steps for firmware upgrade. For details, see device firmware upgrade.



Resource Management Protocol

Last updated: 2025-04-27 17:52:23

Feature Overview

Resource management is mainly used by developers to issue standard device resources such as face recognition libraries, image libraries, and music libraries to end devices, achieving the upload and download of resource content between the platform and devices.
Implement this type of functionality requires using the following two Topics:

- Data Upstream Topic (for publishing): `$resource/up/service/${productid}/${devicename}` .
- Data downstream Topic (for subscription): `$resource/down/service/${productid}/${devicename}` .

Device Resource Upload

Step 1: Create an Upload Task for Resources on the Device Side

1. The device publishes a message to `$resource/up/service/${productid}/${devicename}` via the MQTT protocol to perform the creation of a device resource upload task. The message is in json format, with the following content:

```
{
	"type":"create_upload_task",
	"size":100,
	"name":"file",
	"md5sum":"************",
}
```

2. Creation succeeded. The backend returns the URLs for resource upload through `$resource/down/service/${productid}/${devicename}` . The message is in json format, with the following content:

```
{
	"type":"create_upload_task_rsp",
	"size":100,
	"name":"file",
	"md5sum":"************",
	"url":"https://iothub.cos.ap-guangzhou.myqcloud.com/********"
}
```

Step 2: Report the Upload Progress of Resources

1. Use HTTP PUT requests for resource upload, so the header needs to include the MD5 value (base64 encoded). During the resource upload process, the device reports the upload progress via `$resource/up/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
        "type":"report_upload_progress",
        "name":"file",
        "progress":{
                "state":"uploading",
                "percent":89,
                "result_code":0,
                "result_msg":""
        }
}
```

2. Progress report response, sent to the device via `$resource/down/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
        "type":"report_upload_progress_rsp",
        "result_code":0,
        "result_msg":"ok"
}
```

Platform Resource Distribution

Step 1: Query the Resource Download Link

1. The device reports messages via `$resource/up/service/${productid}/${devicename}` to query download tasks. The message is in json format, with the following content:

```
{
    "type":"get_download_task"
}
```

2. If a download task exists, send the result via `$resource/down/service/${productid}/${devicename}`. The message is in json format,

with the following content:

```
{
        "type":"get_download_task_rsp",
        "size":372338,
        "name":"AAAA",
        "md5sum":"a567907174*****3bb9a2bb20716fd97",
        "url":"https://iothub.cos.ap-guangzhou.myqcloud.com/********"
}
```

Step 2: Report the Download Progress of Resources

1. The resource download progress is reported via `$resource/up/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
        "type":"report_download_progress",
        "name":"file",
        "progress":{
                "state":"downloading",
                "percent":89,
                "result_code":0,
                "result_msg":""
        }
}
```

2. Progress report response, sent to the device via `$resource/down/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
        "type":"report_download_progress_rsp",
        "result_code":0,
        "result_msg":"ok"
}
```

Step 3: Report the Successful Download of Resources

1. The resource download progress is reported via `$resource/up/service/${productid}/${devicename}`. The message is in json format, with

the following content:

```
{
        "type":"report_download_progress",
        "name":"file",
        "progress":{
                "state":"done",
                "result_code":0,
                "result_msg":""
        }
}
```

2. Progress report response, sent to the device via `$resource/down/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
        "type":"report_download_progress_rsp",
        "result_code":0,
        "result_msg":"ok"
}
```

File Management Protocol

Last updated: 2025-04-27 17:52:37

Feature Overview

The file management functionality is used by manufacturers to complete file transfer between the device side, platform side, and application side. It is used for storing the files required by users during device usage after mass production. Since the permissions for these partial files do not belong to the device manufacturer for viewing and managing, they are implemented through the server file management topic message channel.

Use Case Examples

- User records voice in the mini program. The recording needs to be sent to the device for broadcasting.
- The access control camera takes a photo of the visitor when someone visits and views it on the mini program side.

To implement this type of functionality, you need to use the following two Topics:

- Data upstream Topic (for publishing): `$thing/up/service/${productid}/${devicename}`.
- Data downstream Topic (for subscription): `$thing/down/service/${productid}/${devicename}`.

Submitting Device File Version Information

1. The device reports the current file version information. The device side publishes a message to `$thing/up/service/${productid}/${devicename}` through the MQTT protocol to report the version number. The message is in json format, and the content is as follows:

```
{
    "method": "report_version",
    "request_id": "12345678",
    "report":{"resource_name": "123.wav", "version": "1.0.0",
"resource_type": "FILE"}
}
//method: message type
//resource_name: File Name
//version: file version number
//resource_type: file type, firmware (fw), file (file)
//Backend logic: receive messages and update the file version
information to the corresponding product/device.
```

In particular, if the submitted file list is empty, the cloud will reply with the recorded file list of the device. Based on this feature, the device can perform exception recovery operations on the device file list.

```
{
    "method": "report_version",
    "report":{
        "resource_list":[]
     }
}
```

2. After the server receives the file version information report, the server replies to the device side with the received version information through `$thing/down/service/${productid}/${devicename}`. The message is in json format, and the content is as follows:

```
{
    "method": "report_version_rsp",
    "result_code":0,
    "result_msg":"success",
    "resource_list":[
        {"resource_name": "audio_woman_mandarin", "version":
"1.0.0","resource_type": "FILE"},
        {"resource_name": "audio_woman_sichuanhua", "version":
"2.0.0","resource_type": "FILE"}
    ]
}
//method: message type
//result_code: version report result
//result_msg: version report result message
//resource_list: send back the received version information
//If the resource_list reported by the device is empty, the server
responds with the recorded list of resources.
```

Device File Download

1. The user calls the application-side API to upload files and create a file download task when using the mini program.
2. The device side will receive file update messages through the subscribed `$thing/down/service/${productid}/${devicename}`. The content of the file update message is as follows:

```
{
    "method": "update_resource",
    "resource_name": "audio_woman_sichuanhua",
    "resource_type": "FILE",
    "version": "1.0.0",
    "url": "https://ota-125*****59.cos.ap-guangzhou.myqcloud.com",
    "md5sum": "cc03e747a6afbbcbf8**********bee5",
    "file_size": 31242
}
// method: message type
// resource_name: file name
// resource_type: firmware (fw), file (file), dropdown selection in
the console
// version: upgrade version
// url: URL to download file
// md5sum: MD5 value of the file
// file_size: file size in bytes
```

3. After the device receives the file update message, it downloads resources according to the URL. During the download process, the device SDK will continuously report the download progress through `$thing/up/service/${productid}/${devicename}`. The reported content is as follows:

```
{
    "method": "report_progress",
    "report":{
        "progress": {
            "resource_name": "audio_woman_sichuanhua",
            "state":"downloading",
            "percent":"10",
            "result_code":"0",
            "result_msg":""
        },
      "version": "1.0.0"
    }
}
// method: message type
//resource_name: name of the downloading file.
// state: status is downloading
// percent: current download progress, percentage
```

4. When the device completes the file download, it needs to report a download result through `$thing/up/service/${productid}/${devicename}`. The content is as follows:

```
// Download success
{
    "method": "report_result",
    "report":{
        "progress":{
            "resource_name": "audio_woman_sichuanhua",
            "state":"done",
            "result_code":"0",
            "result_msg":"success"
        },
     "version": "1.0.0"
    }
}
// method: message type
// state: status is download complete
// result_code: download result, 0 for success, non-zero for failure
// result_msg: Specific description of the failure scenario
```

Device File Upload

1. The device requests the URL for file upload. The device side publishes a message to `$thing/up/service/${productid}/${devicename}` through the MQTT protocol to request the URL for file upload. The message is in json format, and the content is as follows:

```
{
    "method": "request_url",
    "request_id": "12345678",
    "report":{"resource_name": "123.wav", "version": "1.0.0",
"resource_type": "AUDIO"}
}
//method: message type
//resource_name: File Name
//version: file version number
//resource_type: file type
```

2. After the server receives the file version information report, the server returns the pre-signed cos url to the device side through Topic `$thing/down/service/${productid}/${devicename}`. The message is in json format, and the content is as follows:

```
{
    "method": "request_url_resp",
    "result_code":0,
    "result_msg":"success",
    "resource_url": "presigned_url_xxx",
    "resource_token": "123456abcdef",
    "request_id": "12345678"
}
//method: message type
//result_code: version report result
//result_msg: version report result message
//resource_url: cos pre-signed url
//resource_token: file token, subsequently can be based on the token
to map resource url
```

3. The device puts the resource to the corresponding cos url. After the upload is completed, it reports the upload result. The message is in json format, and the content is as follows:

```
{
    "method": "report_post_result",
    "report":{
        "progress":{
            "resource_token": "123456abcdef",
            "state":"done",
            "result_code":"0",
            "result_msg":"success"
        },
    }
}
//method: message type
//resource_name: file name
//state: upload result
//result_code: Result error code for upload, 0 for success
//result_msg: upload result message
//version: file version
```

File Deletion

1. The device side will receive file deletion messages through the subscribed Topic $thing/down/service/${productID}/${deviceName}. The content of the file deletion message is as follows:

```
{
    "method": "del_resource",
    "resource_name": "audio_woman_sichuanhua",
    "resource_type": "FILE",
    "version": "1.0.0"
}
// method: message type is
// resource_name: file name
```

2. When the device completes the file download, it needs to report a deletion result through Topic `$thing/up/service/${productID}/${deviceName}`. In particular, if the file to be deleted does not exist on the device side (such as due to device flashing), it is recommended to reply that the deletion is successful. Otherwise, the records of this file on the device side and in the cloud will always be inconsistent. The content is as follows:

```
{
    "method": "del_result",
    "report":{
        "progress":{
            "resource_name": "audio_woman_sichuanhua",
            "state":"done",
            "result_code":"0",
            "result_msg":"success"
        },
    "version": "1.0.0"
    }
}
// method: message type
// state: deletion complete
// result_code: deletion result, 0 for success, non-zero for failure
// result_msg: specific description information of the failure
scenario
```

Micro Call TWecall Protocol

Last updated: 2025-04-27 17:52:50

Feature Overview

Twecall is used for device calls to WeChat to perform audio and video calls, mainly applied to products such as cameras, locks, doorbells, and indoor screens.
To implement this feature, you need to use the following two Topics:

- Data upstream Topic (for publishing): `$twecall/up/service/${productid}/${devicename}`
- Data downstream Topic (for subscription): `$twecall/down/service/${productid}/${devicename}`.

Retrieve the snTicket

1. The device publishes a message to `$twecall/up/service/${productid}/${devicename}` via the MQTT protocol to obtain it. The message is in json format, with the following content:

```
{
    "method":"get_wechat_sn_ticket",
    "clientToken":"123",
    "timestamp":1628646783,
    "params":{
        "ModelId":"111",
        "miniProgramAppId":"111"
       }
}
//ModelId: WeChat public platform application
//miniProgramAppId: WeChat appid of the mini program
```

2. After the server receives the request reporting for retrieval, it replies to the device with the snTicket information through `$twecall/down/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
    "method":"get_wechat_sn_ticket_reply",
    "clientToken":"123",
    "code":0, // 0:normal, 1:exception
    "status":"", //error information, empty under normal circumstances
    "params":{
        "snTicket":"111",
```

```
        }
}
```

Activate the TWecall Feature on the Device

1. The device will report the `$twecall/up/service/${productid}/${devicename}` message to activate the Twecall license. The request is as follows:

```
{
    "method":"active_device_voip_license",
    "clientToken":"123",
    "timestamp":1628646783,
    "params":{
        "pkgType":1, // 0-trial; 1-security scenario; 2-wear scenario;
3-entertainment scenario; 4-intercom and other scenarios
        "miniProgramAppId":"111",
        "modelId":"modelId1"
       }
}
//ModelId: WeChat public platform application
//miniProgramAppId: WeChat appid of the mini program
```

2. After the server receives the request reporting for retrieval, it replies to the device with the activation status through `$twecall/down/service/${productid}/${devicename}`. The message is in json format, with the following content:

```
{
    "method":"active_device_voip_license_reply",
    "clientToken":"123",
    "code":0, // 0:normal, 1:exception
    "status":"", //error information, normally empty
    "params":{
        }
}
```

Query Device TWecall Activation Details

1. The device publishes a message to `$twecall/up/service/${productid}/${devicename}` through the MQTT protocol to query the device activation status. The message is in json format, with the following content:

```
{
    "method":"get_voip_device_active_info",
    "clientToken":"123",
    "timestamp":1628646783,
    "params":{
        "miniProgramAppId":"111",
        "modelId":"modelId1"
       }
}
//ModelId: WeChat public platform application
//miniProgramAppId: WeChat appid of the mini program
```

2. After the server receives the query information reporting, it returns the device activation status details to the device through the Topic `$twecall/down/service/${productid}/${devicename}` . The message is in json format, with the following content:

```
{
    "method":"get_voip_device_active_info_reply",
    "clientToken":"123",
    "code":0, // 0:normal, 1:exception
    "status":"", //error information, empty under normal circumstances
    "params":{
        "modelId":"modelId1",
        "sn":"sn1",
        "ExpireTime":1630425600
        }
}
```

SDK Description and Download

Last updated: 2025-04-27 17:53:04

Tencent Cloud IoT Explorer provides Device SDKs in multiple language versions for different device development scenarios for customer use. See Developer Guide.

C SDK Code Hosting

- Starting from V3.1.0, the C Device SDK code is hosted independently on Github.
- Download the latest version of C SDK.
- C SDK versions prior to SDK 3.1.0 access here.

> ⚠ **Notes:**
> Versions prior to V3.1.0 have a different code GitHub path compared with versions V3.1.0 and later. Meanwhile, there are large differences in the interaction protocol with the platform.

AT SDK Code Hosting

After creating products and devices on the IoT explorer platform <1>, select the development method based on the customized MQTT AT module. The <3>AT SDK</3> code on the MCU side will be automatically generated, and the data templates and events created on the platform will generate corresponding configurations and initialization code.

Android SDK Code Hosting

Starting from V3.1.0, the Android SDK uses independent Github to host the code.

Java SDK Code Hosting

Java SDK uses independent Github to host the code.

Python SDK Code Hosting

Starting from V1.0.0, the Python device-side SDK code is hosted on Github.

Device-Side SDK Usage Reference
C SDK Usage Reference
Usage Overview

Last updated: 2025-04-27 17:53:36

Tencent Cloud IoT Device SDK for C relies on a secure and high-performance data channel to provide developers in the IoT domain with the ability to quickly integrate devices with the cloud and perform two-way communication with the cloud.

C SDK Applicable Scope

The C SDK uses modularized design, separates the core protocol service from the Hardware Abstraction Layer, and provides flexible configuration options and multiple compilation methods. It is suitable for development platforms and usage environments of different devices.

Devices with Network Communication Capability and Using Linux/Windows Operating System

- For devices with network communication capabilities and using standard Linux/Windows systems, such as PCs/servers/gateway devices, and more advanced embedded devices, such as Raspberry Pi, etc., you can directly compile and run the SDK on the device.
- For embedded Linux devices that require cross-compilation, if the toolchain in the development environment has glibc or similar libraries, it can provide system calls including socket communication, select synchronous IO, dynamic memory allocation, acquisition time/sleep/random number/print function, and critical data protection, such as the Mutex mechanism (only when multi-threading is required). Just perform simple modifications (for example, modifying the settings of the cross-compiler in CMakeLists.txt or make.settings) to compile and run the SDK.

Devices with Network Communication Capability and Adopting RTOS System

- For IoT devices with network communication capabilities and using RTOS, the C SDK needs to perform porting adaptation work for different RTOS. Currently, the C SDK has been adapted to multiple RTOS platforms for the Internet of Things, including FreeRTOS/RT-Thread/TencentOS tiny.
- When porting the SDK on an RTOS device, if the platform provides a C runtime library similar to newlib and an embedded TCP/IP stack similar to lwIP, the porting adaptation work can also be easily completed.

Devices with MCU+ Communication Module

- For MCUs without network communication capabilities, generally use the MCU + Communication Module approach. Communication modules (including Wi-Fi/2G/4G/NB-IoT) generally provide an AT Command Protocol based on the serial port for MCUs to perform network communication. For this type of scenario, the C SDK encapsulates the AT-socket network layer. The core protocol and service layer above the network layer do not need to be ported. And it provides HAL implementations based on two methods: FreeRTOS and nonOS.
- Besides, Tencent Cloud IoT also provides a dedicated AT Instruction Set. If the Communication Module implements this instruction set, device connectivity and communication become simpler, requiring less code volume. For this type of scenario, please refer to the MCU AT SDK dedicated to Tencent Cloud.

Introduction to SDK Directory Structure

The introduction to the directory structure and top-level files is as follows:

Name	Description
CMakeLists.txt	cmake compile description file.
CMakeSettings.json	cmake Configuration File under visual studio.
cmake_build.sh	Compilation script using cmake under Linux.
make.settings	Makefile configuration file for direct compilation under Linux.
Makefile	Direct compilation using Makefile under Linux.
device_info.json	Device Information File, from which device information will be parsed when DEBUG_DEV_INFO_USED=OFF.
docs	Documentation directory, user guide for using the SDK on different platforms.
external_libs	Third-party software package component, such as mbedtls.
samples	Application example.
include	External header files for users.
platform	Source code files related to the platform. Currently provides implementations for different OS

	(Linux/Windows/FreeRTOS/nonOS), TLS (mbedtls), and AT Module.
sdk_src	SDK Core Communication Protocol and Service Code.
tools	Supportive compilation and code generation script tools for SDK.

SDK Compilation Method Description

C SDK supports three compilation methods:

- cmake method.
- Makefile method.
- Code extraction method.

For detailed instructions on compilation methods and compilation configuration options, please refer to Compilation Configuration Instructions and Compilation Environment Description.

SDK Sample Experience

The samples directory of C SDK contains examples of using each functionality. For detailed instructions on running examples, please refer to all documents in the SDK documentation directory.
For a quick experience of data interaction of Thing Model in IoT Explorer, please refer to Quick Start for Smart Light.

Compilation Configuration Instructions

Last updated: 2025-04-27 17:53:51

This document describes the compilation method and compilation configuration options of the C SDK, and introduces the compilation environment setup and compilation examples in Linux and Windows development environments.

C SDK Compilation Method Description

The C SDK supports the following compilation methods.

cmake Method

- Recommended for use cmake as a cross-platform compilation tool, supporting compilation in Linux and Windows development environments.
- Use cmake method, adopting CMakeLists.txt as the compilation configuration options input file.

Makefile

- For environments that do not support cmake, use the way of direct compilation with Makefile.
- Use the make.settings file as the compilation configuration options input file in the Makefile method. After modification, execute make.

Code Extraction Method

- This method allows you to select features as needed, extracting relevant code into a separate folder. The hierarchical directory of the code in the folder is simple, making it convenient for users to copy and integrate into their own development environment.
- This method relies ON the cmake tool. Configure the switches of relevant feature modules in CMakeLists.txt and set EXTRACT_SRC to ON. Run the following commands in a Linux environment:

```
mkdir build
cd build
cmake ..
```

You can find the relevant code files in output/qcloud_iot_c_sdk. The directory hierarchy is as follows:

```
qcloud_iot_c_sdk
├── include
│   ├── config.h
│   ├── exports
├── platform
└── sdk_src
  └── internal_inc
```

- The include directory contains APIs and variable parameters for users provided by the SDK. Among them, config.h is a compilation macro generated according to compilation options.
- The platform directory contains platform-related code, which can be modified and adapted according to the specific circumstances of the device.
- sdk_src is the core logic and protocol-related code of the SDK. Generally, no modification is required. Among them, internal_inc is the header file for internal use of the SDK.

Description
Users can copy qcloud_iot_c_sdk to the compilation development environment of their target platform and modify the compilation options as appropriate.

C-SDK Option Description

Compilation Configuration Options

Most of the following configuration options are applicable to cmake and make.setting. The ON value in cmake corresponds to y in make.setting, and OFF corresponds to n.

Name	cmake value	Overview
BUILD_TYPE	release/debug	• release: Disable IOT_DEBUG messages and compile the output under the release directory. • debug: Enable IOT_DEBUG messages and compile the output under the debug directory.
EXTRACT_SRC	ON/OFF	Code extraction feature switch, only applicable to cmake usage.

COMPILE_TOOLS	gcc	Support gcc and msvc, or cross-compilers. For example, arm-none-linux-gnueabi-gcc.
PLATFORM	Linux	Including Linux/Windows/Freertos/Nonos.
FEATURE_OTA_COMM_ENABLED	ON/OFF	OTA feature enable switch.
FEATURE_AUTH_MODE	KEY/CERT	Access authentication method.
FEATURE_AUTH_WITH_NOTLS	ON/OFF	• OFF: Enable TLS. • ON: Disable TLS.
FEATURE_EVENT_POST_ENABLED	ON/OFF	Event feature enable switch.
FEATURE_ACTION_ENABLED	ON/OFF	Behavior function enable switch.
FEATURE_DEBUG_DEV_INFO_USED	ON/OFF	Device information acquisition source switch.
FEATURE_SYSTEM_COMM_ENABLED	ON/OFF	Get background time switch.
FEATURE_DEV_DYN_REG_ENABLED	ON/OFF	Device dynamic registration switch.
FEATURE_LOG_UPLOAD_ENABLED	ON/OFF	Log reporting switch.

The configuration items for using the SDK, AT framework and General TCP Module are as follows:

Name	cmake value	Overview
FEATURE_AT_TCP_ENABLED	ON/OFF	AT Module TCP Function Switch.
FEATURE_AT_UART_RECV_IRQ	ON/OFF	AT Module Interrupt Reception Function Switch.
FEATURE_AT_OS_USED	ON/OFF	AT Module Multithreading Function Switch.
FEATURE_AT_DEBUG	ON/OFF	AT Module Debugging Function Switch.

There is a dependency relationship between configuration options. Some configuration options are valid only when the value of the dependent option is a valid value. The main ones are as follows:

Name	Dependent option	Effective value
FEATURE_AUTH_WITH_NOTLS	FEATURE_AUTH_MODE	KEY
FEATURE_AT_UART_RECV_IRQ	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_OS_USED	FEATURE_AT_TCP_ENABLED	ON
FEATURE_AT_DEBUG	FEATURE_AT_TCP_ENABLED	ON

Equipment Information Options

After a device is created in the Tencent Cloud IoT console, you need to configure its information (ProductID/DeviceName/DeviceSecret/Cert/Key file) in the SDK for proper functioning. During the development phase, the SDK offers two ways to store device information:

- Stored in code (compilation option DEBUG_DEV_INFO_USED = ON), modify the equipment information in `platform/os/xxx/HAL_Device_xxx.c`. This method can be used on platforms without a file system.
- Stored in configuration files (compilation option DEBUG_DEV_INFO_USED = OFF), modify the device information in the `device_info.json` file. In this method, there is no need to recompile the SDK to change the device information. This approach is recommended for development on Linux/Windows platforms.

Compilation Environment Description

Last updated: 2025-04-27 17:54:04

The C SDK provides adaptation guides for integrating and using the IoT Explorer on multiple platforms. The example Demo can be used to compile the SDK on Linux and Windows for a quick trial.

Compiling Environment

Linux Compiling Environment

Use cmake + gcc to compile the SDK on Linux. The cmake version should be v3.5 or higher. The default installed cmake version is low. If the compilation fails, click Download and refer to Installation Instruction to download and install the specified version of cmake.

```
$ sudo apt-get install -y build-essential make git gcc cmake
```

Windows Compiling Environment

Use the cmake tool in Visual Studio 2019 to compile the SDK on Windows.
Follow these steps to obtain and install the Visual Studio 2019 development environment:

1. Please visit Visual Studio download website, download and install Visual Studio 2019. The version downloaded and installed in this document is v16.2 Community.

2. Click **desktop development using C++** and ensure that "CMAKE tool for Windows" is checked.

Access Guide

- For compilation and running on the Linux platform, please see Linux Platform Integration Guide.
- For compilation and running on the Windows platform, please see Windows Platform Access Guide.

API and Variable Parameter Descriptions

Last updated: 2025-04-27 17:54:17

The header files for API function declarations, constants, and variable parameter definitions that are provided for user invocation in the device-side C SDK are located under the include directory. This document mainly describes the variable parameters and API functions under this directory.

Variable Parameter Configuration

The C SDK is based on the MQTT protocol. It can configure appropriate parameters according to specific scenario requirements to meet the needs of actual business operations. Variable access parameters include:

1. The timeout period for MQTT blocking calls (including connection, subscribe, publish, etc.), unit: millisecond. Recommend 5000 milliseconds.
2. The buffer size for sending and receiving messages of the MQTT protocol defaults to 2048 bytes, supports up to 16KB.
3. MQTT heartbeat message sending interval, maximum value: 690 seconds, unit: millisecond.
4. Maximum reconnection waiting time, unit: millisecond. When a device disconnects and attempts to reconnect, if it fails, the waiting time will double. If it exceeds this maximum waiting time, the reconnection will be exited.

Modify the `include/qcloud_iot_export_variables.h` file. The macro definition can modify the parameter configuration of the corresponding access. After modification, the SDK needs to be recompiled. Sample code is as follows:

```
/* default MQTT/CoAP timeout value when connect/pub/sub (unit: ms) */
#define QCLOUD_IOT_MQTT_COMMAND_TIMEOUT                                 (5 *
1000)

/* default MQTT keep alive interval (unit: ms) */
#define QCLOUD_IOT_MQTT_KEEP_ALIVE_INTERNAL                             (240
* 1000)

/* default MQTT Tx buffer size, MAX: 16*1024 */
#define QCLOUD_IOT_MQTT_TX_BUF_LEN
(2048)

/* default MQTT Rx buffer size, MAX: 16*1024 */
```

```
#define QCLOUD_IOT_MQTT_RX_BUF_LEN
(2048)

/* default COAP Tx buffer size, MAX: 1*1024 */
#define COAP_SENDMSG_MAX_BUFLEN
(512)

/* default COAP Rx buffer size, MAX: 1*1024 */
#define COAP_RECVMSG_MAX_BUFLEN
(512)

/* MAX MQTT reconnect interval (unit: ms) */
#define MAX_RECONNECT_WAIT_INTERVAL                                    (60
* 1000)
```

API Function Description

The main features provided by C SDK v3.1.0 and the corresponding API descriptions are for customers to write business logic, with more detailed explanations. For example: API parameters and return values can be viewed in the comments of header files such as SDK code `include/exports/qcloud_iot_export_*.h`.

Thing Model API

Thing Model Protocol and feature introduction, please see Thing Model Protocol.

No.	Function Name	Description
1	IOT_Template_Construct	Construct a Thing Model client Data_template_client and connect to the MQTT cloud service.
2	IOT_Template_Destroy	Close the Data_template MQTT connection and terminate the Data_template Client.
3	IOT_Template_Yield	Perform MQTT message reading, message processing, timeout request, heartbeat packet, and reconnection state management tasks in the current Thread Context.
4	IOT_Template_Publish	The Thing Model client publishes an MQTT message.
5	IOT_Template_Subscribe	The Thing Model client subscribes to an MQTT topic.

6	IOT_Template_Unsubscribe	The Thing Model client unsubscribes from the subscribed MQTT topic.
7	IOT_Template_IsConnected	View whether the MQTT of the current Thing Model client is connected.

- **Model attribute API**

No.	Function Name	Description
1	IOT_Template_Register_Property	Register the model attributes of the current device.
2	IOT_Template_UnRegister_Property	Delete the registered model attributes.
3	IOT_Template_Report	Submit Thing Model attribute data asynchronously.
4	IOT_Template_Report_Sync	Synchronously report the data of the device model attribute.
5	IOT_Template_GetStatus	Retrieve Thing Model attribute data asynchronously.
6	IOT_Template_GetStatus_sync	Synchronize and obtain device model attribute data.
7	IOT_Template_Report_SysInfo	Submit system information asynchronously.
8	IOT_Template_Report_SysInfo_Sync	Synchronize and submit system information.
9	IOT_Template_JSON_ConstructSysInfo	Construct the system information to be reported.
10	IOT_Template_ControlReply	Reply to the received device model attribute control messages.
11	IOT_Template_ClearControl	Delete the device model attribute control messages and cooperate with IOT_Template_GetStatus to obtain and use the control messages.

- **Thing Model Event Interface**

No.	Function Name	Description
1	IOT_Post_Event	Report device model events, import events, and the SDK completes the construction of event messages.
2	IOT_Post_Event_Raw	Report device model events, import data that meets the event message format, and the SDK completes the reporting.
3	IOT_Event_setFlag	Set event flag. The SDK supports 10 events by default and can be expanded.
4	IOT_Event_clearFlag	Clear event flags.
5	IOT_Event_getFlag	Retrieve event flags.

- **Thing Model Behavior Interface**

No.	Function Name	Description
1	IOT_ACTION_REPLY	Reply to the Thing Model behavior message.

- **Multi-threaded environment usage instructions**
  SDK has the following precautions for use in a multi-threaded environment:
  - Multi-threaded calls to IOT_Template_Yield, IOT_Template_Construct and IOT_Template_Destroy are not allowed.
  - IOT_Template_Yield, as a function to read and process MQTT messages from the socket, should ensure a certain execution time and avoid being suspended or preempted for a long time.

OTA API

For more information on the OTA firmware download feature, please see Device Firmware Update.

No.	Function Name	Description
1	IOT_OTA_Init	Initialize the OTA module. The client needs to initialize MQTT/COAP first before calling this API.
2	IOT_OTA_Destroy	Release resources related to the OTA module.

3	IOT_OTA_ReportVersion	Report local firmware version information to the OTA server.
4	IOT_OTA_IsFetching	Check if it is in the state of downloading firmware.
5	IOT_OTA_IsFetchFinish	Check if the firmware has been downloaded completely.
6	IOT_OTA_FetchYield	Retrieve firmware from a remote server with a specific timeout period.
7	IOT_OTA_Ioctl	Retrieve specified OTA information.
8	IOT_OTA_GetLastError	Retrieve the last error code.
9	IOT_OTA_StartDownload	Establish an HTTP connection with the firmware server based on the obtained firmware update address and the offset of the local firmware information (whether to resume from the breakpoint).
10	IOT_OTA_UpdateClientMd5	Calculate the MD5 of the local firmware before resuming from the breakpoint.
11	IOT_OTA_ReportUpgradeBegin	Report the status of the upcoming upgrade to the server before performing the firmware upgrade.
12	IOT_OTA_ReportUpgradeSuccess	Report the successfully upgraded status to the server after the firmware upgrade is successful.
13	IOT_OTA_ReportUpgradeFail	Report the upgrade failure status to the server after the firmware upgrade fails.

Log API

For a detailed description of the device log reporting functionality to the cloud, please refer to the device log reporting feature section in the IoT communication platform document under the SDK docs directory.

No.	Function Name	Description
1	IOT_Log_Set_Level	Set the print level of the SDK log.
2	IOT_Log_Get_Level	Return the print level of the SDK log.

3	IOT_Log_Set_MessageHandler	Set the log callback function and redirect the SDK log to other output methods.
4	IOT_Log_Init_Uploader	Enable the feature of SDK log reporting to the cloud and initialize resources.
5	IOT_Log_Fini_Uploader	Disable the feature of SDK log reporting to the cloud and release resources.
6	IOT_Log_Upload	Report the SDK running log to the cloud.
7	IOT_Log_Set_Upload_Level	Set the report level of the SDK log.
8	IOT_Log_Get_Upload_Level	Return the report level of the SDK log.
9	Log_d/i/w/e	Print the API for adding SDK logs by level.

System Time API

No.	Function Name	Description
1	IOT_Get_SysTime	Retrieve the time of the IoT Hub backend system. Currently only support the synchronization feature via MQTT channel.

Thing Model Code Generation

Last updated：2025-04-27 17:54:32

this document introduces how to generate product model code based on the thing model created on the IoT Development Platform IoT Explorer.
There are following 3 steps to generate the Product Model code.

Step 1. Create an Object Model

Create an object model. For details, see Thing Model document.

Step 2: Export the Thing Model Description File

The Thing Model description file is a JSON file that describes the attributes, events, and other information of the product definition.
click **View Thing Model JSON**. After confirming the content, click the download icon to export the JSON file.

Step 3: Generate Product Model Code

1. Perform the following command to copy the downloaded JSON file to the tools directory.

```
./codegen.py -c xx/config.json -d  ../targetdir/
```

2. Generate the Thing Model and event configuration files for the defined product according to the JSON file in the target directory, and copy this generated configuration file to the same-level directory of data_template_sample.c.

```
./codegen.py -c light.json -d ../samples/data_template/
Load light.json file successfully
File../samples/data_template/data_config.c generated successfully
File../samples/data_template/events_config.c generated successfully
```

> **Note:**
> data_template_sample.c describes a common Thing Model processing framework. You can add business logic based on this framework.
> Smart light example. The light_data_template_sample.c is a scenario example based on this framework.

Model Application Development

Last updated: 2025-04-27 17:54:46

The Thing Model example `data_template_sample.c` has implemented a general processing framework for data, event sending and receiving, and response. You can develop business logic based on this example. The entry functions added for upstream and downstream business logic are `deal_up_stream_user_logic` and `deal_down_stream_user_logic` respectively. For details, see the scenario example `light_data_template_sample.c` of smart light to add business processing logic.

Downstream Business Logic Implementation

The server downlink data has been parsed into JSON data by the SDK according to the Thing Model Protocol. ProductDataDefine is a template structure generated in step 3 based on the product Thing Model defined on the platform, and its member variables are composed of the defined attributes. The attribute data pointed to by the input parameter pData has been parsed by the SDK according to the Thing Model Protocol from the server downlink data. In the downstream logic processing function, users can directly use the parsed data to add business logic.

- User performs corresponding business logic processing based on parsed device model data (pData).

```
/*The business logic of downstream data that users need to implement.
The business logic remains to be implemented by users.*/
static void deal_down_stream_user_logic(void *client,
ProductDataDefine   * pData)
{
    Log_d("someting about your own product logic wait to be done");
}
```

- Sample code is as follows:

```
/*Attribute Data Template for smart light*/
typedef struct _ProductDataDefine {
    TYPE_DEF_TEMPLATE_BOOL m_light_switch;
    TYPE_DEF_TEMPLATE_ENUM m_color;
    TYPE_DEF_TEMPLATE_INT  m_brightness;
    TYPE_DEF_TEMPLATE_STRING m_name[MAX_STR_NAME_LEN+1];
} ProductDataDefine;
/*Sample light control processing logic*/
```

```
static void deal_down_stream_user_logic(void *client,ProductDataDefine
*light)
{
    int i;
    const char * ansi_color = NULL;
    const char * ansi_color_name = NULL;
    char brightness_bar[]       = "||||||||||||||||||||";
    int brightness_bar_len = strlen(brightness_bar);
    /*light color*/
    switch(light->m_color) {
        case eCOLOR_RED:
            ansi_color = ANSI_COLOR_RED;
            ansi_color_name = " RED ";
            break;
        case eCOLOR_GREEN:
            ansi_color = ANSI_COLOR_GREEN;
            ansi_color_name = "GREEN";
            break;
        case eCOLOR_BLUE:
            ansi_color = ANSI_COLOR_BLUE;
            ansi_color_name = " BLUE";
            break;
        default:
            ansi_color = ANSI_COLOR_YELLOW;
            ansi_color_name = "UNKNOWN";
            break;
    }
    /* light brightness display bar */
    brightness_bar_len = (light->m_brightness >= 100)?
brightness_bar_len:(int)((light->m_brightness *
brightness_bar_len)/100);
    for (i = brightness_bar_len; i < strlen(brightness_bar); i++) {
        brightness_bar[i] = '-';
    }
    if(light->m_light_switch){
        /* Show according to the control parameter when the light is
enabled */
        HAL_Printf( "%s[  lighting  ]|[color:%s]|[brightness:%s]|
[%s]\n" ANSI_COLOR_RESET,\
                    ansi_color,ansi_color_name,brightness_bar,light-
>m_name);
    }else{
        /* Show light off */
```

```
        HAL_Printf( ANSI_COLOR_YELLOW"[  light is off ]|[color:%s]|
[brightness:%s]|[%s]\n" ANSI_COLOR_RESET,\
                    ansi_color_name,brightness_bar,light->m_name);
    }
#ifdef EVENT_POST_ENABLED
    if(eCHANGED == sg_DataTemplate[0].state){
        if(light->m_light_switch){
            memset(sg_message, 0, MAX_EVENT_STR_MESSAGE_LEN);
            strcpy(sg_message,"light on");
            sg_status = 1;
        }else{
            memset(sg_message, 0, MAX_EVENT_STR_MESSAGE_LEN);
            strcpy(sg_message,"light off");
            sg_status = 0;
        }
        IOT_Event_setFlag(client, FLAG_EVENT0); /* Set event when the
switch status of the light changes. The set event will be reported in
eventPostCheck. */
    }
#endif
}
```

Uplink Business Logic Implementation

- The device end adopts certain policies to monitor and process the device data attributes as required by the business scenario.
- Users can update the attributes that need to be reported to the input parameter pReportDataList attribute in `deal_up_stream_user_logic`, including the report list, the number of attributes to be reported, and the example processing framework of the Thing Model. In `IOT_Template_JSON_ConstructReportArray`, the attribute data list will be processed as the Protocol Format of the Thing Model. `IOT_Template_Report` sends data to the server.

Below is the sample code:

```
/* The user modifies the attribute value according to business, then
sets the attribute status to eCHANGED */
static void _refresh_local_property(void)
{
    //add your local property refresh logic
}
/* The business logic of upstream data that users need to implement.
This is just an example. */
```

```
static int deal_up_stream_user_logic(DeviceProperty *pReportDataList[],
int *pCount)
{
    int i, j;

        /*Monitor whether local data needs to be updated*/
        _refresh_local_property();

        /*Update the adjusted attributes to the report list*/
     for (i = 0, j = 0; i < TOTAL_PROPERTY_COUNT; i++) {
        if(eCHANGED == sg_DataTemplate[i].state) {
            pReportDataList[j++] = &(sg_DataTemplate[i].data_property);
            sg_DataTemplate[i].state = eNOCHANGE;
        }
    }
    *pCount = j;

    return (*pCount > 0)?QCLOUD_RET_SUCCESS:QCLOUD_ERR_FAILURE;
}
```

Device Information Storage

Last updated: 2025-04-27 17:55:00

Tencent Cloud IoT Explorer allocates a unique identifier, ProductID, to each created product. Users can customize the DeviceName to identify devices. The legitimacy of devices is verified using the product identification + device identification + device certificate/key. And these device identity information needs to be stored on the device side and will be used when applying for triplet or quadruple information for device authentication. The C SDK provides APIs and reference implementations for reading and writing device information, which can be adapted based on the actual situation.

Device Identity Information

- Certificate devices must have quadruple information to pass the platform's security authentication: product ID (ProductID), device name (DeviceName), device certificate file (DeviceCert), and device private key file (DevicePrivateKey). Among them, the certificate file and private key file are generated by the platform and have a one-to-one correspondence.
- Key devices must have triplet information to pass the platform's security authentication: product ID (ProductId), device name (DeviceName), and device secret (DeviceSecret), among which the device secret is generated by the platform.

Device Identity Information Burn-In

Device information burning is divided into preset burning and dynamic burning. There are some differences in the convenience and security between the two as follows.

Preset Burning

After creating a product, create devices one by one in IoT Explorer Console or through TencentCloud API, and obtain the corresponding device information. Burn the above quadruple or triplet information into non-volatile medium during the specific stage of device production. The Device SDK runtime reads the stored device information to perform device authentication.

Dynamic Burning

Preset burning needs to execute personalized production actions in the mass production process, which impacts production efficiency. To add convenience of application, the platform supports dynamic burning via a way.

- Implementation approach: After product creation, enable the dynamic registration feature of the product, and the product key (ProductSecret) will be generated. All devices under

the same product can burn unified product information during the production process, namely product ID (ProductId) and product key (ProductSecret). After the device leaves the factory, the device identity information is retrieved via the dynamic registration method and saved, which will be used when applying for triplet or quadruple information for device authentication.

- Device name (DeviceName) generation for dynamic burning:
  - If dynamic registration and automatic device creation are enabled at the same time, the device name can be generated by the device itself, but it must ensure that the device names under the same product ID (ProductId) are non-repeating, generally named IMEI or MAC address.
  - If the dynamic registration is enabled while the automatic device creation is not enabled, the device name needs to be entered into the platform in advance. During the dynamic registration of the device, the applied device name will be verified as to whether it is a legally entered device name. This method can reduce the security risk after the product key leakage to a certain extent.

> ⚠ **Note:**
> Dynamic registration needs to ensure the security of the product key (ProductSecret), otherwise it will generate a potential security risk.

Device Information Read and Write API

The SDK provides a HAL interface for device information read and write, which must be implemented. You can refer to the implementation of device information read and write in HAL_Device_Linux.c on the Linux platform.
Device information HAL interface:

HAL_API	Overview
HAL_SetDevInfo	Write device information.
HAL_GetDevInfo	Read device information.

Device Information Configuration in Development Phase

After creating the device, you need to configure the device information (`ProductID/DeviceName/DeviceSecret/Cert/Key` file) in the SDK to run the demo correctly.
During the development phase, the SDK offers two ways to store device information:

1. Stored in code (compilation option DEBUG_DEV_INFO_USED = ON), modify the device information in `platform/os/xxx/HAL_Device_xxx.c` . This approach can be used on platforms without a file system.

```
/* product Id  */
static char sg_product_id[MAX_SIZE_OF_PRODUCT_ID + 1]     =
"PRODUCT_ID";
/* device name */
static char sg_device_name[MAX_SIZE_OF_DEVICE_NAME + 1]   =
"YOUR_DEV_NAME";
#ifdef DEV_DYN_REG_ENABLED
/* product secret for device dynamic Registration  */
static char sg_product_secret[MAX_SIZE_OF_PRODUCT_SECRET + 1]   =
"YOUR_PRODUCT_SECRET";
#endif
#ifdef AUTH_MODE_CERT
/* public cert file name of certificate device */
static char sg_device_cert_file_name[MAX_SIZE_OF_DEVICE_CERT_FILE_NAME
+ 1]       = "YOUR_DEVICE_NAME_cert.crt";
/* private key file name of certificate device */
static char
sg_device_privatekey_file_name[MAX_SIZE_OF_DEVICE_SECRET_FILE_NAME +
1] = "YOUR_DEVICE_NAME_private.key";
#else
/* device secret of PSK device */
static char sg_device_secret[MAX_SIZE_OF_DEVICE_SECRET + 1] =
"YOUR_IOT_PSK";
#endif
```

2. Stored in configuration files (compilation option DEBUG_DEV_INFO_USED = OFF), modify the device information in the `device_info.json` file. In this method, there is no need to recompile the SDK to change the device information. This approach is recommended for development on Linux/Windows platforms.

```
{
    "auth_mode":"KEY/CERT",

    "productId":"PRODUCT_ID",
    "productSecret":"YOUR_PRODUCT_SECRET",
    "deviceName":"YOUR_DEV_NAME",

    "key_deviceinfo":{
        "deviceSecret":"YOUR_IOT_PSK"
    },

    "cert_deviceinfo":{
```

```
        "devCertFile":"YOUR_DEVICE_CERT_FILE_NAME",
        "devPrivateKeyFile":"YOUR_DEVICE_PRIVATE_KEY_FILE_NAME"
    },

    "subDev":{
        "sub_productId":"YOUR_SUBDEV_PRODUCT_****",
        "sub_devName":"YOUR_SUBDEV_DEVICE_****"
    }
}
```

Application Example

- Initialize connection parameters

```
static DeviceInfo sg_devInfo;
static int _setup_connect_init_params(MQTTInitParams* initParams)
{
    int ret;

    ret = HAL_GetDevInfo((void *)&sg_devInfo);
    if(QCLOUD_ERR_SUCCESS != ret){
        return ret;
    }

    initParams->device_name = sg_devInfo.device_****;
    initParams->product_id = sg_devInfo.product_****;
      ......
}
```

- Generate authentication parameters for key devices

```
static int _serialize_connect_packet(unsigned char *buf, size_t
buf_len, MQTTConnectParams *options, uint32_t *serialized_len) {
            ......
            ......
    int username_len = strlen(options->client_id) +
strlen(QCLOUD_IOT_DEVICE_SDK_APPID) + MAX_CONN_ID_LEN +
cur_timesec_len + 4;
    options->username = (char*)HAL_Malloc(username_len);
    get_next_conn_id(options->conn_id);
    HAL_Snprintf(options->username, username_len, "%s;%s;%s;%ld",
options->client_id, QCLOUD_IOT_DEVICE_SDK_APPID, options->conn_id,
```

```
cur_timesec);
#if defined(AUTH_WITH_NOTLS) && defined(AUTH_MODE_KEY)
    if (options->device_secret != NULL && options->username != NULL) {
        char                sign[41]    = {0};
        utils_hmac_sha1(options->username, strlen(options->username),
sign, options->device_secret, options->device_secret_len);
        options->password = (char*) HAL_Malloc (51);
        if (options->password == NULL)
IOT_FUNC_EXIT_RC(QCLOUD_ERR_INVAL);
        HAL_Snprintf(options->password, 51, "%s;hmacsha1", sign);
    }
#endif
            ......
}
```

Usage Reference

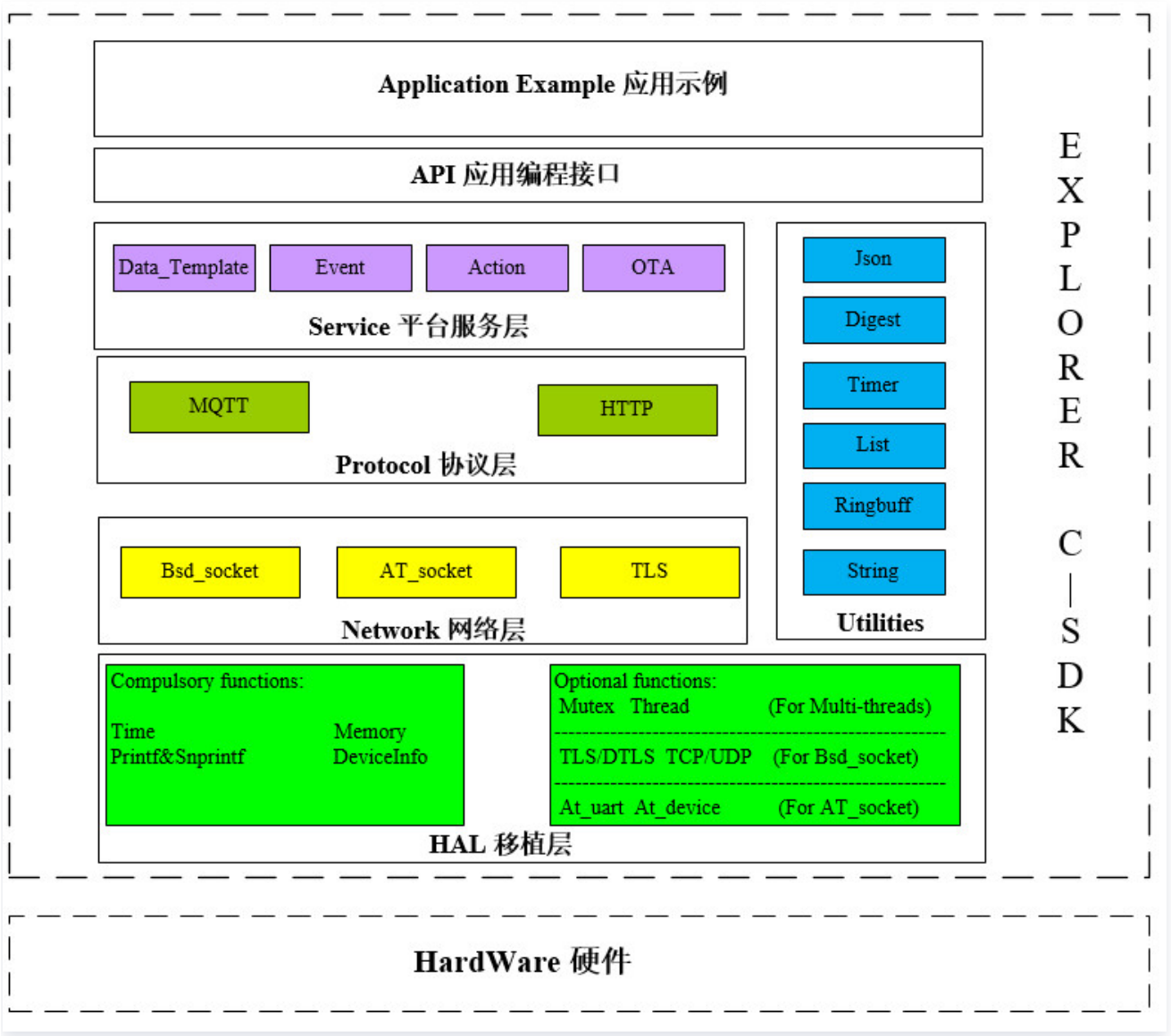
Last updated：2025-04-27 17:55:14

Tencent Cloud IoT Device-side C SDK relies on a secure and high-performance data channel to provide developers in the IoT domain with the ability to quickly integrate devices with the cloud and perform two-way communication with the cloud.

SDK Acquisition

The SDK is managed on Github and can be accessed on Github to download the latest version of the device-side C SDK.

Software Architecture



The SDK is designed in four layers, from top to bottom, namely platform service layer, core protocol layer, network layer, and Hardware Abstraction Layer.

- **service layer**
  - Above the network protocol layer, features such as device access authentication, device shadow, gateway, dynamic registration, log reporting and OTA are implemented.
  - The core protocol for interaction between the device side and the IoT Explorer platform is MQTT. Based on this core protocol, data template and OTA features are implemented. The platform defines the Data Template Protocol through the common abstraction of IoT devices. The cloud and devices implement data flow interaction of the data template protocol data via the payload carried by MQTT. The upgrade command, version and firmware information of the OTA feature are interacted through the MQTT protocol channel, and the firmware download is interacted through the HTTPS protocol channel.
- **protocol layer**
  The network protocols for interaction between the device side and the IoT platform include MQTT/CoAP/HTTP.
- **Network layer**
  The implementation of the network layer supports bsd_socket method and AT_socket method. For systems with abundant resources that have integrated TCP/IP or LwIP network protocol stack, the network interface of bsd_socket can be chosen. For some resource-constrained devices that achieve network access through the interaction between communication modules (cellular modules/Wi-Fi modules, etc.) and MCU, the at_socket framework provided by the SDK can be chosen. For communication modules not supported by the SDK, refer to the implementation of the driver interface in the at_device_op_t structure of the at_device struct supported by the SDK.
- **Hardware Abstraction Layer**
  The Hardware Abstraction Layer needs to be ported for specific software and hardware platforms, and is divided into two parts of HAL layer interfaces: essential implementations and optional implementations. The essential implementation interfaces include time (obtain the number of milliseconds), print, formatted print, memory operations, and device information read and write. For the optional implementation interfaces, if using RTOS, it is required to implement locks, semaphores, thread creation and termination, and delay sleep. If using AT_Socket to access the network, it is required to implement the AT Serial Port Driver and module driver. The SDK already supports the HAL layer example porting implementation for four typical environments: Linux, Windows, FreeRTOS, and nonOS. The relevant examples can be directly compiled and run in the Linux and Windows environments.

Porting Guide

Development Platform	Document
Linux	Linux Platform Access Guide
Windows	Windows Platform Access Guide
MCU+ General AT Module + FreeRTOS	MCU+ General TCP AT Module (FreeRTOS) Access Guide
MCU+ General AT Module + nonOS	MCU+ General TCP AT Module (nonOS) Access Guide
FreeRTOS+lwIP	FreeRTOS+lwIP Platform Access Guide
other platforms	C SDK Porting Integration Guide

SDK Related Documentation

SDK-related documentation. For details, see SDK Usage Reference.

AT SDK Usage Reference

Last updated: 2025-04-27 17:56:37

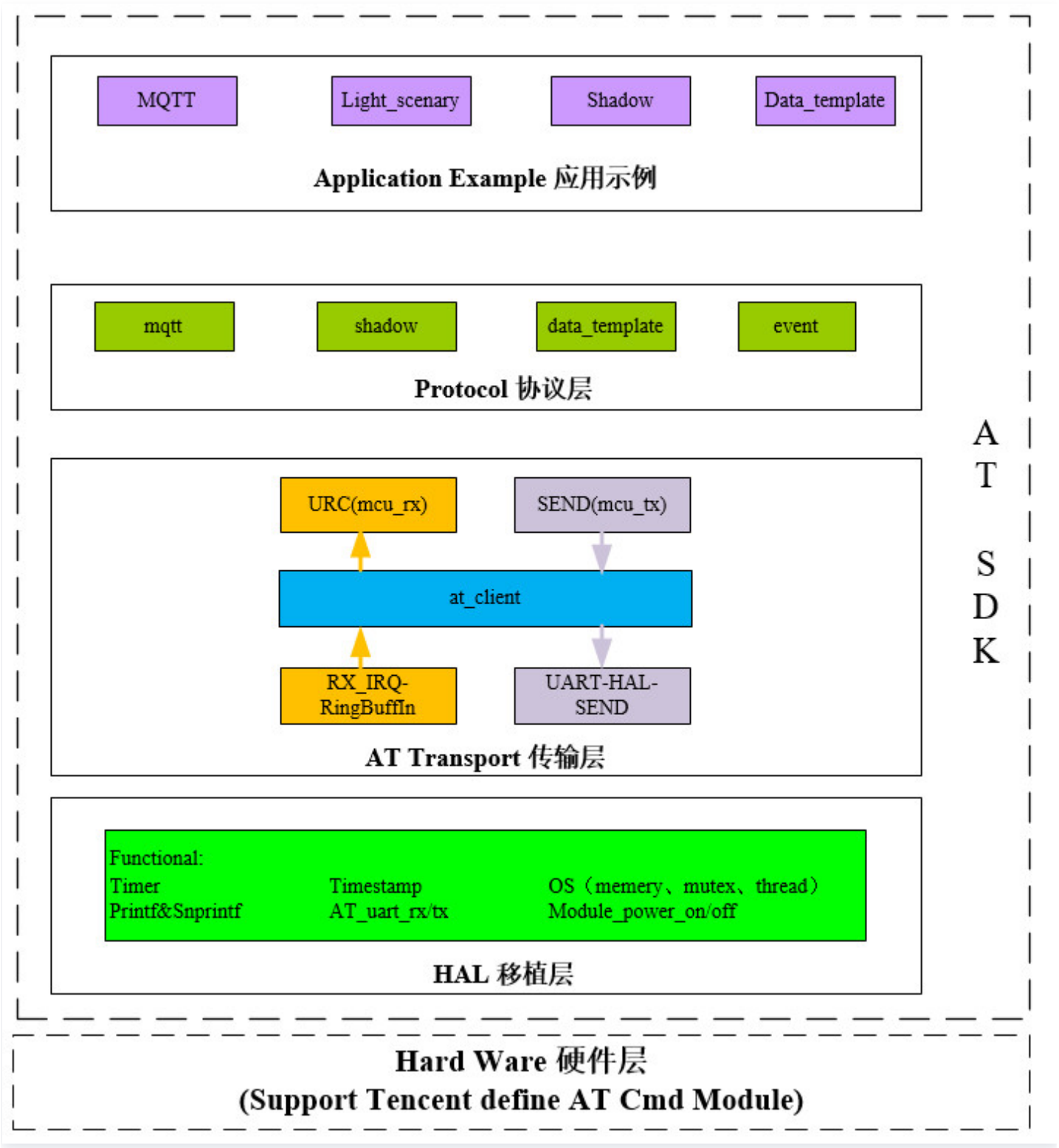
The AT SDK is oriented to modules with built-in Tencent Cloud IoT AT commands and provides an SDK for implementing and customizing module interactions.

SDK Acquisition

After creating products and devices on the IoT explorer platform <1>, select the development method based on MQTT AT customized module. The <3>AT SDK</3> code on the MCU side will be automatically generated, and the corresponding configuration and initialization code for the data template and events created on the platform will also be generated.

Software Architecture

The architecture diagram of the AT SDK software is as follows:



The SDK is designed in four layers, from top to bottom, namely the application layer, core protocol layer, AT transport layer, and Hardware Abstraction Layer.

- **Service layer**
  Above the network protocol layer, it implements features including device access authentication, device shadow, gateway, dynamic registration, log reporting, and OTA.

- **Protocol layer**
  The network protocols for interaction between the device end and the IoT platform include MQTT/COAP/HTTP.
- **AT Transport Layer**
  Implement a network protocol stack customized for AT instructions based on Tencent Cloud IoT.
- **Hardware Abstraction Layer**
  Abstract encapsulation of underlying operations for different hardware platforms. Porting is required targeting specific software and hardware platforms. It is divided into two parts of HAL layer interfaces: required implementation and optional implementation.

Directory Structure

Name	Overview
docs	Documentation directory, including the definition of Tencent AT Instruction Set.
port	HAL layer porting directory. Implement the send and receive interface (interrupt reception) of the serial port, delay function, power on/off of the module, and OS-related interfaces.
sample	Application examples, example usages of MQTT, shadow, and data template.
src	Implementation of AT framework and protocol logic.
— event	Event function protocol encapsulation.
— module_at	Abstract of at client, implementation of RX parsing, command downlink, urc match, and asynchronous resp match.
— shadow	Implementation of shadow logic based on AT framework.
— mqtt	Implementation of MQTT protocol based on AT framework.
— utils	json, timer, linked list applications.
— include	External SDK header files and device information configuration header files.

usr_logic	Automatically generated skeleton code for business logic defined by your own product.
— data_config.c	User-defined data point.
— events_config.c	User-defined event.
—data_template_usr_logic.c	Skeleton framework for users' services processing logic. Just implement the reserved uplink and downlink business logic processing functions.
tools	Code generation script.
README.md	SDK Usage Instructions.

Porting Guide

Development platform	Reference Documentation
Customize AT Module (Cellular type) for MCU+	Integration guide for custom MQTT AT module (cellular type) for MCU+
Customize AT Module (Wi-Fi type) for MCU+	Access guide for custom MQTT AT module (Wi-Fi type) for MCU+

SDK API Description

For more usage methods and understanding of APIs about the SDK, please refer to qcloud_iot_api_export.h .

Using Android SDK Reference

Last updated: 2025-04-27 17:56:56

Android SDK for platforms using Java language to integrate with Tencent Cloud IoT Explorer.

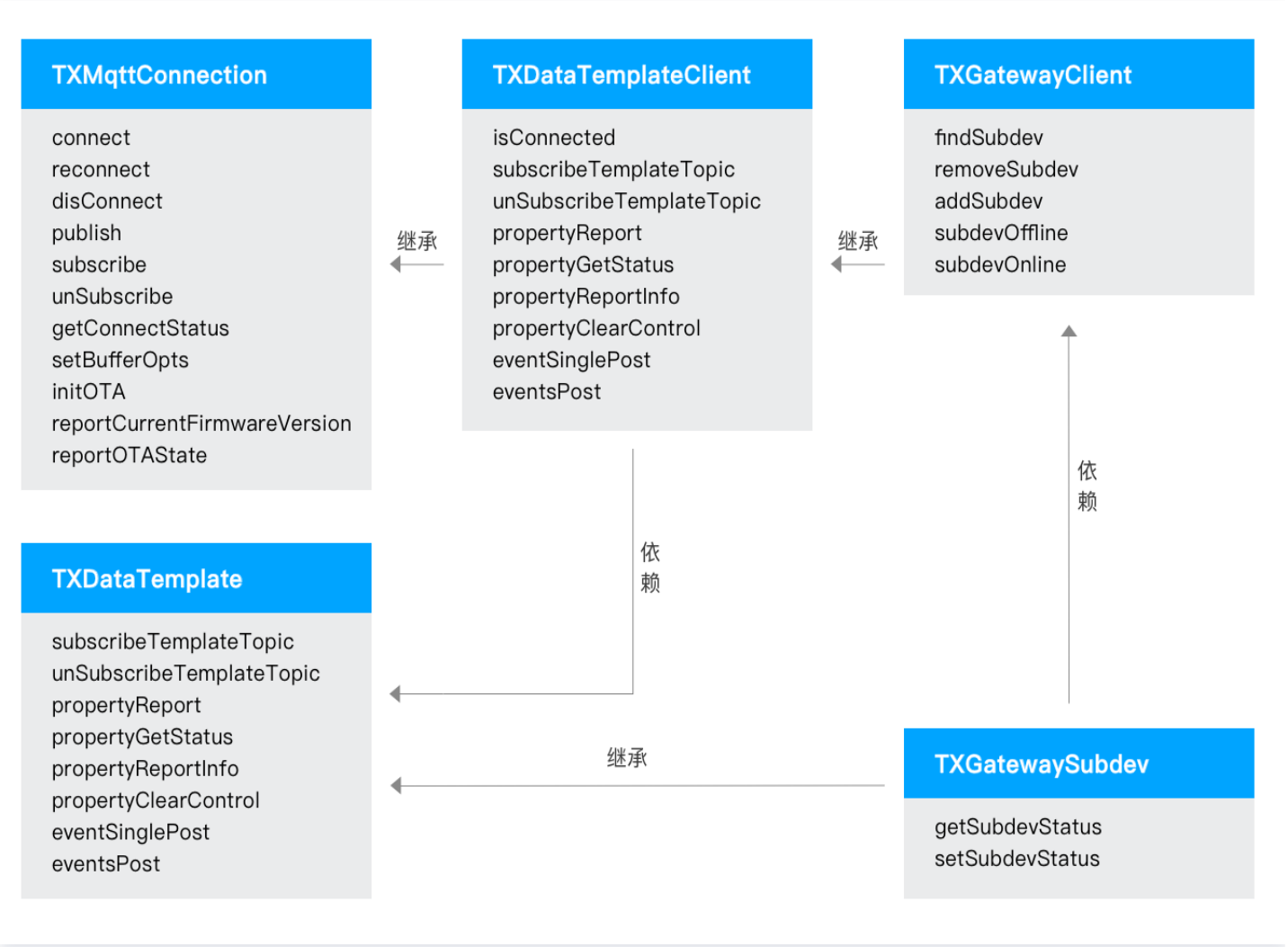
SDK Acquisition

The SDK is managed on Github and accessible on Github to download the latest version of the device-side iot-device-android.

Software Architecture

Class Name	Feature
TXMqttConnection	Connect to IoT Explorer.
TXDataTemplate	Implement basic features of Thing Model.
TXDataTemplateClient	Implement direct devices to connect to the IoT development platform according to the Thing Model.
TXGatewayClient	Implement gateway devices to connect to the IoT development platform according to the Thing Model.
TXGatewaySubdev	Implement the connection of gateway subdevices to the IoT development platform according to the Thing Model.

The architecture diagram of Tencent Cloud IoT Explorer Android SDK is as follows:



Porting Guide

Android SDK porting guide. For details, see Android Platform Access Guide.

SDK API Description

TXMqttConnection

Method Name	Description
connect	MQTT connection.
reconnect	MQTT reconnection.
disConnect	Disconnect MQTT connection.
publish	Publish an MQTT message.

subscribe	Subscribe to an MQTT topic.
unSubscribe	Unsubscribe from MQTT topic.
getConnectStatus	Get MQTT connection status.
setBufferOpts	Set Disconnection Status buffer.
initOTA	Initialize the OTA function.
reportCurrentFirmwareVersion	Submit the current version information of the device to the backend server.
reportOTAState	Report device upgrade status to backend server.

TXDataTemplate

Method Name	Description
subscribeTemplateTopic	Subscribe to Thing Model-related topics.
unSubscribeTemplateTopic	Unsubscribe from Thing Model-related topics.
propertyReport	Reported attribute.
propertyGetStatus	Update status.
propertyReportInfo	Submit device information.
propertyClearControl	Clear control information.
eventSinglePost	Submit a single event.
eventsPost	Submit multiple events.

TXDataTemplateClient

Method Name	Description
isConnected	Whether it has been connected to the IoT development platform.
subscribeTemplateTopic	Subscribe to Thing Model-related topics.

unSubscribeTemplateTopic	Unsubscribe from Thing Model-related topics.
propertyReport	Reported attribute.
propertyGetStatus	Update status.
propertyReportInfo	Submit device information.
propertyClearControl	Clear control information.
eventSinglePost	Submit a single event.
eventsPost	Submit multiple events.

TXGatewayClient

Method Name	Description
findSubdev	Find a sub-device (based on product ID and device name).
removeSubdev	Delete a subdevice.
addSubdev	Add a sub-device.
subdevOffline	Launch a subdevice.
subdevOnline	Decommission a sub-device.

TXGatewaySubdev

Method Name	Description
getSubdevStatus	Get the connection status of the sub-device.
setSubdevStatus	Set the connection status of the sub-device.

Java SDK Usage Reference

Last updated: 2025-04-27 17:59:25

Java SDK realizes the integration of platforms oriented to Java language with Tencent Cloud IoT Explorer.

SDK Acquisition

The SDK is hosted on Github and can be accessed from Github to download the latest version of the device-side explorer-device-java.
If you want to perform project development through jar reference method, you can add dependencies in the build.gradle under the module directory, as follows:

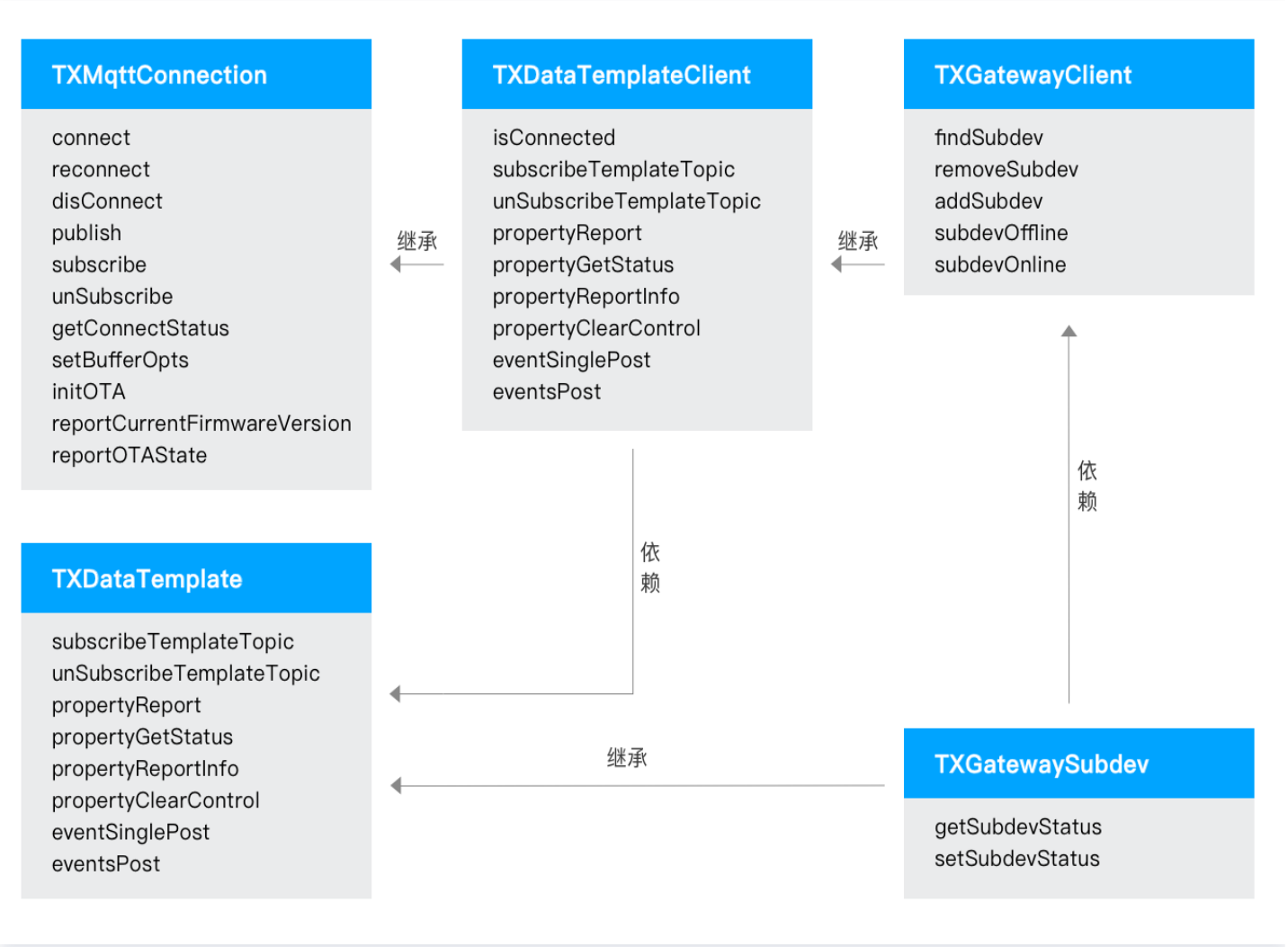
```
dependencies {
 ...
 implementation 'com.tencent.iot.explorer:explorer-device-java:x.x.x'
}
```

Note:
Users can set the above x.x.x to the latest version based on version description.

Software Architecture

Class Name	Feature
TXMqttConnection	Connect to the IoT development platform.
TXDataTemplate	Implement basic features of Thing Model.
TXDataTemplateClient	Connect directly connected devices to IoT Explorer according to Thing Model.
TXGatewayClient	Implement gateway devices to connect to the IoT development platform according to the Thing Model.
TXGatewaySubdev	Implement gateway subdevices to connect to the IoT development platform according to the Thing Model.

The architecture diagram of Tencent Cloud IoT Explorer Java SDK is as follows:



Porting Guide

Java SDK Porting Guide. For details, see Java Platform Access Guide.

SDK API Description

TXMqttConnection

Method Name	Description
connect	MQTT connection.
reconnect	MQTT reconnection.
disConnect	Disconnect the MQTT connection.
publish	Publish an MQTT message.

subscribe	Subscribe to an MQTT topic.
unSubscribe	Unsubscribe from an MQTT topic.
getConnectStatus	Get MQTT connection status.
setBufferOpts	Set the disconnection status buffer.
initOTA	Initialize the OTA function.
reportCurrentFirmwareVersion	Submit the current version information of the device to the backend server.
reportOTAState	Report device upgrade status to backend server.

TXDataTemplate

Method Name	Description
subscribeTemplateTopic	Subscribe to topics related to Thing Model.
unSubscribeTemplateTopic	Unsubscribe from topics related to Thing Model.
propertyReport	Reported attribute.
propertyGetStatus	Update status.
propertyReportInfo	Submit device information.
propertyClearControl	Clear control information.
eventSinglePost	Submit a single event.
eventsPost	Submit multiple events.

TXDataTemplateClient

Method Name	Description
isConnected	Whether it has been connected to the IoT development platform.
subscribeTemplateTopic	Subscribe to topics related to Thing Model.
unSubscribeTemplateTopic	Unsubscribe from topics related to Thing Model.
propertyReport	Reported attribute.

propertyGetStatus	Update status.
propertyReportInfo	Submit device information.
propertyClearControl	Clear control information.
eventSinglePost	Submit a single event.
eventsPost	Submit multiple events.

TXGatewayClient

Method Name	Description
findSubdev	Find a sub-device (by product ID and device name).
removeSubdev	Delete a subdevice.
addSubdev	Add a sub-device.
subdevOffline	Launch a subdevice.
subdevOnline	Decommission a sub-device.

TXGatewaySubdev

Method Name	Description
getSubdevStatus	Get the connection status of the sub-device.
setSubdevStatus	Set the connection status of the sub-device.

Python SDK Usage Reference

Last updated: 2025-04-27 17:59:41

Tencent Cloud IoT Device SDK for Python relies on a secure and high-performance data channel to provide developers in the IoT domain with the ability to quickly access the cloud from the device end and perform two-way communication with the cloud. Developers only need to complete the corresponding configuration of the project to complete the access of the device.

Prerequisites

The product and device have been created on the Explorer platform.

Reference Method

- If you need to perform project development by reference method, you can install SDK, as follows:

```
pip3 install tencent-iot-device
```

If you need to view the used SDK version, use the following command:

```
pip3 show --files tencent-iot-device
```

If you need to upgrade the SDK version, use the following command:

```
pip3 install --upgrade tencent-iot-device
```

- If you need to perform project development through code integration, you can visit Github to download the Python SDK source code.

MQTT API

The relevant APIs of MQTT are defined in the explorer.py class, supporting publish and subscribe functionality. The introduction is as follows:

Method Name	Description
connect	MQTT connection.
disconnect	Disconnect the MQTT connection.

subscribe	MQTT subscribe.
unsubscribe	MQTT unsubscribe.
publish	MQTT publish a message.
registerMqttCallback	Register the MQTT callback function.
registerUserCallback	Registered user callback function.
isMqttConnected	Checks whether MQTT is connected normally.
getConnectState	Get MQTT connection status.
setReconnectInterval	Set MQTT reconnection attempt interval.
setMessageTimout	Set the message sending timeout.
setKeepaliveInterval	Set MQTT keep-alive interval.

MQTT Gateway API

- For devices that cannot be directly accessed to the Ethernet network, you can first integrate them into the network of the local gateway device, and use the communication feature of the gateway device to integrate the proxy device into the explorer platform.
- For subdevices in the local area network that join or exit the network, binding or unbinding operations need to be performed through the platform.

> ⚠ **Note:**
> When a sub-device initiates going online, subsequently, as long as the gateway connects successfully, the backend will display the sub-device online unless the device has initiated an offline operation.

The API definitions related to the MQTT gateway are in the gateway.py Class Interface, as follows:

Method Name	Description
gatewayInit	Gateway initialization.
isSubdevStatusOnline	Determine whether sub-devices are online.
updateSubdevStatus	Update sub-device online status.

gatewaySubdevGetConfigList	Obtain sub-device list from configuration file.
gatewaySubdevOnline	Proxy sub-device online.
gatewaySubdevOffline	Agent Subdevice Offline.
gatewaySubdevBind	Bind sub-device.
gatewaySubdevUnbind	Unbind sub-device.
gatewaySubdevSubscribe	Sub-device subscription.

Thing Model Interface

If you need to use the Thing Model feature, you need to use the APIs in the template.py class. The introduction is as follows:

API Name	API Description
templateInit	Thing model initialization.
getEventsList	Obtain device event list.
getActionList	Obtain device action list.
getPropertyList	Obtain device property list.
templateSetup	Parse Thing Model.
templateEventPost	events reporting.
templateJsonConstructReportArray	Construct the json structure for reporting.
templateReportSysInfo	Device information reporting.
templateControlReply	Control message response.
templateActionReply	action message response.
templateGetStatus	Obtain the latest status of the device.
templateReport	Device attribute reporting.
clearControl	Clear control.
templateDeinit	Terminate the Thing Model.

Dynamic Registration API

If you need to use the dynamic registration function, you need to use the API in the explorer.py class. The introduction is as follows:

Method Name	Description
dynregDevice	Retrieve information on dynamic device registration.

OTA API

If you need to use the OTA feature, you need to use the API in the explorer.py class. The introduction is as follows:

Method Name	Description
otaInit	Initialize OTA.
otaIsFetching	Determine whether it is downloading.
otaIsFetchFinished	Determine whether the download is complete.
otaReportUpgradeSuccess	Submit an upgrade success message.
otaReportUpgradeFail	Submit an upgrade failure message.
otaIoctlNumber	Retrieve int type information such as firmware size for download.
otaIoctlString	Retrieve string type information such as firmware md5 for download.
otaResetMd5	Reset the md5 information.
otaMd5Update	Update the md5 information.
httpInit	Initialize HTTP.
otaReportVersion	Submit the current firmware version information.
otaDownloadStart	Start firmware download.
otaFetchYield	Read the firmware.

C# Integration Reference

Last updated: 2025-04-27 17:59:59

C# is a modern, common, object-oriented programming language developed and launched by Microsoft in the.NET framework. It combines the powerful features of C++ and the ease of use of Java, enabling developers to build various types of applications, including Windows client applications, Web applications, database applications, mobile applications, and games. Tencent Cloud IoT Explorer supports integration using C#. This article introduces how to use the MQTTnet Client library in a C# project to achieve connection, topic subscription, uplink and downlink message interaction, and other features with Tencent Cloud IoT Explorer.

Note:

1. This example is based on donet 6.0.
2. The versions of third-party libraries used are as follows:
   - MQTTnet:v3.0.11
   - MQTTnet.Extensions.ManagedClient:v3.0.11
   - Newtonsoft.Json:v12.0.3
   - NLog:v5.2.3

Overview

A smart light is integrated into the IoT Explorer. Through the IoT Explorer, the brightness, color, and switch of the light can be remotely controlled, and the data reported to the IoT Explorer by the smart light can be accessed in real time.

Preparations

1. Apply for Tencent Cloud IoT Explorer service.
2. One Windows computer with VS2017 and above versions installed.

Operation Steps

Project Creation

1. Log in to IoT Explorer Console, select the platform default **public instance** or the **enterprise instance** purchased by user.
2. Click an instance. By default, enter the project list page and click **Create a New Project**.
   - Project name: required, input "Smart Light Demo" or other name.
   - Project description: Fill in the project description according to actual needs.

3. After completing the basic information filling of the project, click **Save** to complete the creation of the new project.
4. After the project is successfully created, you can create a product.

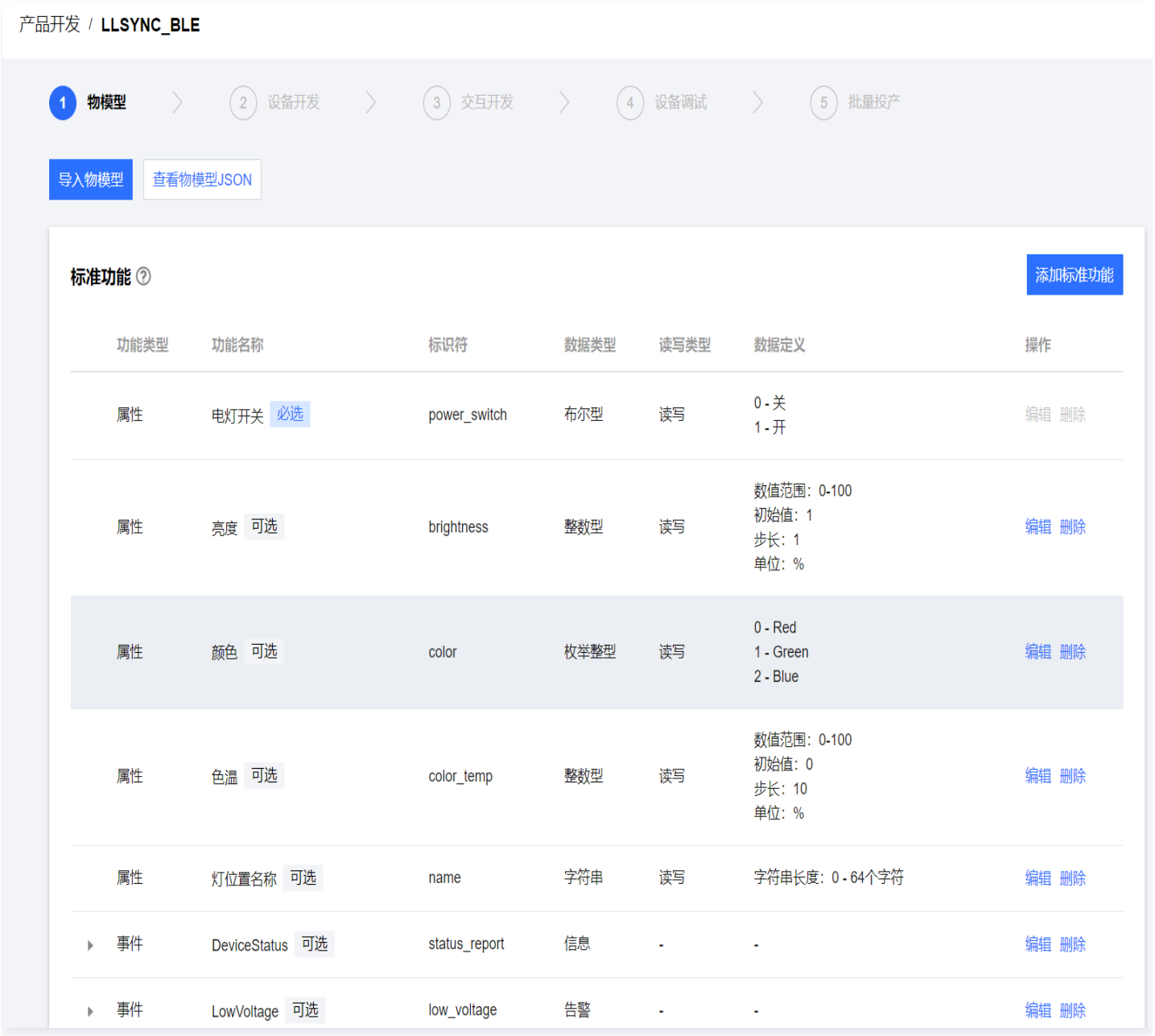
Create Product

1. Click the project name, enter the product list page, and click **Create Product**.
2. On the create product page, fill in the basic information of the product.
   - Product name: required, manually input "Smart Light" or other product names.
   - Product category: Select the standard category "Intelligent Life" > "Electrical Lighting" > "Lights".
   - Device type: Select "device".
   - Authentication method: Select "key authentication".
   - Communication method: select as needed.
   - Others are default options.

3. After the product information is filled in, click **Save** to complete the creation of the product.
4. After the product is successfully created, you can view "Smart Light" on the product list page.

Define the Thing Model of a Product

After selecting the "Light" type, the system automatically generates standard features.



Creating Device

On the device debugging page, click **Create New Device**. The device name is dev001.

For an introduction to the protocol of the Thing Model, please see Thing Model Protocol.

Using C# MQTT Client

Through these steps, you have already successfully obtained the triplet information of the device on Tencent Cloud IoT Explorer. Next, you can use this sample code to integrate with C#.

Input Triplet Information

Click the name of the created device to obtain the device triplet information.

```
// Cloud-generated triplet information
static string PRODUCT_ID = "YOUR_PRODUCT_ID";  // PRODUCT ID
static string DEVICE_NAME = "YOUR_DEVICE_NAME"; // device name
static string DEVICE_SECRET = "IOT_PSK";  // device key
```

Generating MQTT client Connection Parameters

Generate MQTT's `clientId`, `userName`, and `password` using the triplet information just filled in.

```
// Generate MQTT connection parameters
byte[] decodeBytes = Convert.FromBase64String(DEVICE_SECRET);
string clientId = PRODUCT_ID + DEVICE_NAME;
string usrNmae = clientId + ";21010406;" + GetNextConnId() +";"+
0x7fffffff.ToString();
string password = ComputeHmacSha1(usrNmae, decodeBytes) + ";hmacsha1";
```

Create an MQTT client and Connect to the Cloud Platform

```
IManagedMqttClient mqttClient = new
MqttFactory().CreateManagedMqttClient();

var mqttOptions = new ManagedMqttClientOptionsBuilder()
    .WithAutoReconnectDelay(TimeSpan.FromSeconds(10))
    .WithClientOptions(new MqttClientOptionsBuilder()
        .WithClientId(clientId)
        .WithCredentials(usrNmae, password)
        .WithTcpServer(url, 1883) // Non-tls mode
        .WithCleanSession()
        .Build())
    .Build();
```

If you use TLS encryption for access, configure as follows:

```
IManagedMqttClient mqttClient = new
MqttFactory().CreateManagedMqttClient();

var mqttOptions = new ManagedMqttClientOptionsBuilder()
    .WithAutoReconnectDelay(TimeSpan.FromSeconds(10))
    .WithClientOptions(new MqttClientOptionsBuilder()
        .WithClientId(clientId)
        .WithCredentials(usrNmae, password)
        .WithTcpServer(url, 8883) // tls integration
        .WithTls(new MqttClientOptionsBuilderTlsParameters {
            UseTls = true,
            IgnoreCertificateChainErrors = true,
            IgnoreCertificateRevocationErrors = true,
            AllowUntrustedCertificates = true,
        })
        .WithCleanSession()
        .Build())
    .Build();
```

Listen for MQTT Connection, Disconnection or Receipt Events

```
// Listen for connection events
mqttClient.UseConnectedHandler(e =>
{
    log.Info("mqtt connect success with " + deviceId);
    return Task.CompletedTask;
```

```
});

// Listen for disconnection events
mqttClient.UseDisconnectedHandler(e =>
{
    log.Error("mqtt disconnect with " + deviceId);
    return Task.CompletedTask;
});

// Listen to received messages.
mqttClient.UseApplicationMessageReceivedHandler(e =>
{

    // Process model data
    string topic = e.ApplicationMessage.Topic;
    string payload =
Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
    log.Debug($"down message topic: {topic} paylaod : {payload}");
    if (topic.Contains("/property/"))
    {
        // Attribute processing

    }else if (topic.Contains("/event/"))
    {
        // Event handling

    }else if (topic.Contains("/action/"))
    {
        // Handle behavior

    }
    return Task.CompletedTask;
});
```

Connecting to Cloud Platform

```
// Connect to the platform
await mqttClient.StartAsync(mqttOptions);
```

Subscribe to the Thing Model topic

```
 // Subscribe to Thing Model topic
```

```
var topicFilters = new[]
{
    new
MqttTopicFilterBuilder().WithTopic("$thing/down/property/"+deviceId).Bui
ld(),
    new
MqttTopicFilterBuilder().WithTopic("$thing/down/event/"+deviceId).Build(
),
    new
MqttTopicFilterBuilder().WithTopic("$thing/down/action/"+deviceId).Build
()
 };
await mqttClient.SubscribeAsync(topicFilters);
```

Brightness Example Submission

```
  int brightness = 25;
  var payload_json = new JObject
  {
      ["method"] = "report",
      ["clientToken"] =
DateTimeOffset.Now.ToUnixTimeSeconds().ToString(),
      ["params"] = new JObject
      {
          ["brightness"] = brightness
      }
  };
  string payload = JsonConvert.SerializeObject(payload_json).ToString();
  var message = new MqttApplicationMessageBuilder()
      .WithTopic("$thing/up/property/" + deviceId)
      .WithPayload(payload)
      .WithQualityOfServiceLevel(0)
      .WithRetainFlag(false)
      .Build();
  await mqttClient.PublishAsync(message);
```

The complete sample code is as follows:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Security.Cryptography;
```

```
using NLog;
using MQTTnet;
using MQTTnet.Client.Options;
using MQTTnet.Extensions.ManagedClient;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;


namespace IoT.Explorer.Test
{
    class DatatemplateTest
    {
        private static Logger log = LogManager.GetCurrentClassLogger();
        // Cloud-generated triplet information
        static string PRODUCT_ID = "F2F43QKKA4";
        static string DEVICE_NAME = "5629fbfa12f4";
        static string DEVICE_SECRET = "a91V4htL41oILv80lgCeLA==";

        public static string GetNextConnId()
        {
            char[] connId = new char[6];
            Random random = new Random();
            for (int i = 0; i < 6 - 1; i++)
            {
                int flag = random.Next(3);
                switch (flag)
                {
                    case 0:
                        connId[i] = (char)(random.Next(26) + 'a');
                        break;
                    case 1:
                        connId[i] = (char)(random.Next(26) + 'A');
                        break;
                    case 2:
                        connId[i] = (char)(random.Next(10) + '0');
                        break;
                }
            }

            connId[6 - 1] = '\0';
            return new string(connId);
        }
```

```
        public static string ComputeHmacSha1(string inputString, byte[]
keyBytes)
        {
            byte[] inputBytes = Encoding.UTF8.GetBytes(inputString);

            using (HMACSHA1 hmac = new HMACSHA1(keyBytes))
            {
                byte[] hashBytes = hmac.ComputeHash(inputBytes);
                string hashString =
BitConverter.ToString(hashBytes).Replace("-", "").ToLower();
                return hashString;
            }
        }

        static async Task Main(string[] args)
        {
            // Generate MQTT connection parameters
            byte[] decodeBytes =
Convert.FromBase64String(DEVICE_SECRET);
            string clientId = PRODUCT_ID + DEVICE_NAME;
            string usrNmae = clientId + ";21010406;" + GetNextConnId()
+";"+ 0x7fffffff.ToString();
            string password = ComputeHmacSha1(usrNmae, decodeBytes) +
";hmacsha1";
            string url = PRODUCT_ID + ".iotcloud.tencentdevices.com";
            string deviceId = PRODUCT_ID + "/" + DEVICE_NAME;

            try
            {
                IManagedMqttClient mqttClient = new
MqttFactory().CreateManagedMqttClient();

                var mqttOptions = new ManagedMqttClientOptionsBuilder()
                    .WithAutoReconnectDelay(TimeSpan.FromSeconds(10))
                    .WithClientOptions(new MqttClientOptionsBuilder()
                        .WithClientId(clientId)
                        .WithCredentials(usrNmae, password)
                        .WithTcpServer(url, 8883)
                        .WithTls(new
MqttClientOptionsBuilderTlsParameters {
                            UseTls = true,
                            IgnoreCertificateChainErrors = true,
                            IgnoreCertificateRevocationErrors = true,
```

```
                        AllowUntrustedCertificates = true,
                    })
                    .WithCleanSession()
                    .Build())
                .Build();

            // Listen for connection events
            mqttClient.UseConnectedHandler(e =>
            {
                log.Info("mqtt connect success with " + deviceId);
                return Task.CompletedTask;
            });

            // Listen for disconnection events
            mqttClient.UseDisconnectedHandler(e =>
            {
                log.Error("mqtt disconnect with " + deviceId);
                return Task.CompletedTask;
            });

            // Listen to received messages.
            mqttClient.UseApplicationMessageReceivedHandler(e =>
            {

                // Process model data
                string topic = e.ApplicationMessage.Topic;
                string payload =
Encoding.UTF8.GetString(e.ApplicationMessage.Payload);
                log.Debug($"down message topic: {topic} paylaod :
{payload}");
                if (topic.Contains("/property/"))
                {
                    // Attribute processing

                }else if (topic.Contains("/event/"))
                {
                    // Event handling

                }else if (topic.Contains("/action/"))
                {
                    // Handle behavior

                }
```

```
                    return Task.CompletedTask;
                });

                // Connect to the platform.
                await mqttClient.StartAsync(mqttOptions);
                Thread.Sleep(2000);

                // Subscribe to Thing Model topic
                var topicFilters = new[]
                {
                    new
MqttTopicFilterBuilder().WithTopic("$thing/down/property/"+deviceId).Bui
ld(),
                    new
MqttTopicFilterBuilder().WithTopic("$thing/down/event/"+deviceId).Build(
),
                    new
MqttTopicFilterBuilder().WithTopic("$thing/down/action/"+deviceId).Build
()
                 };
                await mqttClient.SubscribeAsync(topicFilters);
                int brightness = 0;
                while (true) {
                    // Submit brightness periodically
                    var payload_json = new JObject
                    {
                        ["method"] = "report",
                        ["clientToken"] =
DateTimeOffset.Now.ToUnixTimeSeconds().ToString(),
                        ["params"] = new JObject
                        {
                            ["brightness"] = brightness
                        }
                    };
                    string payload =
JsonConvert.SerializeObject(payload_json).ToString();
                    var message = new MqttApplicationMessageBuilder()
                        .WithTopic("$thing/up/property/" + deviceId)
                        .WithPayload(payload)
                        .WithQualityOfServiceLevel(0)
                        .WithRetainFlag(false)
                        .Build();
                    await mqttClient.PublishAsync(message);
```

```
                    log.Debug("publish message :" + payload);
                    brightness++;
                    if (!mqttClient.IsConnected)
                    {
                        break;
                    }
                    Thread.Sleep(10000);
                }
                // disconnect
                await mqttClient.StopAsync();

            }
            catch (Exception ex)
            {
                var name = ex.GetType().FullName;
                log.Error("Exception: " + ex.Message);
            }

        }
    }
}
```

Running logs are as follows:

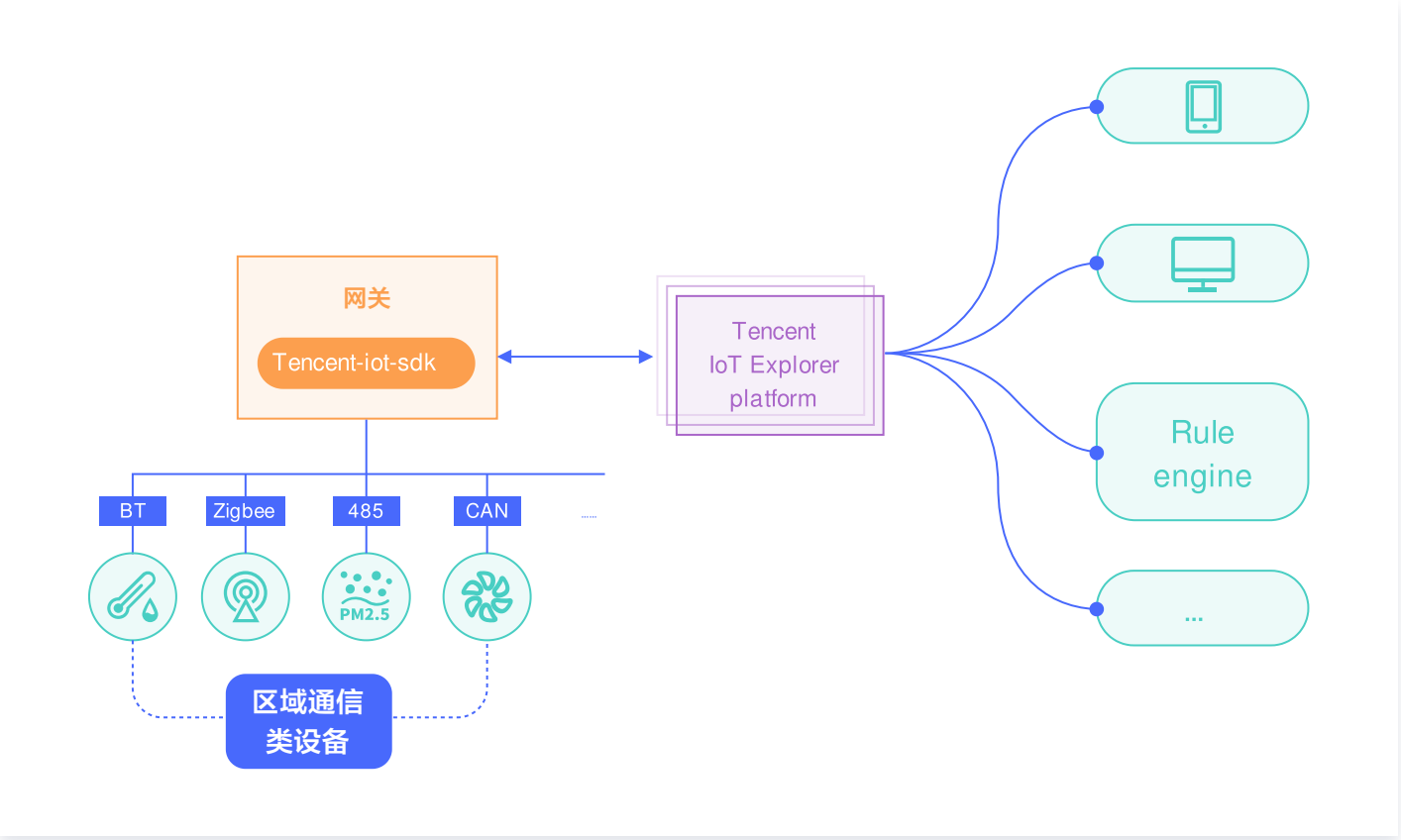
```
2023-08-09 12:04:20.0860 INFO IoT.Explorer.Test.DatatemplateTest - mqtt
connect success with F2F43QKKA4/5629fbfa12f4
2023-08-09 12:04:21.7918 DEBUG IoT.Explorer.Test.DatatemplateTest -
publish message :{"method":"report","clientToken":"1691553861","params":
{"brightness":0}}
2023-08-09 12:04:21.8791 DEBUG IoT.Explorer.Test.DatatemplateTest - down
message topic: $thing/down/property/F2F43QKKA4/5629fbfa12f4 paylaod :
{"method":"report_reply","clientToken":"1691553861","code":0,"status":"s
uccess"}
2023-08-09 12:04:25.6359 DEBUG IoT.Explorer.Test.DatatemplateTest - down
message topic: $thing/down/property/F2F43QKKA4/5629fbfa12f4 paylaod :
{"method":"control","clientToken":"v2149648760ozOCb::af400362-c384-40dd-
b39c-94d82950e1ad","params":{"power_switch":1}}
2023-08-09 12:04:29.4171 DEBUG IoT.Explorer.Test.DatatemplateTest - down
message topic: $thing/down/property/F2F43QKKA4/5629fbfa12f4 paylaod :
{"method":"control","clientToken":"v2146761678bxCmt::295125a1-6b64-4cba-
b2ca-036bbef43390","params":{"power_switch":0}}
2023-08-09 12:04:31.7989 DEBUG IoT.Explorer.Test.DatatemplateTest -
publish message :{"method":"report","clientToken":"1691553871","params":
```

```
{"brightness":1}}
2023-08-09 12:04:31.8919 DEBUG IoT.Explorer.Test.DatatemplateTest - down
message topic: $thing/down/property/F2F43QKKA4/5629fbfa12f4 paylaod :
{"method":"report_reply","clientToken":"1691553871","code":0,"status":"s
uccess"}
```

Gateway and Sub-Device Development
Gateway and Subdevice Integration Instruction

Last updated: 2025-04-28 10:41:25

For devices such as BLE, Zigbee and 485 that do not have direct network access capabilities, it is necessary to connect to an access gateway first, and then indirectly realize device connectivity to Tencent IoT Explorer through a gateway proxy. The specific process solution framework diagram is as follows:



Term Definition

- Gateway device: Possess northbound and southbound communication capabilities
  - Northbound: Can communicate with the cloud for data interaction.
  - Southbound: Can mount sub-devices through wired or wireless methods, and possess the ability to manage sub-devices and proxy their communication.

- Sub-device: A device that does not have the ability to directly access the network and indirectly realizes data interaction with the cloud through a gateway proxy, such as BLE, Zigbee, 485, 433 devices.
- Topology relationship: The association between sub-devices and gateways is a topology relationship. Gateways and sub-devices need to establish an associated topology relationship so that gateways can proxy sub-devices to go online or offline and communicate. To establish an association between gateway devices and sub-devices, it is required to be implemented at the product level and then at the device level. That is, first associate the product corresponding to the sub-device with the product corresponding to the gateway, and then associate the specific sub-device with the specific gateway device.

Development Process

For scenarios of gateways and subdevices, development needs to be carried out on both the gateway side and the subdevice side. Please refer to the following documentation links respectively.

- Gateway device development
- Sub-device development

Gateway Device Access Guide

Last updated: 2025-04-28 10:45:53

this document introduces How to develop gateway access to Tencent IoT Explorer based on IoT Explorer C SDK.

C SDK Gateway Features

Gateway and Sub-Device Information Management

The device information management APIs of the HAL layer for gateways and subdevices have differences from those for directly connected devices. Call `HAL_GetGwDevInfo` to obtain the device information of gateways and subdevices. This API needs to be adapted and implemented. That is, developers need to manage device information according to the target platform and use cases, especially the device information mapping of real subdevices and platform subdevices.

Gateway and Cloud Connection

Gateway Example `gateway_sample.c` introduces how to use the APIs provided by the C SDK to implement device information acquisition, gateway-cloud connection, proxy sub-device online and offline, proxy sub-device communication, and shows how sub-devices communicate based on data templates.

Proxy Sub-Device Online/Offline

After the gateway is online, perform sub-device management via specific protocols (for example: BLE, Zigbee).

- For subdevices that have already established communication, call `IOT_Gateway_Subdev_Online` to implement the online operation of subdevices in the cloud. For details, see Online Data Request Format.
- For a sub-device that has lost communication, call `IOT_Gateway_Subdev_Offline` to implement the decommissioning of the sub-device in the cloud. For details, see Request Format for Decommissioning Data.

Dynamic Binding and Unbinding Sub-Devices

The gateway can proxy the online and offline status and communication of subdevices, provided that the gateway device and subdevices have established a topological association relationship on the platform.

- Product-level correlation, that is, the correlation between the product of the subdevice and the gateway product must be established in the console.

- Device-level correlation can be established on the console and can be dynamically bound and unbound. For dynamic binding and unbinding, the gateway needs the subdevice key to implement signature. If the signature is calculated on the gateway side, the gateway needs the key corresponding to the subdevice to be bound. Dynamic Binding and Unbinding Data Request Format.

Proxy Sub-Device Communication

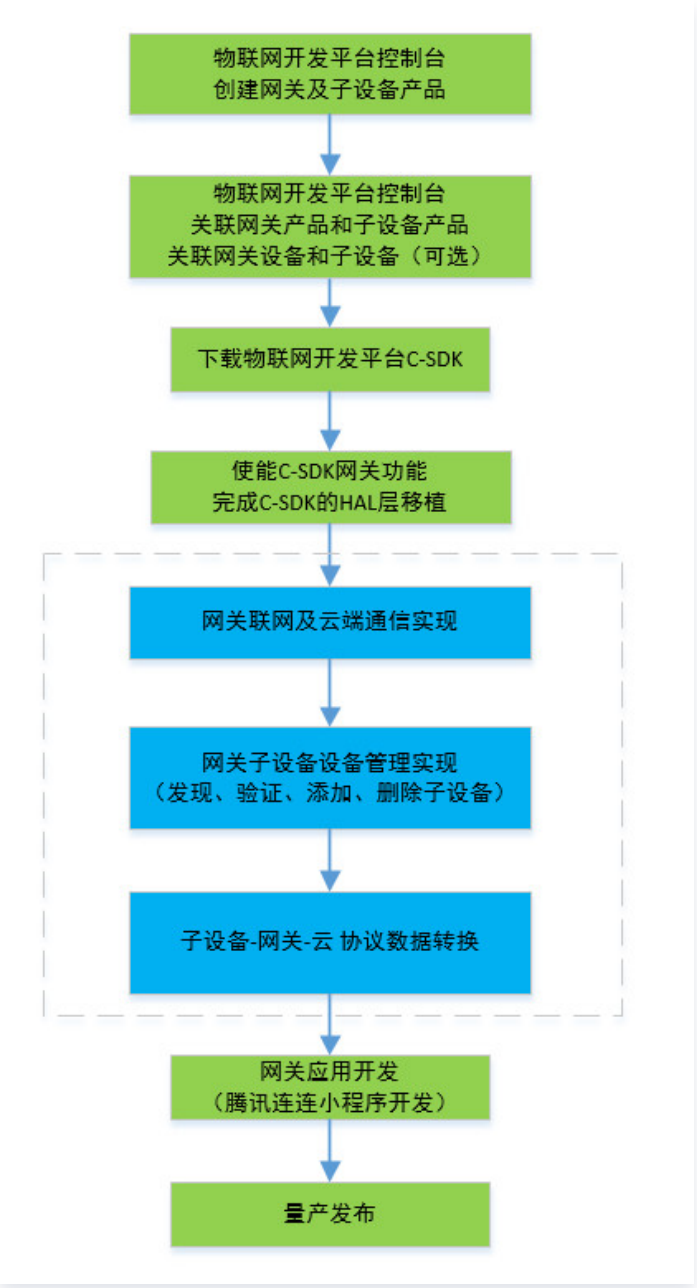
Upon success of the proxy sub-device online, you can call `IOT_Gateway_Subscribe` for the subscription messages of the proxy sub-device, and call `IOT_Gateway_Publish` to push messages to the cloud.

Gateway Example

Gateway Example `gateway_sample.c` introduces how to use the APIs provided by the C SDK to implement device information acquisition, gateway-cloud connection, proxy sub-device online and offline, proxy sub-device communication, and shows how sub-devices communicate based on data templates.

Gateway Development Implementation

The typical development process of gateway products is as follows:



The three steps within the dotted line are implemented by the gateway manufacturer. It is usually recommended as follows:

- For gateway devices connected to the internet via Wi-Fi, it is necessary to refer to Wi-Fi Distribution Network Protocol and cooperate with Tencent Lianlian Mini Program to complete the configuration.
- For the communication between the gateway and the subdevice, the platform has no restrictions and it is freely defined by the gateway manufacturer and the subdevice. However, for the gateway to implement device information management and legitimacy verification of the subdevice, it needs to map well with the subdevice associated with the cloud. Generally, the unique information (MAC/EUI) of the subdevice is used as the device

name of the platform subdevice, and the gateway manages the mapping relationship during the inbound process of the subdevice.

- The communication protocol between the gateway and IoT Explorer must be Data Template Protocol. Each product type and data template are different. The communication between the gateway and the subdevice is generally binary. Therefore, the gateway needs to implement two-way conversion from the binary data of the subdevice to the data of the Data Template Protocol based on different types of products. The general practice is to create two message queues: the message queue from the subdevice to the gateway `stack_msg_queue` and the message queue from the cloud to the gateway `cloud_msg_queue`. Independently retrieve messages from the message queues in an independent thread, perform corresponding conversions, and then distribute and push the messages.
- For Bluetooth scenarios, Tencent has defined LianLian LLSync Protocol. For gateways integrated with Tencent IoT Platform, it is recommend using LianLian LLSync Protocol as the communication protocol between the gateway and sub-devices.

Sub-Device Access Guide

Last updated: 2025-04-28 10:46:15

The IoT Explorer C SDK is oriented to devices that can communicate directly with the platform. Since subdevices are unable to communicate directly with the platform, they do not need to integrate the IoT Explorer C SDK. Theoretically, users can fully customize the interaction logic between subdevices and gateways. However, to enable subdevices to interact with the platform, this document provides some commonly used handling suggestions.

Sub-Device Information Management

The gateway needs to get the device information of subdevices before it can proxy them to realize corresponding features.

Sub-Device Discovery and Authentication

The discovery logic of gateways for subdevices can be implemented freely based on different communication methods (for example, BLE, Zigbee, and 485). However, after device discovery, it is usually recommended to perform legitimacy authentication of devices. Authentication can be implemented by combining the device information of the platform. It is usually recommended that the product corresponding to a subdevice select the key authentication method. Each subdevice has corresponding triplet information on the platform, that is, productId + deviceName + psk. If the gateway side saves the psk information of a subdevice, signature verification can be performed based on the psk.

1. Click the Discover Sub-devices button in the Tencent Lianlian Mini Program.
   There are two ways to enter the process of binding a primary device in the Tencent Lianlian Mini Program:
   - On the gateway device interface, click the "+" button to add a sub-device.
   - Scan the Device QR Code in the Batch Production section of the Product Development interface for the sub-device in the console.

2. The gateway device will receive a search_devices message. At this point, the gateway device can start the process of discovering and searching for subdevices. The gateway receives the payload for searching subdevices: `{"payload":{"status":1},"type":"search_devices"}` . The gateway replies with the payload to start search:

```
{"type":"search_devices",
"payload":{"status":1, "result":0}}
```

3. The search logic here is implemented by the customer. After successfully discovering subdevices (discovery conditions are customer-defined), the subdevice can be bound to the gateway. For specific binding, refer to the implementation of IOT_Gateway_Subdev_Bind. Then, the Mini Program Interface will display the device that

was successfully bound just now.

Note:
A subdevice generally refers to a device that cannot directly connect to IoT Explorer and needs to connect to the platform through a gateway. A gateway product needs to add a subproduct before it can add subdevices under the gateway device. Please refer here for the description of the topological relationship and parameter fields of gateway subdevices.

Dynamic Binding of Sub-Devices

The topological binding relationship between gateway devices and subdevices can be pre-completed on the console or dynamically bound during the running process. The prerequisite for gateway devices and subdevices to perform dynamic binding is that the binding relationship between the gateway product and the subdevice product has been completed on the console. After a gateway device discovers a subdevice, the judgment logic on whether to initiate dynamic binding needs to be developed and implemented by the gateway side based on the device management logic of the gateway side. Dynamic binding requires signature information based on the subdevice key. For signature calculation, refer to the C SDK API `subdev_bind_hmac_sha1_cal`. The signature calculation can be sent to the gateway after

being completed on the subdevice or completed on the gateway side. The latter requires the gateway to save the psk information of the subdevice.
A successful binding log, see the following content:

```
DBG|2022-01-14
16:57:25|mqtt_client_publish.c|qcloud_iot_mqtt_publish(346): publish
packetID=0|topicName=$gateway/operation/TZ9Y9CLTEA/gateway_001|payload=
{"type":"bind","payload":{"devices":
[{"product_id":"xxxx","device_name":"subdev_003","signature":"xxxxx","ra
ndom":1820758684,"timestamp":1642150644,"signmethod":"hmacsha1","authtyp
e":"psk"}]}}
INF|2022-01-14 16:57:25|gateway_sample.c|_message_handler(138): Receive
Message With topicName:$thing/down/property/HW9ME416BI/subdev_002,
payload:{"method":"report_reply","clientToken":"xxxx-
45","code":0,"status":"success"}
DBG|2022-01-14 16:57:25|gateway_common.c|_gateway_message_handler(419):
gateway recv : {"type":"bind","payload":{"devices":
[{"result":0,"product_id":"HW9ME416BI","device_name":"subdev_003"}]}}
INF|2022-01-14 16:57:25|gateway_common.c|_gateway_message_handler(510):
client_id(HW9ME416BI/subdev_003), bind result 0
DBG|2022-01-14 16:57:25|gateway_sample.c|show_subdev_bind_unbind(262):
bind HW9ME416BI/subdev_003 success
```

Dynamic Registration of Sub-Devices

Considering the mass production convenience of the product, subdevices can use the dynamic registration method to obtain the triplet information of subdevices. Dynamic registration requires the product key of the subdevice as a signature. The calculation of the signature can be sent to the gateway after being completed on the subdevice side, or it can be completed on the gateway side. The latter requires the gateway to save the psk information of the subdevice. It is recommended that the name of the dynamically registered subdevice be unique information that the gateway can obtain during the device discovery process, such as MAC address, EUI, etc.

Sub-Device Communication

Gateway and Subdevice Communicate

The communication protocol and data format between the gateway and subdevices are not restricted by the platform. However, when the gateway proxy communicates with the cloud on behalf of the subdevice, it needs to convert the messages of the subdevice into formatted data of data template.

Sub-Device and Sub-Device Interconnection

Configure the message forwarding rules between subdevices through the console or Mini Program Configuration to achieve device interconnection between subdevices.

Audio/Video Device Development P2P Access Guide

Last updated：2025-04-28 10:47:01

The Internet of Things development platform provides users with P2P access capability based on the X-P2P protocol. This document introduces relevant content about P2P access.

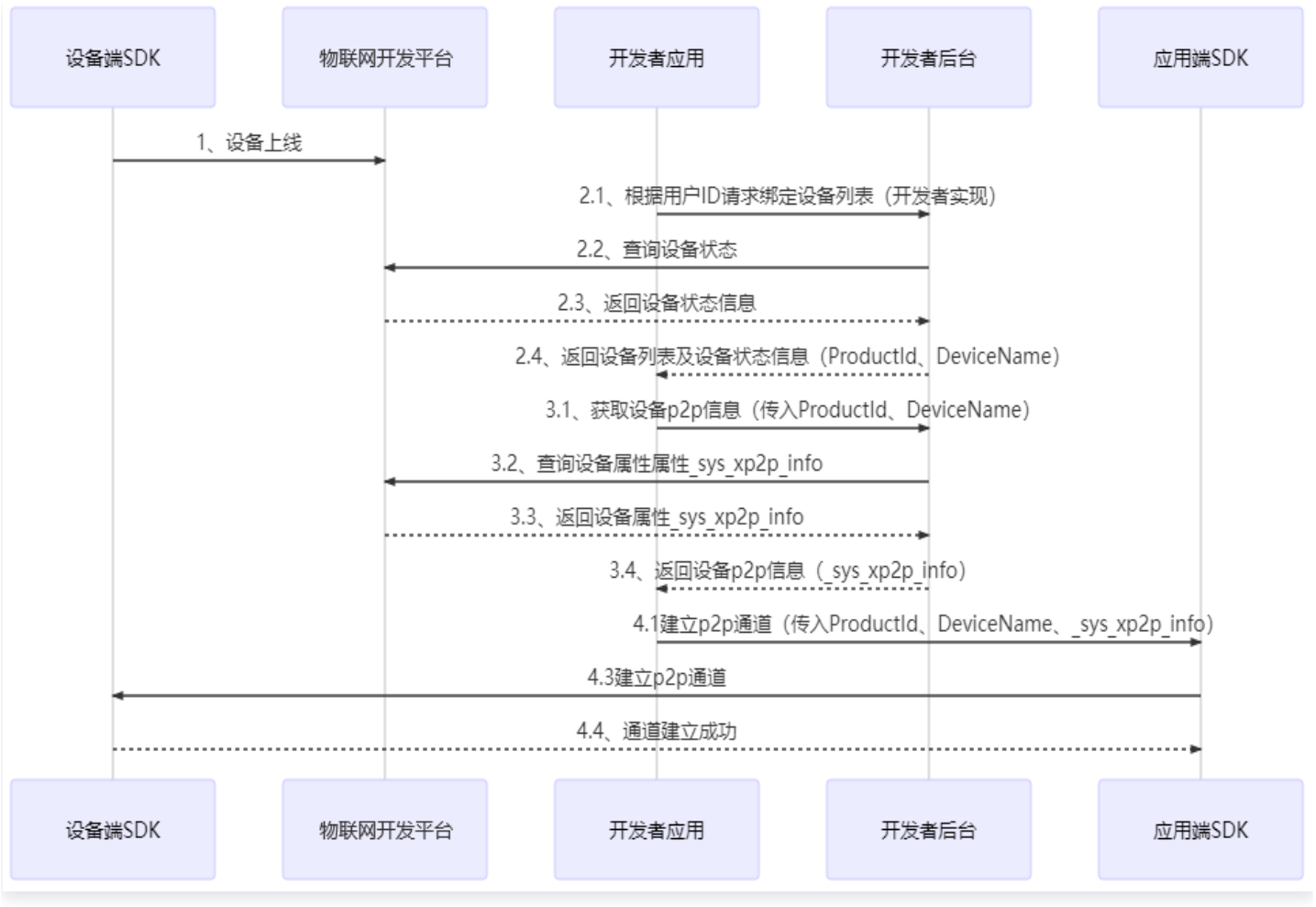
Feature Overview

- Currently, one device supports the creation of a maximum of 4 P2P transmission channels.
- P2P channels support not only audio/video transmission but also user-defined data transmission.

Operation Steps

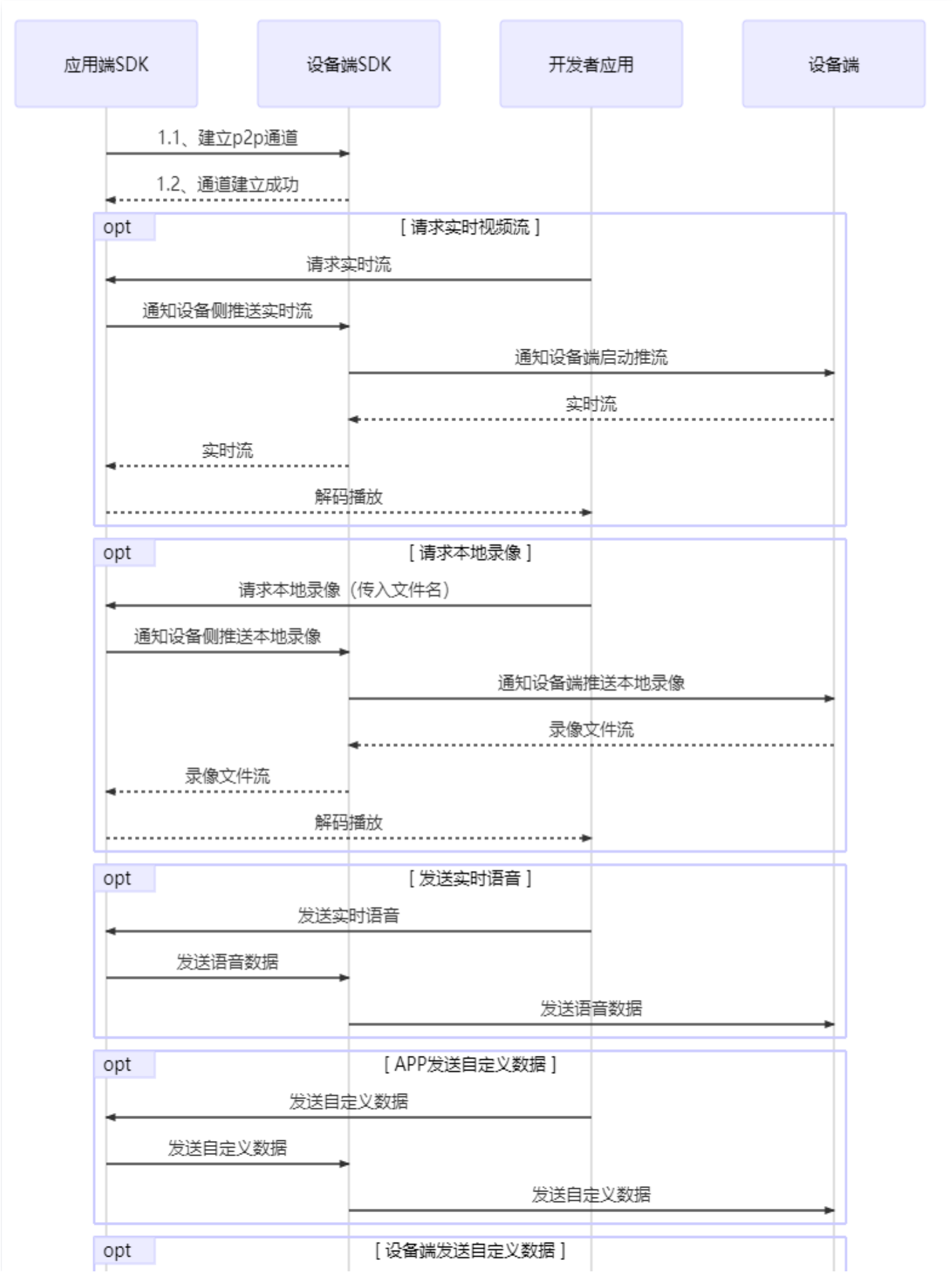
Establishing a P2P Connection

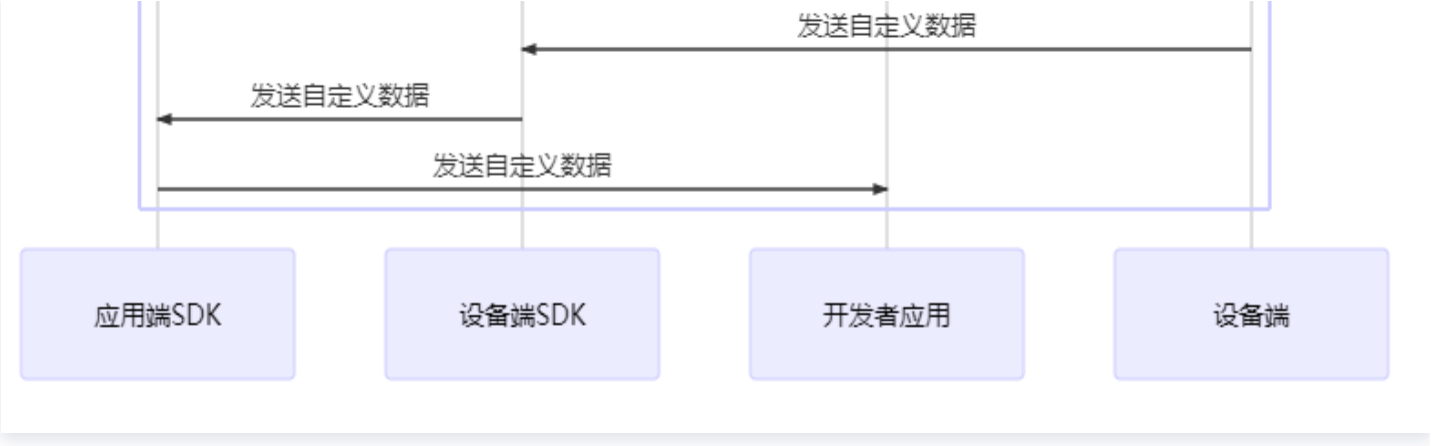
Users need to use X-P2P to perform audio and video transmission between the application side and the bound device side. They need to establish a P2P connection first. Users query device status and device properties through the IoT development platform console to obtain _sys_xp2p_info. The specific process is as follows:



P2P Audio and Video Transmission

After the application side and device side have established a P2P channel, audio and video and custom data transmission can be performed. The process is as follows:





Cloud Storage Access Guide

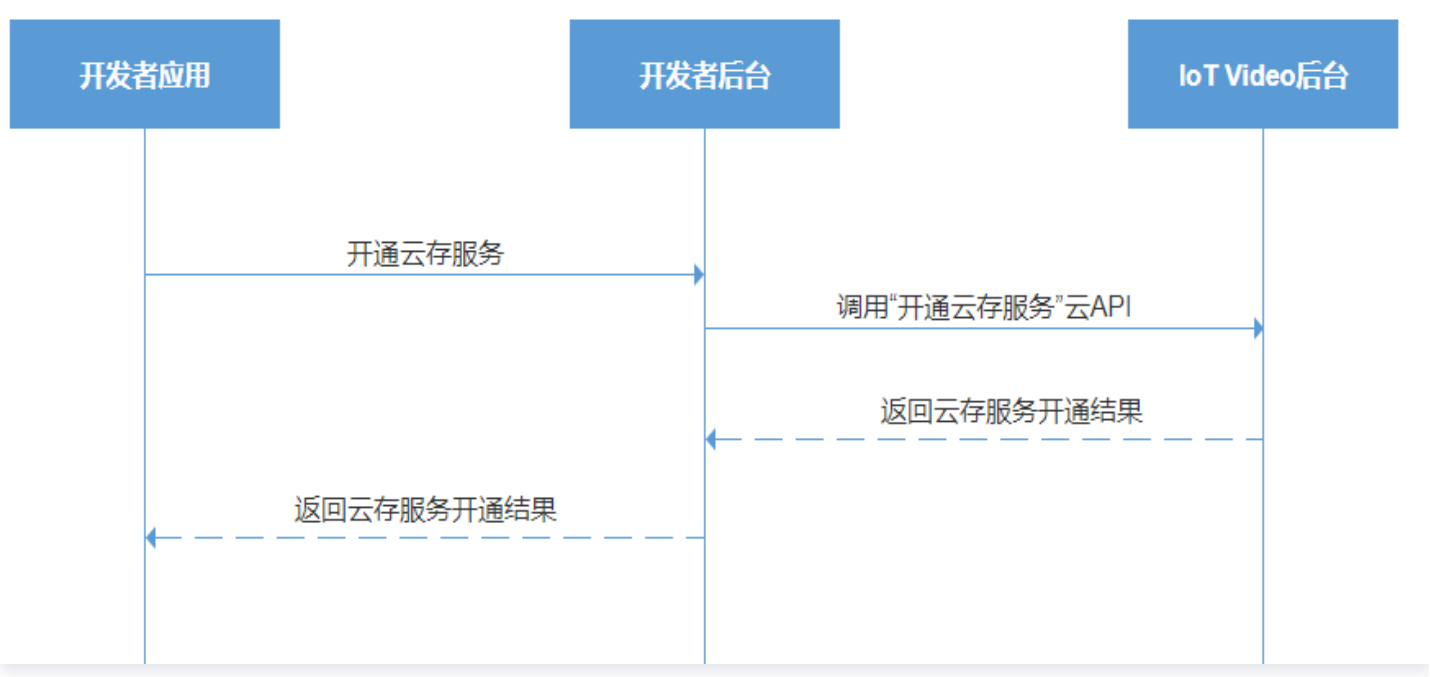
Last updated: 2025-04-28 10:47:17

The IoT development platform provides users with a cloud-based storage feature. It supports the storage of audio and video data from the device side in the cloud. When viewing is required, the application side pulls data from the cloud.

Operation Steps

Enabling Cloud Storage Package

Before using the cloud storage feature, you need to enable the Cloud Storage Package for the device. By calling the TencentCloud API, you can purchase a cloud storage package for the device. After the cloud storage package is successfully purchased, the device will obtain the status of the device's cloud storage package through the Thing Model.



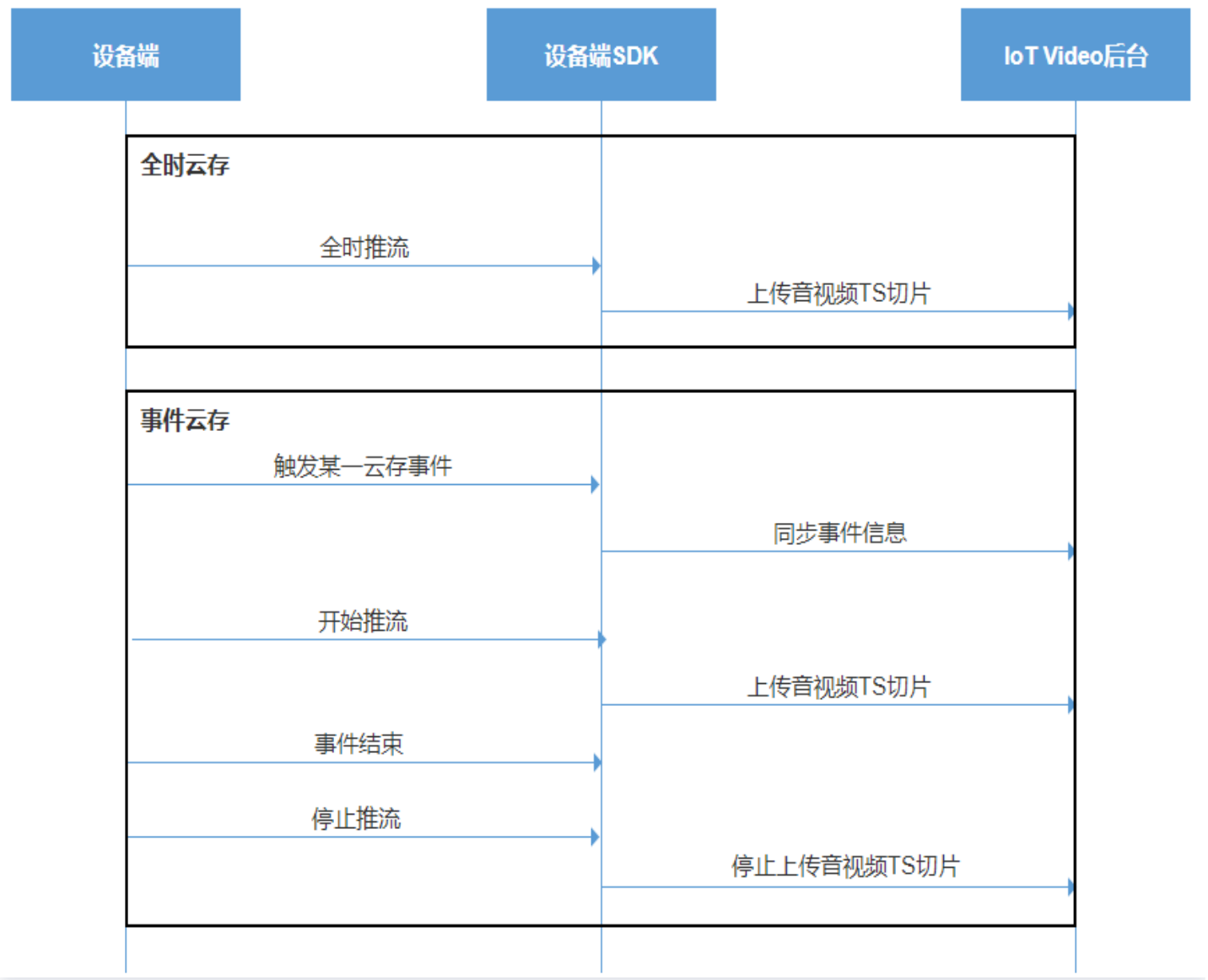
Cloud Storage Recording Upload (Video Stream Device)

Cloud Storage Packages are divided into Full-time Cloud Storage Package and Event Cloud Storage Package:

- **All-time Cloud Storage**: Refers to the ability of devices to upload all generated video recordings to cloud storage for storage during the package activation time.
- **Event Cloud Storage**: Refers to the ability of devices to upload video recordings of events to the cloud for storage when an event trigger occurs during the package activation time.

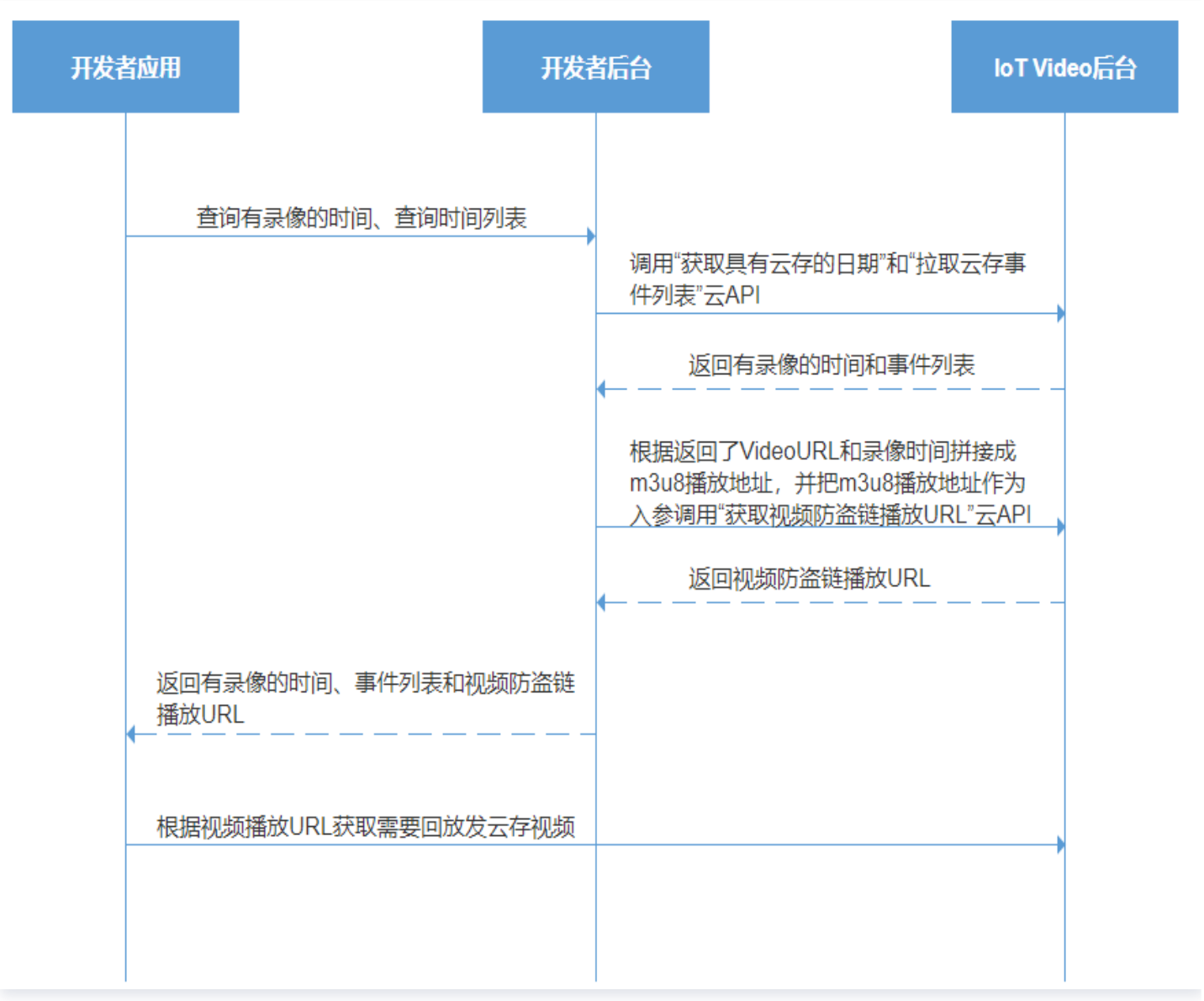
> **Note:**
> A device can exist only one Cloud Storage Package at the same time.

The process of uploading video recordings to cloud storage is as follows:



Viewing Cloud Storage Recordings (Video Stream Device)

IoT Video (Consumer Version) provides TencentCloud API support for querying dates with video recordings and event lists, and concatenates the returned dates and VideoURL into an m3u8 playback address to obtain the video stream through the m3u8 address.



m3u8 Playback Address Concatenation

1. Retrieve the Cloud Storage Date by calling the acquire Cloud Storage Date API.
2. Call the get Cloud Storage Timeline for a certain day API to retrieve the playback address (VideoUrl) of the cloud storage recording for the specified date and the start time (StartTime) and end time (EndTime) of each recording clip.
3. Concatenate the returned VideoUrl, StartTime, and EndTime in the following format in UNIX timestamp format.

```
{VideoUrl}?starttime_epoch={StartTime}&endtime_epoch={EndTime}
```

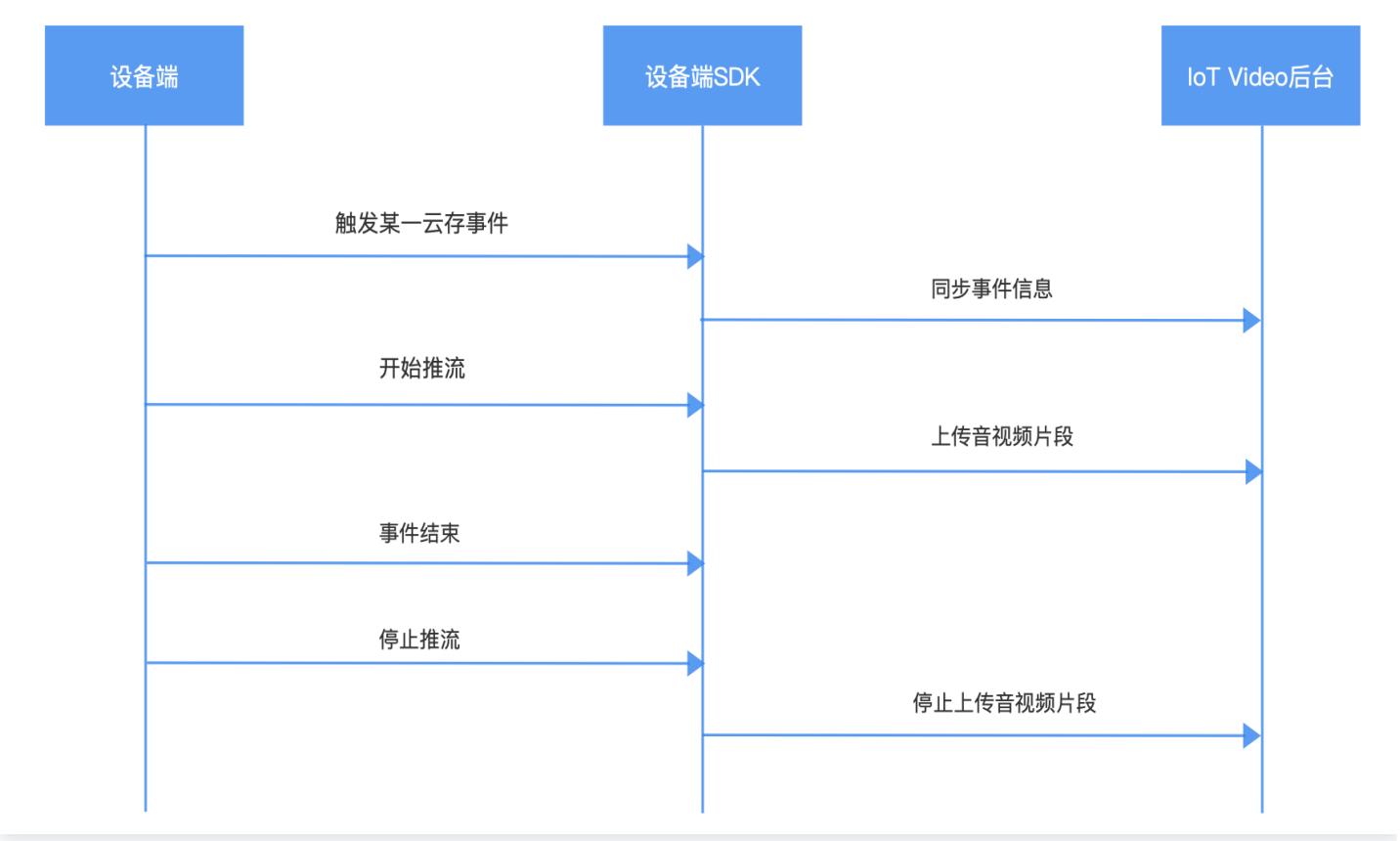
4. Call the get anti-hotlinking playback URL for video API with the concatenated VideoURL as a calling parameter to obtain the anti-hotlinking playback address of the cloud storage

video, and then use the Mini Program Plugin capacity to achieve normal playback.

Cloud Storage Recording Upload (Image Stream Device)

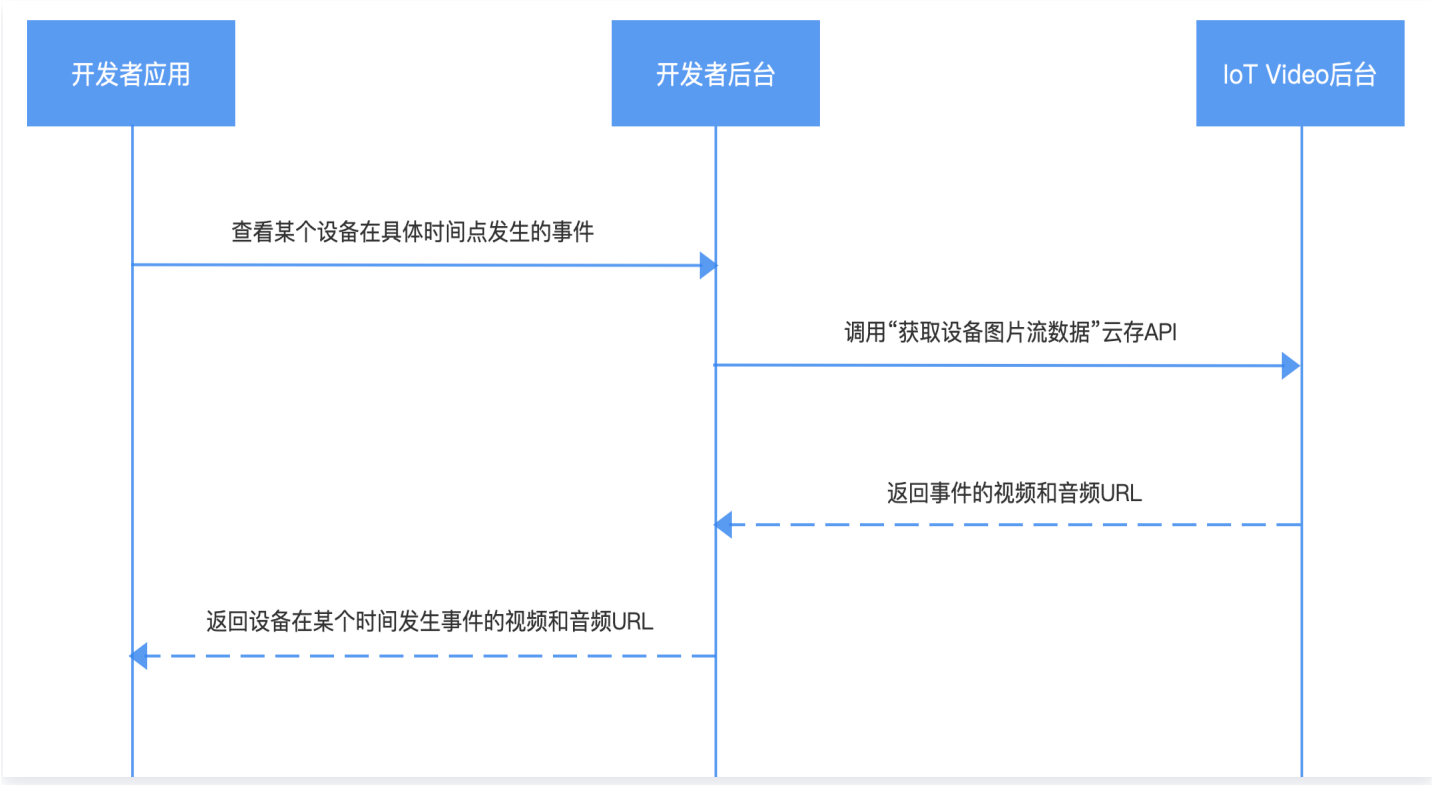
There is only an Event Cloud Storage Package in Cloud Storage Packages.
Event Cloud Storage: It refers to that when an event is triggered during the package activation time, the device can upload the video recording taken when the event occurs to the cloud for storage.

> **Note:**
> A device can have only one Cloud Storage Package at the same time.



Viewing Cloud Storage Recordings (Image Stream Device)

The platform offers TencentCloud API support for retrieving Event Cloud Storage video recordings of image stream devices (mjpg format video + acc format audio).



Get Image Stream Playback Address

1. Retrieve the Event Cloud Storage Date by calling the acquire Cloud Storage Date API.
2. Retrieve the event list on a certain date by calling the retrieve cloud storage event list API, including each event's StartTime (start time), EndTime (end time), Thumbnail (thumbnail file name), and EventId (event ID).
3. Call the pull Cloud Storage Event thumbnail API to get the access address of the specified event thumbnail.
4. Call the retrieve device image stream data API to obtain the playback addresses of the event's image stream video and audio, and then utilize the Mini Program Plugin capability to achieve normal playback.

> ⚠ **Note:**
> The storage time of each event record does not exceed 10 s.

Signal Interaction Instructions between Device Side and Application Side

Last updated: 2025-04-28 10:47:33

Overview

After the P2P transmission channel between the underlying application side and device side is established, P2P message sending and audio and video transmission features will be implemented based on this channel. The following will describe in detail the signaling interaction process between the device side and the application side, which is divided into audio and video transmission (the device pushes audio and video streams to the application side), intercom audio and video transmission (the application side pushes audio and video streams to the device side, suitable for voice intercom or audio and video intercom), receiving signaling (the application side sends messages to the device), and sending signaling (the device sends information to the application side).

> **Note:**
> Unmentioned parameters are internal parameters. Do not use them.

Audio and Video Transmission

Request Method get
Request path /ipc.flv

Basic Request Parameters

_crypto

Type: optional
Visibility: external parameters
Description: encryption switch. The value is on or off. If this parameter does not exist or is not off, encryption is enabled by default.
Example

```
_crypto=off
```

action

Type: Required
Visibility: external parameters
Description: The type of this stream pull request can be live for live stream, playback for replay, or download for download.
Example

```
action=live
```

channel

Type: Required
Visibility: external parameters
Description: The channel number of this stream pull request. For a single camera, it is 0. For a multi-camera device, it refers to the sequence number of the camera. For an nvr device, it refers to the sequence number of the data. Counting begins from 0.
Example

```
channel=0
```

user_args

Type: Selectable
Visibility: external parameters
Description: User-defined parameter. It should not be too long. Keep it below 1 KB. Content and format are unlimited, but illegal URL characters should not appear.
Example

```
user_args=this_is_user_args%E4%BD%A0%E5%A5%BD
```

requester

Type: Optional (recommended to be a required parameter in subsequent development)
Visibility: external parameters
Description: The role that initiates the stream pull request. It should not be too long and is recommended to be within 64 bytes. Currently, there are several types: app-android, app-ios, wxmp-android, wxmp-ios, server-voip. If necessary, additional information such as version numbers can be included, for example: app-android-xiaomi, wxmp-ios-p2p-player-version123.
Example

```
requester=app-android
```

Live Streaming Request

Used for viewing real-time trtc content from the camera
live stream request parameter action = live

Additional Request Parameters

quality

Type: Required
Visibility: external parameters
Description: The video quality of this stream pull request has three levels: low, medium, high, which are standard, high, super, respectively.
Example

```
quality=standard
```

Request Example

```
http://ipc.p2p.com/ipc.flv?
action=live&channel=0&quality=high&_token=123456789&_peername=xxxxxxxx&r
equester=app-
android&req_id=xxxxxxxx&user_args=this_is_user_args%E4%BD%A0%E5%A5%BD
```

Replay Request

Used for replaying video recordings on the camera's SD card
Replay request parameter action=playback

Additional Request Parameters

start_time

Type: Required
Visibility: external parameters
Description: Request the start time of the replay, UNIX second-level timestamp. The interval with end_time is recommended to be more than 5 seconds. Too small an interval may cause an index error.
Example

```
start_time=123456789
```

end_time

Required
Visibility: external parameters
Description: The end time of the replay request, a UNIX second-level timestamp. The interval with start_time is recommended to be more than 5 seconds. Too small an interval may cause an index error.
Example

```
end_time=123456789
```

Request Sample

```
http://ipc.p2p.com/ipc.flv?
action=playback&channel=0&start_time=1633060850&end_time=1633060880&_tok
en=123456789&_peername=xxxxxxxx
```

Download Request

Used for downloading files in the SD card, in conjunction with the "query file list" signaling.
Replay request parameter action=download

Additional Request Parameters

file_name

Type: Required
Visibility: external parameters
Description: The name of the downloaded file. If it contains a file path, please perform URL encoding.
Example

```
file_name=test_file.mp4
```

offset

Type: Required
Visibility: external parameters

Description: The starting position of the downloaded file. Users can develop the breakpoint resume function with the aid of this parameter.
Example

```
offset=0
```

Request Sample

```
http://ipc.p2p.com/ipc.flv?
action=download&channel=0&file_name=test_file.mp4&offset=0&_crypto=off&_
peername=xxxxxxxx
```

Intercom Audio and Video Transmission

Request method: post.
Request path /voice

Request Parameters

Basic Request Parameters

Refer to other parameters in "audio and video transmission – basic request parameters" except action and channel.

Additional Request Parameters

channel

Type: Selectable for camera devices, required for NVR devices.
Visibility: external parameters
Description: The channel number of this intercom request. This parameter is not yet used for camera devices and can be omitted; for NVR devices, it refers to the sequence number, counting begins from 0.
Example

```
channel=0
```

Request Example

```
http://ipc.p2p.com/voice?channel=0&_token=123456789&_peername=xxxxxxxx
```

Receiving Signaling

Request Method post
Request path /command
Encryption is enabled by default and cannot be closed.

Request Parameters

action

Required
Visibility: external parameters
Description: `inner_define` is an internal signaling. Do not use it; `user_define` is an external signaling, i.e., custom signaling.
Example

```
action=inner_define
action=user_define
```

channel

Type: Selectable for camera devices, required for NVR devices.
Visibility: external parameters
Description: The channel number of this intercom request. This parameter is not yet used for camera devices and can be omitted; for NVR devices, it refers to the sequence number, counting begins from 0.
Example

```
channel=0
```

cmd

Type: Required
Visibility: external parameters
Description: Message type
Example

```
cmd=my_message
```

Internal Signaling

Local Recording Playback Related Signaling

Query Video Recordings by Start and End Time

Request Parameter

Key	Value	Description
cmd	get_record_index	Signaling Type
start_time	UNIX timestamp	Query the start time of the video recording, in seconds. It is recommended that the interval with the end_time be more than 5 seconds. An interval that is too small may cause an index error.
end_time	UNIX timestamp	The end time of querying the video recording, in seconds. The interval with start_time is recommended to be more than 5 seconds. Too small an interval may cause an index error.
type	Error Example	Indicates the type of video recording, which is an integer. The user defines its specific meaning. Currently not supported.

Return Value

json format video recording index list, basic format as follows:

```
{
    "video_list": [
        {
            "type": 0,
            "start_time": "<unix timestamp>"
            "end_time": "<unix timestamp>"
        },
        {
            "type": 0,
            "start_time": "<unix timestamp>"
            "end_time": "<unix timestamp>"
        }
    ]
}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=get_record_i
```

```
ndex&start_time=000&end_time=111
```

Monthly Query of Video Recordings

Request Parameter

Key	Value	Description
cmd	get_month_record	Signaling Type
time	Time	A 6-bit integer, the first 4 bits represent the year, and the last 2 bits represent the month

Return Value
json format, where the value of `video_list` is converted to a 32-bit integer. Each bit from the low position to the high position represents whether there is a video recording on the corresponding day of the month; for example: 8320 (0010000010000000) means there are video recordings on the 8th and 14th.

```
{"video_list":"123456"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=get_month_re
cord&time=202401
```

Replay Suspension

Request Parameter

Key	Value	Description
cmd	playback_pause	Signaling Type

Return Value
json format, json format, `<code>` is the signaling execution result return value, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_see
k&progress=123456
```

Replay Continues

Request Parameter

Key	Value	Description
cmd	playback_resume	Signaling Type

Return Value
json format, json format, `<code>` is the signaling execution result return value, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_res
ume
```

Set Playback Position

Request Parameter

Key	Value	Description
cmd	playback_seek	Signaling Type
progress	Integer	Timestamp in millisecond unit, indicating the playable position within the current file where you hope to redirect for playback.

Return Value
json format, json format, `<code>` is the return value of the signaling execution result. 0 means normal, and other values indicate exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_see
k&progress=123456
```

Get Playback Position

Request Parameter

Key	Value	Description
cmd	playback_progress	Signaling Type

Return Value
json format, json format, `<code>` is the signaling execution result return value, 0 for normal, other for exceptions; `<timestamp>` is the current playback progress in milliseconds.

```
{
    "status":"<code>",
    "progress":"<timestamp>"
}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_pro
gress
```

Set Fast-Forward

Request Parameter

Key	Value	Description
cmd	playback_ff	Signaling Type
value	Integer	• 0 means normal playback. • 1 means playing only I frames. • 2 means playing one I frame from two adjacent I frames.

		• 3 means playing one I frame from three adjacent I frames. • And so on

Return Value
json format. json format. `<code>` is the return value of signaling execution result. 0 means normal, and others mean exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_ff&
value=1
```

Set Fast Refund

Request Parameter

Key	Value	Description
cmd	playback_rewind	Signaling Type
start_time	Timestamp	Milliseconds, indicating the end time of reverse playback.
end_time	Timestamp	Milliseconds, indicates the start time of reverse playback.

Return Value
json format, json format, `<code>` is the return value of signaling execution result, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_rew
ind&start_time=100&end_time=200
```

Set Playback Speed

Request Parameter

<Key.>	Value	Description
cmd	playback_speed	Signaling Type
speed	Integer	Unit milliseconds, which indicates the frame interval. Synchronize based on the video. Assume the normal frame rate of the video is 25fps, i.e., the frame interval is 50ms. A value smaller than the normal frame interval means fast playback. For example, setting it to 10 is equivalent to 5x fast playback; a value greater than the normal frame interval means slow playback. For example, setting it to 100 is equivalent to 0.5x slow playback.

Return Value
json format, json format, `<code>` is the return value of signaling execution result, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=playback_spe
ed&speed=50
```

TWeCall and Voice Intercom Related Signaling

Hangup

Sent by the client to the device, used to answer later, indicating proactive hang up.
Request Parameter

Key	<Value>	Description
cmd	call_hang_up	Signaling Type

Return Value

json format, json format, `<code>` is the signaling execution result return value, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=call_hang_up
```

Reject Call

Description: Sent from the client side to the device side, used before answering to indicate rejection.
Request Parameter

Key	<Value>	Description
cmd	call_reject	Signaling Type

Return Value
json format, json format, `<code>` is the signaling execution result return value, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=call_reject
```

Cancel

Description: Sent from the client side to the device side, used before answering to proactively cancel the call.
Request Parameter

Key	Value	Description
cmd	call_cancel	Signaling Type

Return Value
json format. json format. `<code>` is the signaling execution result return value, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=call_cancel
```

Line Busy

Description: Sent from the client side to the device side, used after answering to indicate that another device is in an ongoing call.
Request Parameter

Key	Value	Description
cmd	call_busy	Signaling Type

Return Value
json format, json format, `<code>` is the return value of signaling execution result, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=call_busy
```

No Response

Description: Sent from the client side to the device side, used before answering to indicate that the call timed out and was not answered.
Request Parameter

Key	Value	Description
cmd	call_timeout	Signaling Type

Return Value
json format, json format, `<code>` is the return value of the signaling execution result, 0 for normal, other for exceptions.

```
{"status":"<code>"}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=call_timeout
```

Other Signaling

Query Device Status

Request Parameter

Key	Value	Description
cmd	get_device_st	Signaling Type
type	live / voice / playback	Required, ask the device whether to accept live stream, intercom, and replay requests.
quality	standard / high / super	Required when the type is live, ask the device whether it supports live streaming requests with low, medium, and high image quality.

Return Value
json format, status code, 0 means accepting connection request, other values mean refusing connection request; `appConnectNum` the number of Apps or mini-programs currently connected to the device; `maxConnectNum` the maximum number of connections supported by the device.

```
[
    {
        "status":"code",
        "appConnectNum":"2",
        "maxConnectNum": "3"
    }
]
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=get_device_s
t&type=live&quality=standard
```

Query File List

Request Parameter

Key	Value	Description
cmd	get_file_list	Signaling Type

Return Value

json format, `file_type` file type; `file_name` file name; `file_size` file size; `start_time` start time; `end_time` end time; `extra_info` extended information, which can be used to carry hash value or other custom file information.

```
{
    "file_list": [
        {
            "file_type": "video",
            "file_name": "1234.mp4",
            "file_size": "1024000",
            "start_time": "2024-01-23_12-34-00",
            "end_time": "2024-01-23_12-34-59",
            "extra_info": {
                "md5": "abvdefg"
            }
        },
        {
            "file_type": "picture",
            "file_name": "abcd.jpg",
            "file_size": "10240",
            "start_time": "2024-01-23_12-34-56",
            "end_time": "2024-01-23_12-34-56"
        }
    ]
}
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=get_file_lis
t
```

Query Device Names under NVR

Request Parameter

Key	Value	Description
cmd	get_nvr_list	Signaling Type

Return Value
json format, `DeviceName` device name; `Channel` located in which channel; `Online` whether online.

```
[
    {
        "DeviceName": "name1",
        "Channel": "1",
        "Online": "0"
    },
    {
        "DeviceName": "name2",
        "Channel": "2",
        "Online": "1"
    }
]
```

Request sample:

```
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=inner_define&cmd=get_nvr_list
```

External Signaling

i.e., user-customized signaling
Request Parameter

Key	Value	Description

cmd	signaling content	User-customized signaling should not be too long. Keep it below 1 KB. Content and format are unlimited, but illegal URL characters should not appear.

Return Value
Unlimited format and type. It should not be too long. Keep it below 1 KB.
Request sample:

```
Custom signaling for performing Panning Control example:
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=user_define&cmd=ptz_left_spee
d_1
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=user_define&cmd=ptz_right_spe
ed_1
Transmit a small amount of binary data via custom signaling:
http://ipc.p2p.com/command?
_token=123456789&_peername=xxxxxxxx&action=user_define&cmd=36D7F336FCEC9
E57318ECA0323BA94970296A776CE028D19F5CA5AC44E92952B4CE77642354CBAE7036F4
DA954317462C61A1BC033DAC882FAAB9EDCBFEE47DA
```

Sending Signaling

Request Method: post
Request path /feedback
Encryption is enabled by default and cannot be closed.
Messages are sent in the HTTP message body using json format.

Request Parameters

Key	Value	Description
product_id	Device triple information	For scenarios such as device identification, message management for the other party to recognize
device_name	Device triple information	For scenarios such as equipment identification, message management, etc. for the other party to identify

Request Example

```
http://127.0.0.1:34567/ipc.p2p.com/forward/xxxxxxxxxxxx/proxy.sample.se
rver/feedback?_token=123456789&product_id=xxxxxxxx&device_name=xxxxxxxx
```

User-Customized Signaling

The custom message format adopted by the user shall be in json format. It shall not contain the `iv_private_cmd` field. The remaining content is unlimited. The length should not be too large. Keep it below 16 KB.
Example

```
send
{
"my_message":"hello world"
}
receipt
{
"status":"<code>"
}
```

`<code>` is the return value. 0 means normal, and other values indicate exceptions.

Internal Signaling

Internal messages use the following json format.

```
send
{"iv_private_cmd":"<cmd>"}
receive
{"status":"<code>"}
```

Hangup

Description: Sent from the device side to the client side, used to answer later, indicating proactive hangup.

```
{"iv_private_cmd":"call_hang_up"}
```

Response

```
{"status":"<code>"}
```

<code> is the return value. 0 means normal, and other values indicate exceptions.

Cancel

Description: Sent from the device side to the client side, used before answering to proactively cancel the call.

```
{"iv_private_cmd":"call_cancel"}
```

Response

```
{"status":"<code>"}
```

<code> is the return value. 0 means normal, and other values indicate exceptions.

Other Signaling

Device Actively Stops Receiving Audio/Video Data Sent by Peer

Description: Sent from the device side to the client side, used in the two-way intercom scenario.

```
{"iv_private_cmd":"recv_stop"}
```

Response

```
{"status":"<code>"}
```

<code> is the return value. 0 means normal, and other values indicate exceptions.

Tencent Lianlian Mini Program Interactive Signaling Description

Last updated: 2025-04-28 10:47:49

The signaling interaction protocol used for device-side integration with the Tencent Lianlian Mini Program IoT Video standard panel.

Request Method

```
/**
 * command is a string or object
 */
function sendCommand(command) {
    message_id++
    if (typeof command !== 'string') command =
'action=user_define&channel=0&cmd=' + JSON.stringify(command)
    return p2pExports.sendCommand(id, command)
}
```

Internal Signaling

Refer to Signal Interaction Description between Equipment Side and Application Side_Internal Signal.

External Signaling

Operating the Gimbal

Request Field

Field	Type	Description
topic	string	ptz operation type.
message_id	number | string	Request ID.
cmd	string	• ptz_release_pre: release. • ptz_up_press: Up. • ptz_right_press: Rotate right. • ptz_down_press: Down.

		• ptz_left_press: Left.

Request Example

```
{
    "topic": "ptz",
    "message_id": 0,
    "data": {
        "cmd": "ptz_release_pre"
    }
}
```

Returned Field

Field	Type	Description
apex	string	Whether it is at the top. • yes: Reached the top. • no: Not at the top.
current_x	number	Horizontal position.
current_y	number	Vertical position.
max_x	number	Maximum horizontal position
max_y	number	Maximum vertical position
min_x	number	Minimum horizontal position.
min_y	number	Minimum vertical position.

Response Sample

```
{
    "confirmation_topic": "ptz",
    "result": "0",
    "apex": "yes",
      "current_x": 123,
      "current_y": 123,
      "max_x": 123,
      "max_y": 123,
```

```
        "min_x": 123,
        "min_y": 123,
    "message_id": 0
}
```