

IoT Hub

Getting Started

Product Documentation



Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

Contents

Getting Started

Quick Start

Scenario 1: Device Interconnection

Overview

Console Operation Steps

Device-Side Operation Steps

Scenario 2: Device Status Reporting and Setting

Scenario Overview

Device Status Reporting

Device Temperature Setting

MQTT.fx Connection Guide

Getting Started

Quick Start

Last updated : 2021-10-25 10:52:23

To get started quickly, you are advised to connect to IoT Hub using MQTT.fx. Find the connection guide as follows:

- [MQTT.fx Connection Guide](#)
- To use sites other than Chinese mainland, see [Device Connection Regions](#) to select a suitable site for connection.

We have built a smart home demo to demonstrate the features of IoT Hub. Scenarios include:

- Scenario 1: the door is connected with the air conditioner, and actions of entering/leaving the house are monitored through the door to instruct the air conditioner to turn on/off.
- Scenario 2: It describes how the user can set the air conditioner temperature and check the energy consumption of the air conditioner using an app.

The demo involves the following features of IoT Hub:

- Message publishing and subscribing
- Device shadow-based status reporting and configuration distribution
- Rule engine-based device message communication

You can refer to the demo and customize based on your own needs:

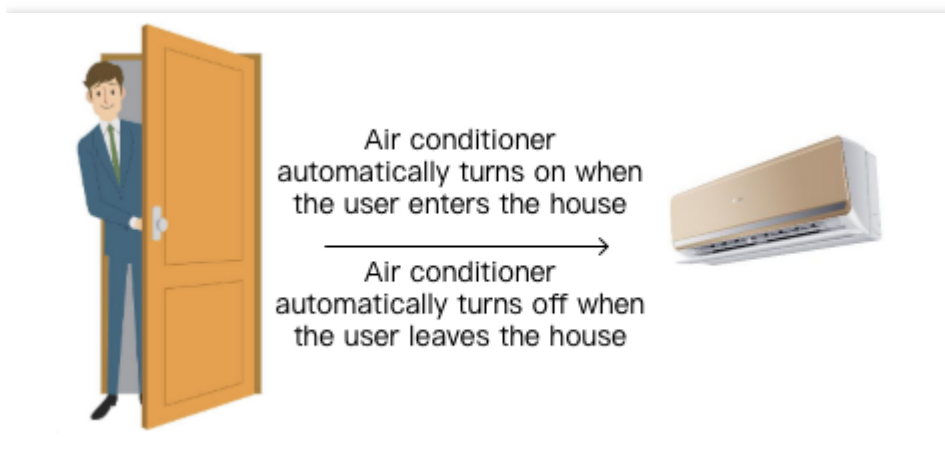
Demo	Content
Scenario 1	Device Interconnection
Scenario 2	Device Status Reporting and Setting

Scenario 1: Device Interconnection Overview

Last updated : 2021-09-24 17:48:30

Overview

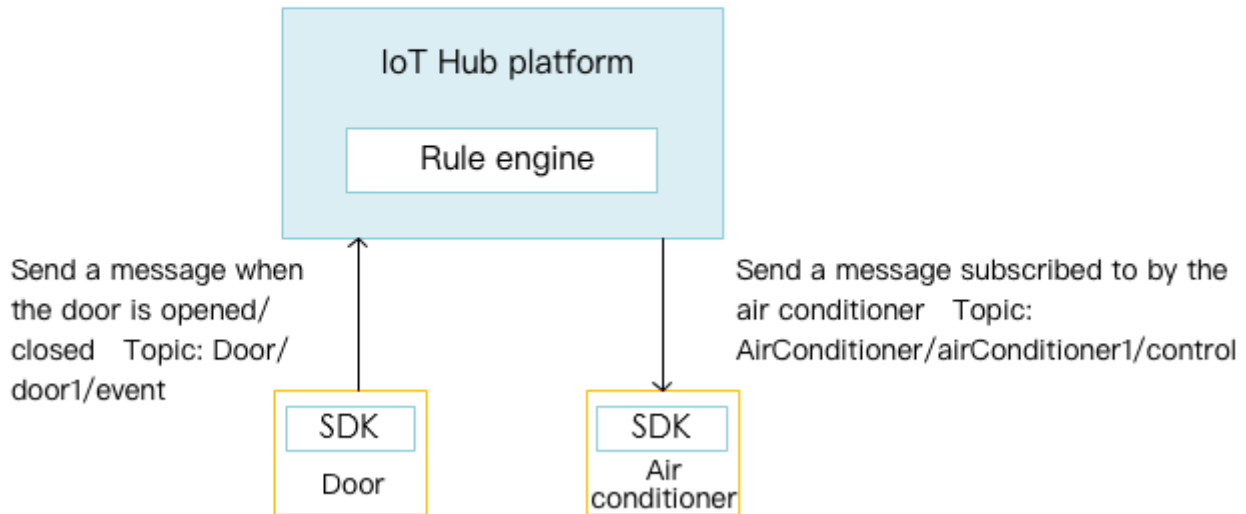
If you need to achieve the features as shown below in a smart home scenario (this is not a real product but only used to demonstrate IoT Hub's capabilities), you can follow the steps in this document.



Solution

Two types of smart devices (door and air conditioner) can be created in the IoT Hub SDK and connected with each other based on cross-device messaging and the rule engine as shown below:

Configure a repub (forwarding) rule in the IoT Hub console: set to repub (forward) messages of the Door/door1/event as messages of the AirConditioner/airConditioner1/control topic



Note :

`airConditioner1` cannot achieve message communication by directly subscribing to the update messages of `door1` . For the reason, please see [Feature Components - Permission Management](#).

Console Operation Steps

Last updated : 2021-10-25 10:57:49

Creating Door Product and Device

1. Log in to the [IoT Hub console](#) and click **Products** on the left sidebar.
2. On the product list page, click **Create Product**.
3. Create a door product (Door), select the authentication method, enter the product description, and click **Confirm**.

Create Product

Region *

Guangzhou

Product Type *

General

Gateway

Product Name *

Supports Chinese characters, "-", letters, Number, underscores, "@", "(", ")", "/", "\", Space; Max 40 characters

Authentication Method *

Certificate

Key

CA Certificate *

Tencent Cloud certificate ▼

Data Format *

JSON

Custom

Description

Optional

Max 500 characters

Confirm

Cancel

Note :

- For more information on authentication method, see [Device Connection Preparations](#).

- When **Custom** is selected as the data format, the parsing may fail, resulting in garbled characters. In such cases, you are advised to create the product again and select **JSON** as the data format.

4. After successful creation, you can view the basic information of the product.

5. Click the door product (Door), select the **Devices** tab, and create a device (door1).

Note :

A device key will be returned after device creation under asymmetric encryption, and will be used for device communication. The key will not be stored in the IoT Hub backend. Keep it properly.

6. Click **Manage** to view the device information.

- Certificate authentication:

Basic Information	
Device Name	door1
Remarks	door-test
Online Status	Inactive Reset
Tag	No tag information. Add
Device Certificate	Download
Device Private Key	Download
Enabled/Disabled ⓘ	<input checked="" type="checkbox"/> Enabled
Firmware Version	Not reported

- Key authentication:

Basic Information

Device Name	door1
Remarks	None
Online Status	Inactive Reset
Tag	No tag information. Add
Enabled/Disabled ⓘ	<input checked="" type="checkbox"/> Enabled
Firmware Version	Not reported

Creating Air Conditioner Product and Device

1. Log in to the [IoT Hub console](#) and click **Products** on the left sidebar.
2. On the product list page, click **Create Product**.
3. Create an air conditioner product (AirConditioner), select the authentication method, enter the product description, and click **Confirm**.

Create Product

Region *

Guangzhou

Product Type *

General

Gateway

Product Name *

Supports Chinese characters, "-", letters, Number, underscores, "@", "(", ")", "/", "\", Space; Max 40 characters

Authentication Method *

Certificate

Key

CA Certificate *

Tencent Cloud certificate ▼

Data Format *

JSON

Custom

Description

Optional

Max 500 characters

Confirm

Cancel

4. After successful creation, you can view the basic information of the product.

5. Create a device (airConditioner1) on the **Devices** tab page.

6. Click **Manage** to view the device information.

Basic Information

Device Name

airConditioner1

Remarks

None

Online Status

Inactive [Reset](#)

Tag

No tag information. [Add](#)

Enabled/Disabled ⓘ

☒ Enabled

Firmware Version

Not reported

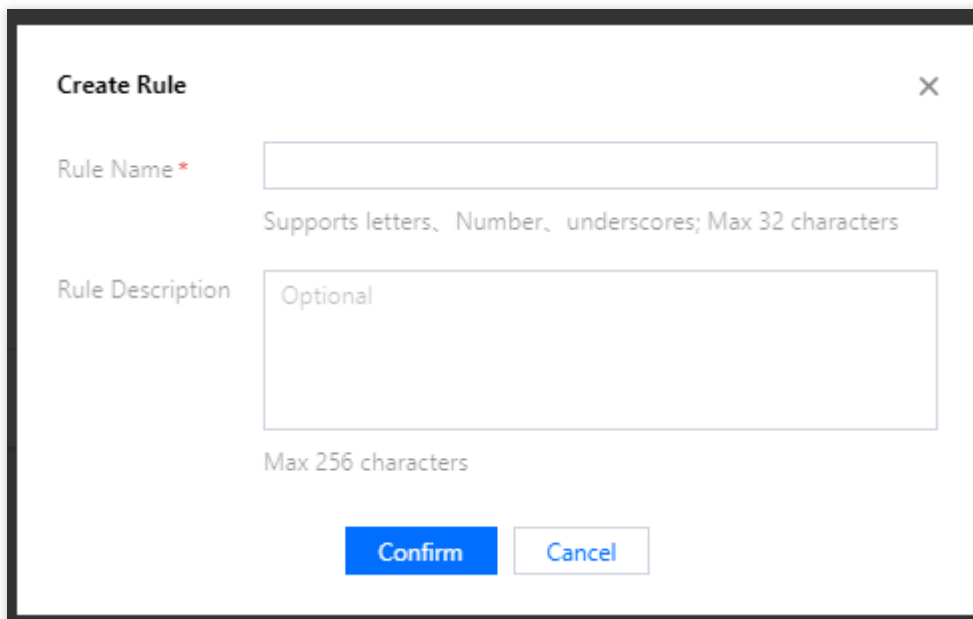
In the device information page, device certificate and device private key are used for MQTT over TLS asymmetric encryption. Symmetric key is used for symmetric encryption (for the differences between the two communication methods, see [Feature Components - Device Connection](#)).

Note :

The creation of the above resources can be done by the backend through RESTful APIs. For more information, please see [API Overview](#).

Creating Rule

1. Log in to the [IoT Hub console](#) and click **Rule Engine** on the left sidebar.
2. On the rule engine page, click **Create Rule**, enter the rule name, and click **Confirm**.
 - Rule Name: it can contain up to 32 letters, digits, and underscores (the name cannot be modified once confirmed).
 - Rule Description: 0-256 characters. This can be modified.



Create Rule ×

Rule Name *
Supports letters, Number, underscores; Max 32 characters

Rule Description
Optional
Max 256 characters

Confirm **Cancel**

3. After the rule is created successfully, you will be automatically redirected to the rule details page.

Basic Information

Rule Name

rule1

Status

Disabled

Rule Description

Filter Data ?

Field

Topic

\$(productid)/\$(devicename)/event

Condition

Current SQL

SELECT FROM '\$(productid)/\$(devicename)/event'

Action

Add Action

Forward Error Actions

Add Action

Then, you can write different forwarding rules.

Device-Side Operation Steps

Last updated : 2021-10-25 15:05:08

Downloading SDK

For the SDK download method, please see [SDK Download](#).

Compiling and running SDK for C demo

SDK for C demo:

- `samples/scenarized/door_mqtt_sample.c` is the MQTT-based logic code for the door device.
- `samples/scenarized/aircond_shadow_sample.c` is the MQTT-based logic code for the air conditioner device.

Below describes how to compile and run the device interconnection demo in a Linux environment with **key authentication** as an example:

1. Compile the SDK

Modify `CMakeLists.txt` to ensure that the following options exist:

```
set(BUILD_TYPE "release")
set(COMPILER_TOOLS "gcc")
set(PLATFORM "linux")
set(FEATURE_MQTT_COMM_ENABLED ON)
set(FEATURE_MQTT_DEVICE_SHADOW ON)
set(FEATURE_AUTH_MODE "KEY")
set(FEATURE_AUTH_WITH_NOTLS OFF)
set(FEATURE_DEBUG_DEV_INFO_USED OFF)
```

Run the following script for compilation:

```
./cmake_build.sh
```

The demo outputs `aircond_shadow_sample` and `door_mqtt_sample` are in the `output/release/bin` folder.

2. Enter the device information

Enter the information of the `airConditioner1` device created above in the JSON file `aircond_device_info.json`.

```
"auth_mode": "KEY",
"productId": "GYT9V6D4AF",
"deviceName": "airConditioner1",
"key_deviceinfo": {
  "deviceSecret": "vXeds12qazsGsMyf5SMfs60A6y"
}
```

Enter the information of the `door1` device in another JSON file `door_device_info.json`.

```
"auth_mode": "KEY",
"productId": "S3EUVRJLB",
"deviceName": "door1",
"key_deviceinfo": {
  "deviceSecret": "i92E3QMNmxi5hvIxUHjO8gTdg"
}
```

3. Run the `aircond_shadow_sample` demo

In the code of `aircond_shadow_sample`, `_register_subscribe_topics` implements the subscription to the `/[{productId}]/[{deviceName}]/control` topic and registers the corresponding callback handler. After receiving a message from this topic, the callback determines whether the message content is `"come_home"` or `"leave_home"` and instructs `airConditioner` to turn on or off accordingly.

`_simulate_room_temperature` simply simulates the changes in indoor temperature and energy consumption of `airConditioner`. You can also implement other custom logic.

Because the device interconnection scenario involves two demos running simultaneously, you can run the air conditioner demo in the current terminal console first, and you can see that the demo subscribes to the topic and then enters the loop waiting status. The initial status of the air conditioner is `close`:

```
./output/release/bin/aircond_shadow_sample -c ./device_info.json
INF|2019-09-16 23:25:17|device.c|iot_device_info_set(67): SDK_Ver: 3.1.0, Product
_ID: GYT9V6D4AF, Device_Name: airConditioner1
INF|2019-09-16 23:25:19|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with
id: Nh9Vc success
DBG|2019-09-16 23:25:19|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(138): t
opicName=$shadow/operation/result/GYT9V6D4AF/airConditioner1|packet_id=56171
DBG|2019-09-16 23:25:19|shadow_client.c|_shadow_event_handler(63): shadow subscri
be success, packet-id=56171
INF|2019-09-16 23:25:19|aircond_shadow_sample.c|event_handler(96): subscribe succ
ess, packet-id=56171
INF|2019-09-16 23:25:19|shadow_client.c|IOT_Shadow_Construct(172): Sync device da
ta successfully
INF|2019-09-16 23:25:19|aircond_shadow_sample.c|main(256): Cloud Device Construct
```

Success

```
DBG|2019-09-16 23:25:19|mqtt_client_subscribe.c|qcloud_iot_mqtt_subscribe(138): topicName=GYT9V6D4AF/airConditioner1/control|packet_id=56172
DBG|2019-09-16 23:25:19|shadow_client.c|_shadow_event_handler(63): shadow subscribe success, packet-id=56172
INF|2019-09-16 23:25:19|aircond_shadow_sample.c|event_handler(96): subscribe success, packet-id=56172
INF|2019-09-16 23:25:19|aircond_shadow_sample.c|main(291): airConditioner state: close
INF|2019-09-16 23:25:19|aircond_shadow_sample.c|main(292): currentTemperature: 32.000000, energyConsumption: 0.000000
```

4. Run the door_mqtt_sample demo to simulate a homecoming event

Open another terminal console and run the door demo. According to the program launch parameter `-t airConditioner1 -a come_home`, you can see that the demo sends a JSON message `{"action": "come_home", "targetDevice": "airConditioner1"}` to the `/productID/deviceName/event` topic, which notifies the target device `airConditioner1` of the homecoming event.

```
./output/release/bin/door_mqtt_sample -c ./output/release/bin/device_info.json -t airConditioner1 -a come_home
INF|2019-09-16 23:29:11|device.c|iot_device_info_set(67): SDK_Ver: 3.1.0, Product_ID: S3EUVBRJLB, Device_Name: door1
INF|2019-09-16 23:29:11|mqtt_client.c|IOT_MQTT_Construct(125): mqtt connect with id: d89Wh success
INF|2019-09-16 23:29:11|door_mqtt_sample.c|main(229): Cloud Device Construct Success
DBG|2019-09-16 23:29:11|mqtt_client_publish.c|qcloud_iot_mqtt_publish(329): publish topic seq=46683|topicName=S3EUVBRJLB/door1/event|payload={"action": "come_home", "targetDevice": "airConditioner1"}
INF|2019-09-16 23:29:11|door_mqtt_sample.c|main(246): Wait for publish ack
INF|2019-09-16 23:29:11|door_mqtt_sample.c|event_handler(81): publish success, packet-id=46683
```

5. Observe the message reception of the air conditioner and simulate a message response

Observe the printout of `aircond_shadow_sample`. You can see that the homecoming message sent by `door1` and forwarded by the cloud has been received, the `state` has changed to `open`, and the indoor temperature `currentTemperature` (adjusted to the configured default temperature) and the energy consumption `energyConsumption` have changed dynamically.

```
INF|2019-09-16 23:29:11|aircond_shadow_sample.c|main(291): airConditioner state: open
INF|2019-09-16 23:29:11|aircond_shadow_sample.c|main(292): currentTemperature: 32.000000, energyConsumption: 0.000000
```

```
2.000000, energyConsumption: 0.000000
INF|2019-09-16 23:29:12|aircond_shadow_sample.c|on_message_callback(140): Receive
Message With topicName:GYT9V6D4AF/airConditioner1/control, payload:{"action":"com
e_home","targetDevice":"airConditioner1"}
INF|2019-09-16 23:29:12|aircond_shadow_sample.c|main(291): airConditioner state:
open
INF|2019-09-16 23:29:12|aircond_shadow_sample.c|main(292): currentTemperature: 3
1.000000, energyConsumption: 1.000000
```

Configuring SDK for Android demo

Implement SDK for Android door demo

`Door.java` is the door device class. Please enter the **PRODUCT_ID**, **DEVICE_NAME**, **DEVICE_CERT_NAME**, and **DEVICE_KEY_NAME** obtained in the previous steps for product and device creation and place the device certificate and device private key files in the **assets** directory:

```
/**
 * Product ID
 */
private static final String PRODUCT_ID = "YOUR_PRODUCT_ID";
/**
 * Device name
 */
protected static final String DEVICE_NAME = "YOUR_DEVICE_NAME";
/**
 * Key
 */
private static final String SECRET_KEY = "YOUR_DEVICE_PSK";
/**
 * Device certificate name
 */
private static final String DEVICE_CERT_NAME = "YOUR_DEVICE_NAME_cert.crt";
/**
 * Device private key file name
 */
private static final String DEVICE_KEY_NAME = "YOUR_DEVICE_NAME_private.key";
```

1. Perform an emptiness check on the MQTT connection instance in `enterRoom()`. If it is empty, perform initialization and initiate `connect()`; otherwise, determine whether the connection is valid, and if so, publish the `event` topic.
2. As an example, the content of the message is assembled and published based on the action (`come_home` or `leave_home`) and `targetDeviceName` (name of the device to relay to) parameters you specified when

running the program. You can organize the message content and topic on your own to execute your own message publishing logic.

Implementing SDK for Android air conditioner demo

[Airconditioner.java](#) is the air conditioner device class. Just like in [Implementing SDK for Android door demo](#), you need to enter the information related to the product and device first.

1. Initialize the MQTT connection instance in the `Airconditioner` constructor and initiate `connect()`.
2. After the MQTT connection is successfully established, subscribe to the `control` topic.

Running demo

1. Click the **Run** icon in Android Studio to install the demo.
2. Switch the bottom tab to the device interconnection fragment and observe the log information in the demo and logcat. The following is the log information in logcat:
airConditioner1 was connected to IoT Hub and subscribed to the topic

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Start connecting to ssl://connect.iot.qcloud.com:8883
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: onSuccess!
com.qcloud.iot I/IoTEntryActivity: connected to ssl://connect.iot.qcloud.com:8883
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Starting subscribe topic: *****/airConditioner1/control
com.qcloud.iot I/IoTEntryActivity: onSubscribeCompleted, subscribe success
```

3. Click **Enter** to connect to IoT Hub and publish the `control` topic. The corresponding message is:

```
{"action": "come_home", "targetDevice": "airConditioner1"}
```

4. Observe the log information in the demo and logcat. The following is the log information in logcat:

- door1 was connected to IoT Hub

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Start connecting to ssl://connect.iot.qcloud.com:8883
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: onSuccess!
com.qcloud.iot I/IoTEntryActivity: connected to ssl://connect.iot.qcloud.com:8883
```

- door1 published a topic (come_home)

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Starting publish topic: *****/door1/event Message: {"action": "come_home", "targetDevice": "airConditioner1"}
```

- airConditioner1 received the topic forwarded by the rule engine

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Received topic: *****/airConditioner1/control, message: {"action": "come_home", "targetDevice": "airConditioner1"}
com.qcloud.iot D/IoTEntryActivity: receive command: open airconditioner, count: 1
```

5. Click **Leave** to publish the `control` topic. The corresponding message is:

```
{"action": "leave_home", "targetDevice": "airConditioner1"}
```

6. Observe the log information in the demo and logcat. The following is the log information in logcat:

- door1 published a topic (leave_home)

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Starting publish topic: *****/door1/event Message: {"action": "leave_home", "targetDevice": "airConditioner1"}
```

- airConditioner1 received the topic forwarded by the rule engine

```
com.qcloud.iot I/com.qcloud.iot.mqtt.TXMqttConnection: Received topic: *****/airConditioner1/control, message: {"action": "leave_home", "targetDevice": "airConditioner1"}
com.qcloud.iot D/IoTEntryActivity: receive command: close airconditioner, count: 2
```

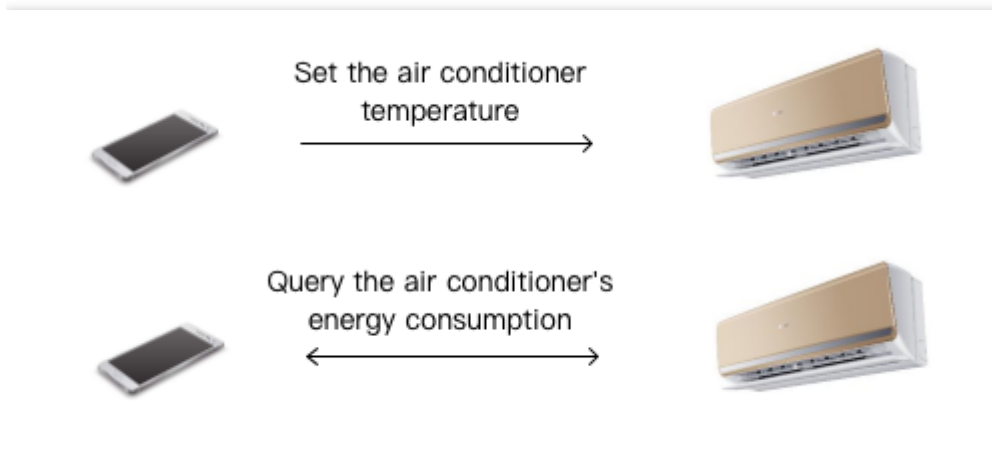
Scenario 2: Device Status Reporting and Setting

Scenario Overview

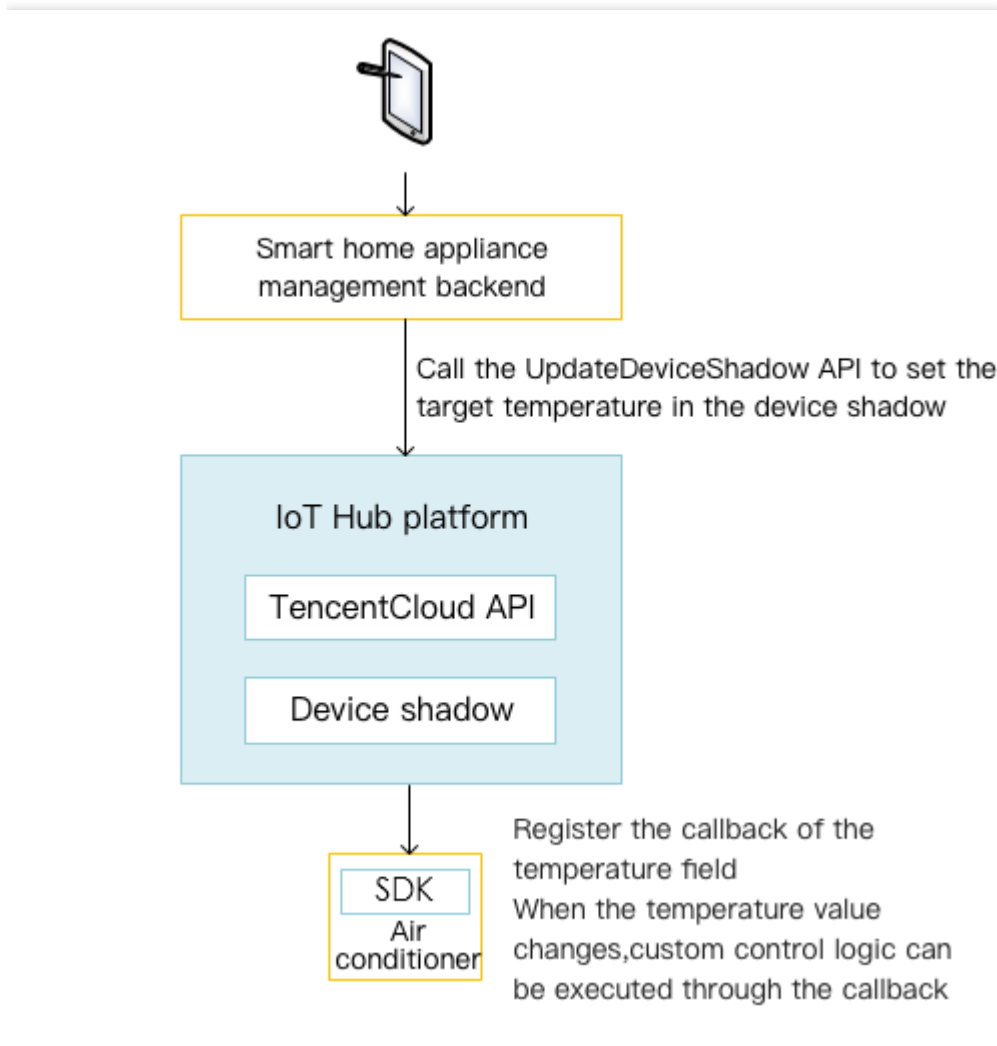
Last updated : 2021-10-25 10:43:58

Operation Scenario

If you need to set a device target temperature and report device status information in a smart home scenario (this is not a real product but only used to demonstrate IoT Hub's capabilities), you can follow the steps in this document.



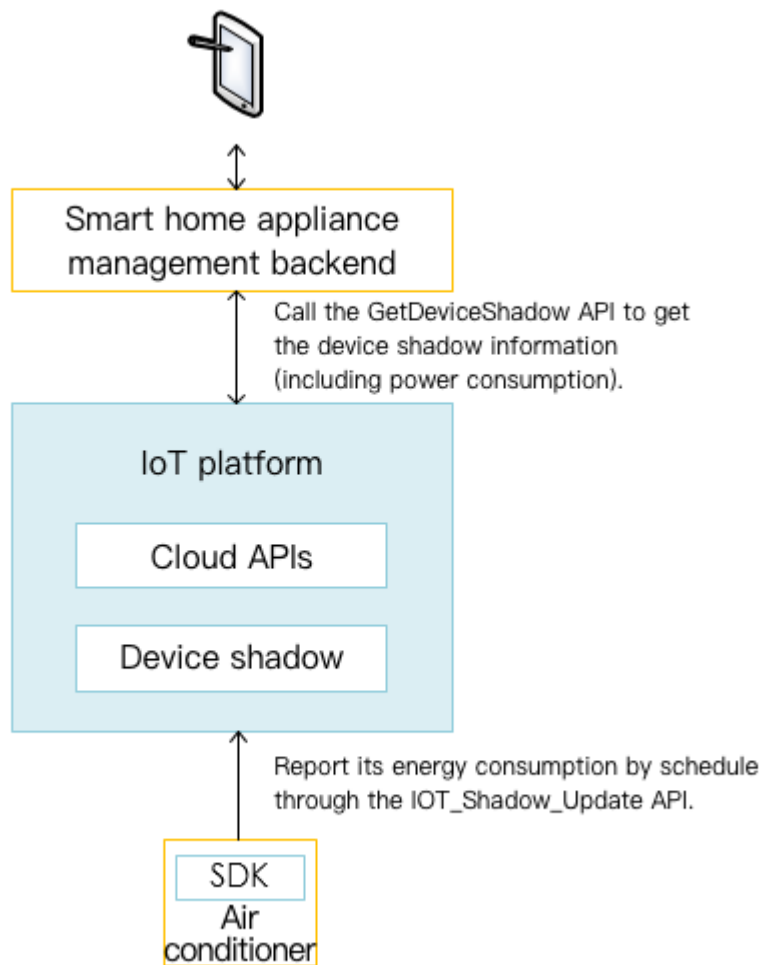
Setting Device Target Temperature



The management backend uses the cloud APIs provided by IoT Hub to update device shadow's configuration and registration parameters, and associate the corresponding callback function to update the configuration locally.

For the implementation of relevant TencentCloud APIs for device shadow, please download [iotcloud_RestAPI_python.zip](#). You need to configure your profile according to the RESTful API description. You can customize the features by modifying the parameters in `airConditionerCtrl.py` in the `RestAPI` folder.

Reporting Device Status Information



The device reports its own status data to the device shadow, and the home appliance management backend directly gets data from the device shadow through the RESTful API.

Device Status Reporting

Last updated : 2021-10-25 15:11:02

Directions for SDK for C

Program implementation

As an example, the energy consumption status is reported to the device shadow through the call of

`IOT_Shadow_Update` by the following function in the SDK code

`sample/scenarized/aircond_shadow_sample_v2.c`. Then, the corresponding callback function is

registered to handle the response of the device shadow. You can customize the reported attributes here.

```
_do_report_energy_consumption(...)
...
IOT_Shadow_Update(...)
```

Program compilation and execution

1. Run `./aircond_shadow_sample_v2`. Please note that if MQTT asymmetric encryption is used, the root certificate, device certificate, and device key files should be placed in the parent directory of `../../certs`.
2. Call the relevant RESTful API to get the status data of the shadow. For detailed directions, please see "Querying and getting device information". Observe the output log of the demo:

```
get_device_shadow rsp:
{"state":{"reported":{"energyConsumption":0}}, "metadata":{"reported":{"energyConsumption":{"timestamp":1506577440577}}}, "timestamp":1506577478410, "version":10, "message":"","codeDesc":"Success", "code":0}
```

3. Run `./door_mqtt_sample come_home/leave_home airConditioner1`. `door1` will communicate with `airConditioner1`, and then `airConditioner1` will be instructed to turn on through the rule engine. The reported changes in energy consumption and indoor temperature can be observed in the log, and the shadow data is obtained again through the RESTful API (as detailed in step 2):

```
INF[2018-01-08 16:14:49: actuate callback jsonString=10], "desired":{"temperatureDesire":10}, "reported":{"energyConsumption":1.0}, "timestamp":1515399267567, "version":161, "result":0, "timestamp":1515399289
, "type":"get"}
|dataLen=2
INF[2018-01-08 16:14:49: modify desire temperature to: 10.000000
INF[2018-01-08 16:14:49: Method=GET|Ack=ACK_ACCEPTED
INF[2018-01-08 16:14:49: received jsonString={"clientId":"0YPTD4UD8T-0", "payload":{"metadata":{"delta":{"temperatureDesire":{"timestamp":1515399267567}}, "desired":{"temperatureDesire":{"timestamp":1515399267567}}, "reported":{"energyConsumption":{"timestamp":1515398894625}}}, "state":{"delta":{"temperatureDesire":10}, "desired":{"temperatureDesire":10}, "reported":{"energyConsumption":1.0}, "timestamp":1515399267567, "version":161, "result":0, "timestamp":1515399289, "type":"get"}
```

It can be seen that after `airConditioner1` is turned on, the air conditioner energy consumption is dynamically reported to the shadow, and the data can be successfully queried and obtained through the RESTful API.

Directions for SDK for Android

Program implementation

Please see the instructions in [Directions for SDK for Android - Program implementation](#).

Program compilation and execution

Please see the instructions in [Directions for SDK for Android - Program compilation and execution](#).

Querying and getting device information

Call the RESTful API GetDeviceShadow to get the status data of the shadow, which is used by the application to display the device's energy consumption.

The RESTful API request parameter is: `deviceName=airConditioner1,`
`productName=AirConditioner` .

Device Temperature Setting

Last updated : 2021-10-26 15:16:57

Directions for SDK for C

Program implementation

1. The device shadow uses the code logic of `sample/scenarized/aircond_shadow_sample_v2.c` . It adds the following logic to `sample/scenarized/aircond_shadow_sample.c` :
2. As an example, the SDK internally calls `IOT_Shadow_Register_Property` to bind the shadow's configuration class attribute and callback function. When the shadow has a configuration change of this attribute, the underlying layer of the SDK will perform the corresponding callback. The `temperatureDesire` field in the shadow is registered here, which means that when the application sets the target temperature for the device shadow, the local configuration can be corrected by the callback function to adjust the desired temperature. You can also implement custom configuration-based attribute listening and callback binding.

```
rc = _register_config_shadow_property();
```

Program compilation and execution

1. Run `make` in the root directory of the SDK, compile, and get the `aircond_shadow_sample_v2` executable program.
2. Run `./aircond_shadow_sample_v2` in the `./output/release/bin` directory. Please note that if MQTT asymmetric encryption is used, the root certificate, device certificate, and device key files should be placed in the parent directory of `./../certs` .
3. Run `./door_mqtt_sample come_home airConditioner1` in the `./output/release/bin` directory to turn on `airConditioner` .

```
INF|2018-01-11 20:52:50|aircond_shadow_sample_v2.c|main(377): Cloud Device Construct Success
INF|2018-01-11 20:52:50|aircond_shadow_sample_v2.c|main(389): Cloud Device Register Delta Success
```

4. Call the RESTful API to simulate the home appliance management backend and publish the target temperature configuration. For detailed directions, please see "Publishing target temperature configuration" and observe the output log of the demo:

In the output log, it can be seen that the `on_temperature_actuate_callback` function has been called, indicating that the `delta` topic sent by the shadow has been received, and the operation `modify desire temperature to: 10.000000` has been performed for updating the locally set temperature.

```

INF|2018-01-11 21:04:31|aircond_shadow_sample_v2.c|on_temperature_actuate_callback(181): actuate callback jsonString=10,{"desired":{"temperatureDesire":10},"reported":{"energyConsumption":0.0},"timestamp":1515675847609,"version":5},"result":0,"timestamp":1515675871,"type":"get"}|dataLen=2
INF|2018-01-11 21:04:31|aircond_shadow_sample_v2.c|on_temperature_actuate_callback(184): modify desire temperature to: 10.000000
INF|2018-01-11 21:04:31|aircond_shadow_sample_v2.c|on_request_handler(123): Method=GET|Ack=ACK_ACCEPTED
INF|2018-01-11 21:04:31|aircond_shadow_sample_v2.c|on_request_handler(124): received jsonString={"clientToken":"EJSKHKIS1M-0","payload":{"metadata":{"delta":{"temperatureDesire":{"timestamp":1515675847609},"desired":{"temperatureDesire":{"timestamp":1515675847609},"reported":{"energyConsumption":{"timestamp":1515674881485}}},"state":{"delta":{"temperatureDesire":10},"desired":{"temperatureDesire":10},"reported":{"energyConsumption":0.0},"timestamp":1515675847609,"version":5},"result":0,"timestamp":1515675871,"type":"get"}

```

In the above output log of `airConditioner1`, it can be seen that the configuration operation has taken effect and `airConditioner` has adjusted the locally set temperature.

Directions for SDK for Android

Program implementation

`ShadowSample.java` is the device shadow class with the following main features:

1. Establish a shadow connection: `connect()`, which internally calls the `connect()` API of `TXShadowConnect`.
2. Close the shadow connection: `closeConnect()`, which internally calls the `disconnect()` API of `TXShadowConnection`.
3. Register the device attribute: `registerProperty()`, which internally calls the `registerProperty()` API of `TXShadowConnection`.
4. Get the device shadow: `getDeviceShadow()`, which internally calls the `get()` API of `TXShadowConnection`.
5. Regularly update the device shadow: `loop()`, which internally calls the `update()` API of `TXShadowConnection`.

Program compilation and execution

Before running the application, please enter the **PRODUCT_ID**, **DEVICE_NAME**, **DEVICE_CERT_NAME**, and **DEVICE_KEY_NAME** obtained in the previous steps for product and device creation and place the device certificate and device private key files in the **assets** directory:

```

/**
 * Product ID

```

```
*/
private static final String PRODUCT_ID = "YOUR_PRODUCT_ID";
/**
 * Device name
 */
protected static final String DEVICE_NAME = "YOUR_DEVICE_NAME";
/**
 * Key
 */
private static final String SECRET_KEY = "YOUR_DEVICE_PSK";
/**
 * Device certificate name
 */
private static final String DEVICE_CERT_NAME = "YOUR_DEVICE_NAME_cert.crt";
/**
 * Device private key file name
 */
private static final String DEVICE_KEY_NAME = "YOUR_DEVICE_NAME_private.key";
```

1. After entering the device information, click the **Run** icon in Android Studio to install and run the demo.
2. Switch the bottom tab to the **device shadow** fragment to use the features of the shadow.
3. Each feature has a corresponding operation button. Click a button and observe the log output of the demo and logcat.
4. For more information on the operations of the RESTful APIs, please see [Publishing target temperature configuration](#) or [Querying and getting device information](#).

Publishing target temperature configuration

Call the RESTful API UpdateDeviceShadow to simulate the home appliance management backend and publish the target temperature configuration.

The RESTful API request parameter is: `deviceName=airConditioner1, state={"desired" : {"temperatureDesire": 10}}, productName=AirConditioner` , which adjusts the control temperature to 10°C.

MQTT.fx Connection Guide

Last updated : 2021-10-26 15:48:18

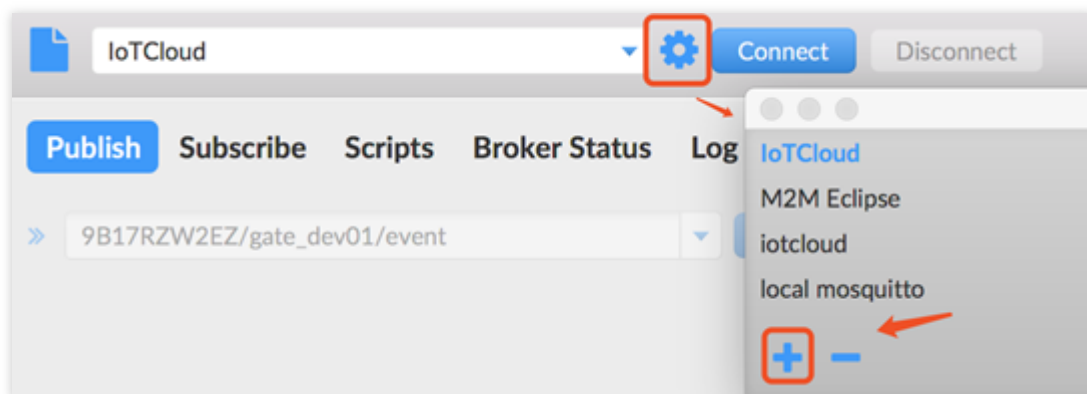
Overview

MQTT.fx is a mainstream MQTT desktop client. Compatible with Windows, macOS, and Linux, it can quickly verify whether it is possible to connect to IoT Hub and publish or subscribe to messages. For more information on the MQTT protocol, please see [MQTT Introduction](#). This document describes how MQTT.fx can interact with IoT Hub with MQTT.fx 1.7.0 for macOS as an example.

Directions

Connecting to IoT Hub

1. Download an appropriate version of MQTT.fx client on the [MQTT.fx download page](#) and install it.
2. Open the MQTT.fx client program and click the **Settings** icon.
3. Click + to create a profile.



4. Enter the **Connection Profile** and **General** information.

Profile Name

Profile Type **MQTT Broker**

MQTT Broker Profile Settings

Broker Address

Broker Port

Client ID **Generate**

General User Credentials SSL/TLS Proxy LWT

Connection Timeout

Keep Alive Interval

Clean Session ☒

Auto Reconnect ☒

Max Inflight

MQTT Version ☒ Use Default

Clear Publish History

Clear Subscription History

Parameter description

Parameter	Description
Profile Name	Name of the profile
Broker Address	MQTT server connection address. For more information, see Device Connection Regions . `PRODUCT_ID` in the domain is a variable parameter, and you should replace it with the product ID automatically generated when you create the product, such as `9****ZW2EZ.iotcloud.tencentdevices.com`.
Broker Port	MQTT server connection port. For certificate authentication, enter `8883`. For key authentication, enter `1883`.
Client ID	MQTT protocol field. Enter product ID + device name according to IoT Hub's requirement, such as `9****ZW2EZgate_dev01`, where `9****ZW2EZ` is the product ID, and `gate_dev01` is the device name.
Connection Timeout	Connection timeout period in seconds

Parameter	Description
Keep Alive Interval	Heartbeat interval in seconds
Auto Reconnect	Automatic reconnection after network disconnection

5. Enter the **User Credentials** information.

- **User Name** : MQTT protocol field. Enter product ID + device name + SDKAppID + connid according to IoT Hub's requirement, such as `9*****ZW2EZgate_dev01;12010126;12345` (the `ProductID` can be viewed on the product list or product details page after the product is created). It is sufficient to replace only the product ID and device name in the example. As the last two parameters are automatically generated by the connection SDK of IoT Hub, fixed test values are entered here.
- **Password** : the password is required.
 - **Certificate authentication**: as MQTT.fx sets the password flag to `true` by default, you need to enter a random non-empty string as the password; otherwise, you will not be able to connect to IoT Hub's backend. When actually accessing IoT Hub's backend, the authentication is based on the certificate, and the random password entered here will not be used as the access credential.
 - **Key authentication**: you can access the corresponding device list in the IoT Hub console to view and get the password (on the page as described in the key authentication steps below). You can also manually generate a password as instructed [MQTT-Based Device Connection over TCP](#).

The screenshot shows the 'MQTT Broker Profile Settings' window. The 'User Credentials' tab is active. The 'User Name' and 'Password' fields are highlighted with red rectangles. The 'Password' field is masked with dots. Other fields visible include 'Profile Name', 'Profile Type' (MQTT Broker), 'Broker Address', 'Broker Port' (8883), and 'Client ID' with a 'Generate' button.

6. (Optional) certificate verification: select **Enable SSL/TLS**, check **Self signed certificates**, and upload related files.

Profile Name

Profile Type MQTT Broker

MQTT Broker Profile Settings

Broker Address

Broker Port

Client ID Generate

General **User Credentials** **SSL/TLS** **Proxy** **LWT**

Enable SSL/TLS ☒ Protocol TLSv1.2

☐ CA signed server certificate
☐ CA certificate file
☐ CA certificate keystore
☒ **Self signed certificates**

CA File ...
 Client Certificate File ...
 Client Key File ...
 Client Key Password
 PEM Formatted ☒

☐ Self signed certificates in keystores

File description

File	Description
CA File	Root certificate. Click the ca.crt link to download the file.
Client Certificate File	Client certificate file, i.e., the device certificate which can be downloaded if the device is created in a certificate-authenticated product. For more information, please see Device Connection Preparations .
Client Key File	Client key file, i.e., the device key which can be downloaded if the device is created in a certificate-authenticated product. For more information, please see Device Connection Preparations .
PEM Formatted	The IoT Hub root certificate, device certificate, and device key are all generated by OpenSSL, and they are all in PEM format. MQTT.fx is a Java client, so it does not recognize PEM certificates. You need to select this option to enable the client to automatically convert the certificates into Java-recognized JKS format.

7. (Optional) Key authentication:

The screenshot shows the 'Edit Connection Profiles' dialog box. On the left, a list of profiles includes 'Alot', 'IoTCloud', 'M2M Eclipse', and 'local mosquitto'. The main area is titled 'MQTT Broker Profile Settings' and contains fields for 'Profile Name', 'Profile Type' (set to 'MQTT Broker'), 'Broker Address', 'Broker Port' (set to '1883'), and 'Client ID' with a 'Generate' button. Below these is a tabbed interface with 'General', 'User Credentials', 'SSL/TLS', 'Proxy', and 'LWT'. The 'User Credentials' tab is selected, showing 'User Name' and 'Password' fields. A red rectangle highlights these two fields. At the bottom are 'Revert', 'Cancel', 'OK', and 'Apply' buttons.

You can go to the console to get the username and password of the corresponding device:

← dev1

Chinese mainland IoT Hub Documentation

Device Information Permissions Online Debugging Device Shadow Device Simulator

Basic Information Edit

Device Name dev1

Remarks None

Online Status Inactive [Reset](#)

Tag label01:01 [Add](#)

Enabled/Disabled [i](#) ☒ Enabled

Firmware Version Not reported

Log Configuration Edit

Device Log Disabled

Log Level None

Device Key [i](#)


Device Key *****

Client ID *****

MQTT Username *****

MQTT Password *****

- After completing the above steps, click **Apply** > **OK** to save. Then, select the name of the file just created in the profile box and click **Connect**.
- If the round icon in the top-right corner is green, the connection to IoT Hub is successful, and publishing and subscribing operations can be performed.

IoTCloud [Connect](#) [Disconnect](#) 

[Publish](#) [Subscribe](#) [Scripts](#) [Broker Status](#) [Log](#)

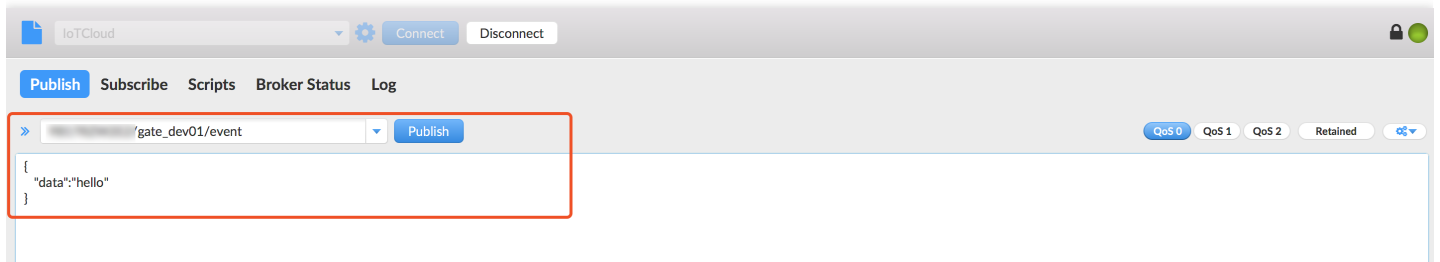
/gate_dev01/event [Publish](#)

QoS 0 QoS 1 QoS 2 Retained [v](#)

```
{
  "data": "hello"
}
```

Publishing a message

Select the **Publish** tab in the client, enter a topic name, select a QoS level, and click **Publish** to publish the message. The publishing result can be queried through [Cloud Log](#).



Subscribe to a topic

Select the **Subscribe** tab in the client, enter a topic name, select a QoS level, and click **Subscribe** to subscribe to the topic. The subscribing result can be queried through [Cloud Log](#).

