ALIBABA CLOUD

Alibaba Cloud

物联网平台

Quick Start

Document Version: 20211229

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	Danger: Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	Warning: Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	Notice: If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	Note: You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.Use IoT Platform

1.1. Overview

This article helps you understand the basic capabilities of IoT Platform. The following capabilities are provided: connect devices with IoT Platform, send messages from devices to IoT Platform, subscribe to device messages from IoT Platform, and send commands from IoT Platform to devices.

This article describes how to connect a device with IoT Platform and use the capabilities that are provided by IoT Platform. A street lamp is used in this example.

Before you begin

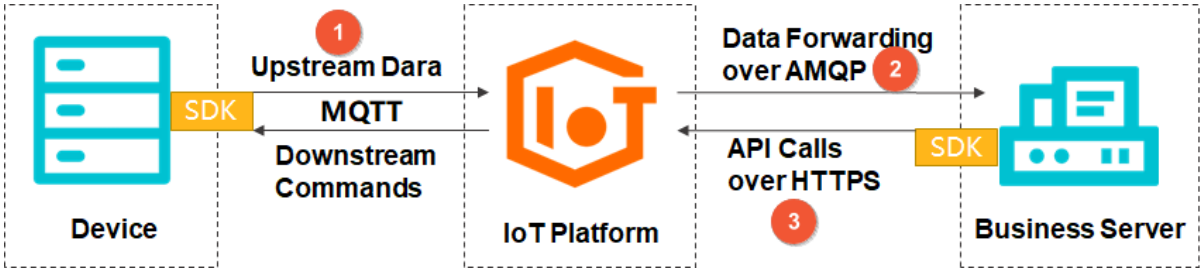
Before you connect a device to IoT Platform, you must perform the following operations:

- Activate IoT Platform.
- Prepare a development environment for the C programming language. In this article, the device is developed on the Linux operating system by using Link SDK for C.
- Prepare the Java development environment. In this article, the online debugging feature of IoT Platform is used to send downstream commands. AMQP SDK for Java is used to receive device messages.

  Required elements:

  - Operating system: Windows 10 (64-bit)
  - Java Development Kit (JDK): JDK 8
  - Integrated development environment (IDE): IntelliJ IDEA Community Edition

Message communication process



1. Submit device data to IoT Platform.
   i. Create a product and a device: Register your device in IoT Platform and obtain a device certificate (including ProductKey, DeviceName, and DeviceSecret). Burn the device certificate to the device. When the device connects to IoT Platform, the certificate is used for authentication.
   ii. Define a TSL model for a product: Define product features, including properties, services, and events. IoT Platform generates a Thing Specification Language (TSL) model based on the defined product features. The TSL model is used for the communication between the device and IoT Platform.
   iii. Establish a connection between a device and IoT Platform: Develop the device SDK and pass in the device certificate to connect the device with IoT Platform.
2. Subscribe to device messages from IoT Platform: Configure a server-side subscription to receive device messages. The message types include online or offline notifications, device lifecycle

changes, and upstream device messages.

3. Send commands from IoT Platform to devices: Use the online debugging feature of IoT Platform to send commands to the device.

1.2. Create a product and a device

When you use IoT Platform, you must first create a product, create a device that belongs to the product, and obtain the device certificate. The certificate information includes the ProductKey, DeviceName, and DeviceSecret.

Context

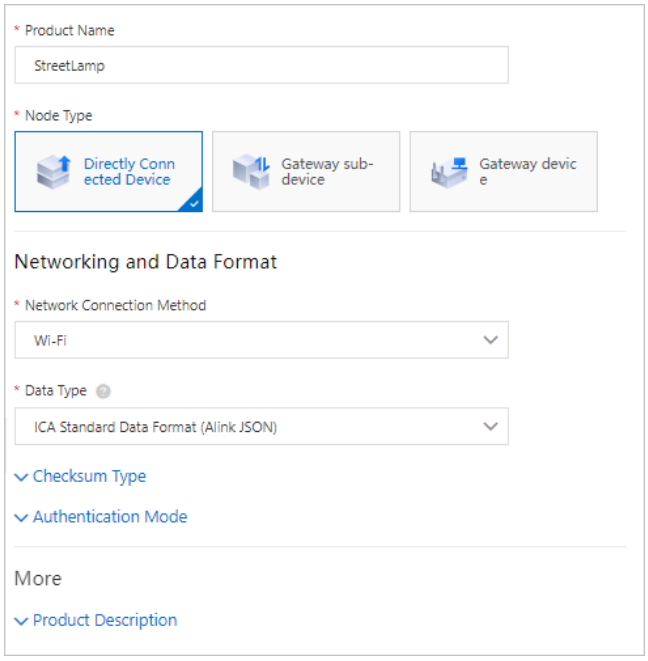
- A product indicates a collection of devices that have the same features. You can manage devices in batches based on products. For more information, see Define TSL models and Custom topics.
- Each actual device must correspond to an IoT Platform device. You must burn a device certificate (ProductKey, DeviceName, and DeviceSecret) that is issued by IoT Platform to a device. Then, use the certificate to authenticate the device when you connect the device to IoT Platform. For more information, see Obtain device certificates.

This article describes how to create a street lamp product, create a street lamp device, and then obtain a device certificate in the IoT Platform console.

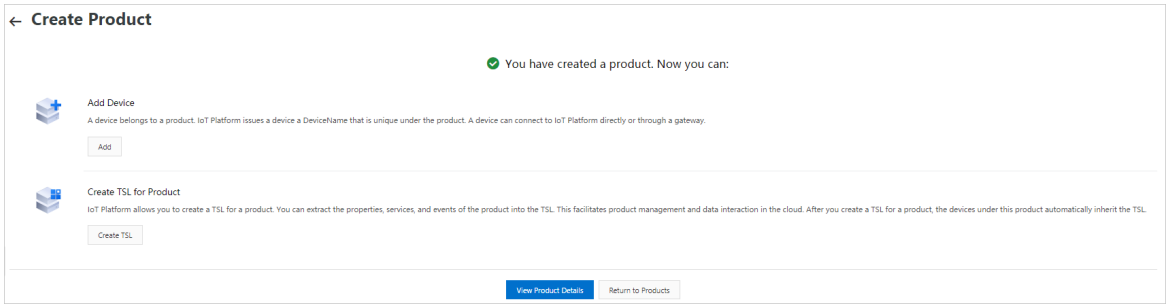
Procedure

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose **Devices > Products**.
3. On the **Create Product** page, set the parameters and click **OK**.

   In this example, set the **Product Name** parameter to StreetLamp. For other parameters, use the default values.

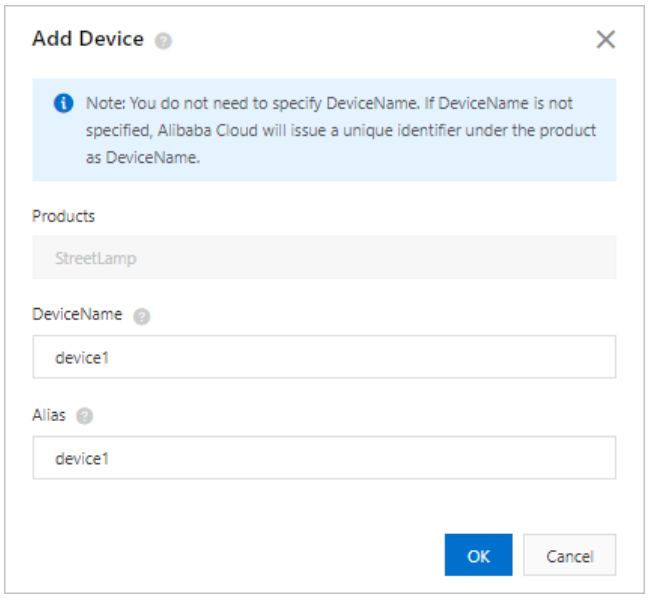


4. In the **Create Product** dialog box, click **Add**.



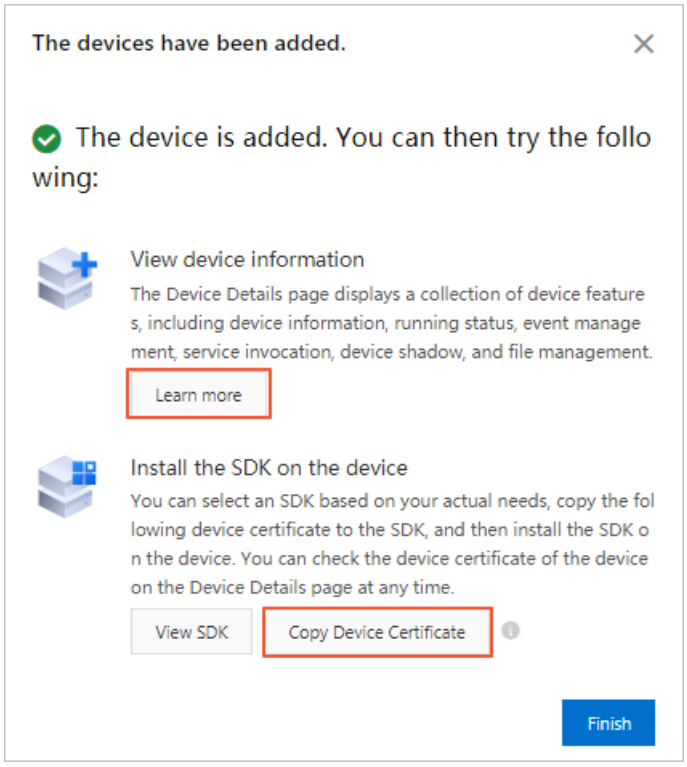
5. On the **Device List** tab, click **Add Device**. Set the DeviceName parameter to device1, set the Alias parameter to device1, and then click **OK**.

   For more information, see Create a device.



   In the **The devices have been added** dialog box, click **Copy Device Certificate** to obtain the device certificate.

   You can also click **Learn More**. On the **Device Details** page, click **View** next to **DeviceSecret** to obtain the device certificate.



A device certificate consists of a ProductKey, a DeviceName, and a DeviceSecret. A device certificate is the credential that a device uses to communicate with IoT Platform. We recommend that you copy and secure the device certificate. The device certificate is required when you connect the device to IoT Platform.

Parameter	Description
ProductKey	The ProductKey of the product to which the device belongs. The ProductKey is the GUID that is issued by IoT Platform to the product.
DeviceName	The DeviceName of the device, which is the unique identifier of the device within the product. A combination of the DeviceName and ProductKey is used as the device identifier. IoT Platform authenticates the device and communicates with the device based on the device identifier.
DeviceSecret	The DeviceSecret of the device, which is issued by IoT Platform for device authentication and encryption. The DeviceSecret must be used in combination with the DeviceName.

What's next

Define a TSL model for a product.

1.3. Define a TSL model for a product

IoT Platform allows you to define a Thing Specification Language (TSL) model for a product. You can abstract product features into a data model that consists of properties, services, and events. This facilitates cloud management and data interaction. After the product is created, you can define the TSL model. Devices under the product automatically inherit the TSL model.

Prerequisites

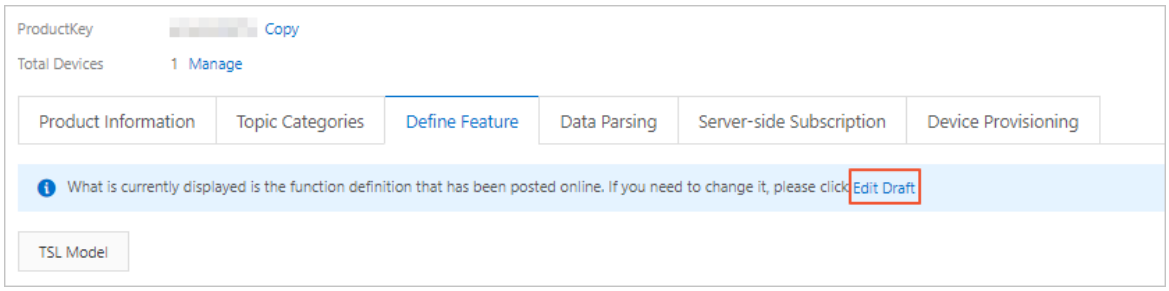
A product is created. For more information, see Create a product and a device.

Context

This article describes how to define the Current and Main Light Switch TSL properties for the product to which a street lamp belongs. After you define the TSL model, you can add street lamps to the product. The street lamps automatically inherit the defined TSL model. For more information about TSL models, see What is a TSL model?.

Procedure

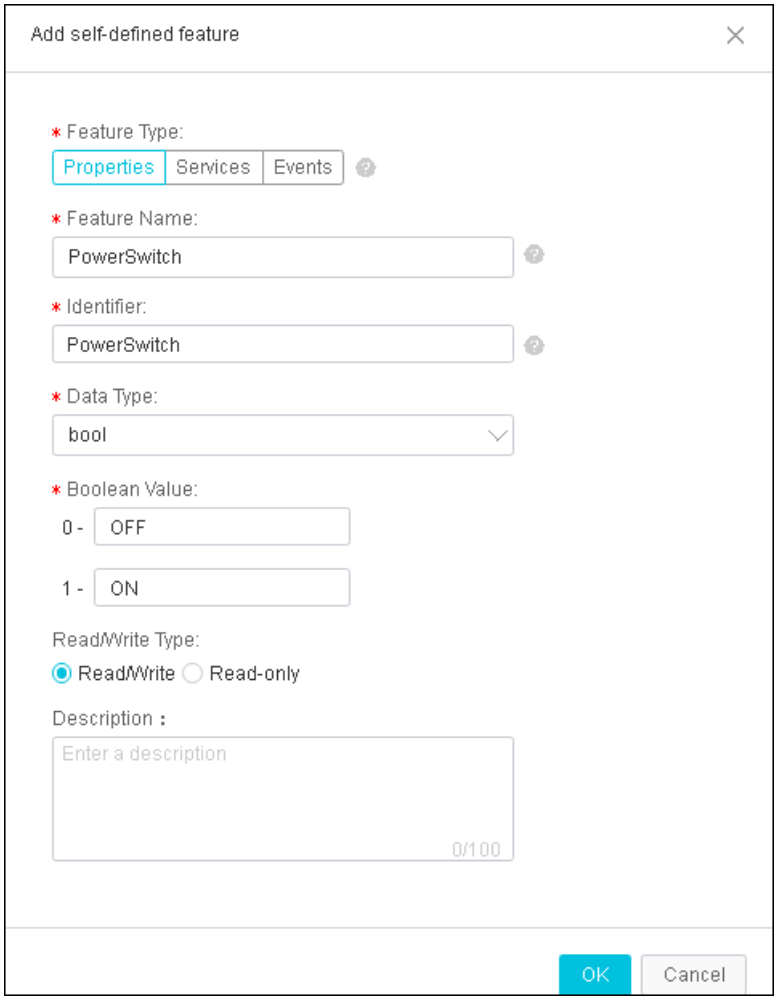
1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose **Devices > Products**. In the product list, find the product and click **View** in the Actions column.
3. On the Product Details page, click the **Define Feature** tab, and then click **Edit Draft**.



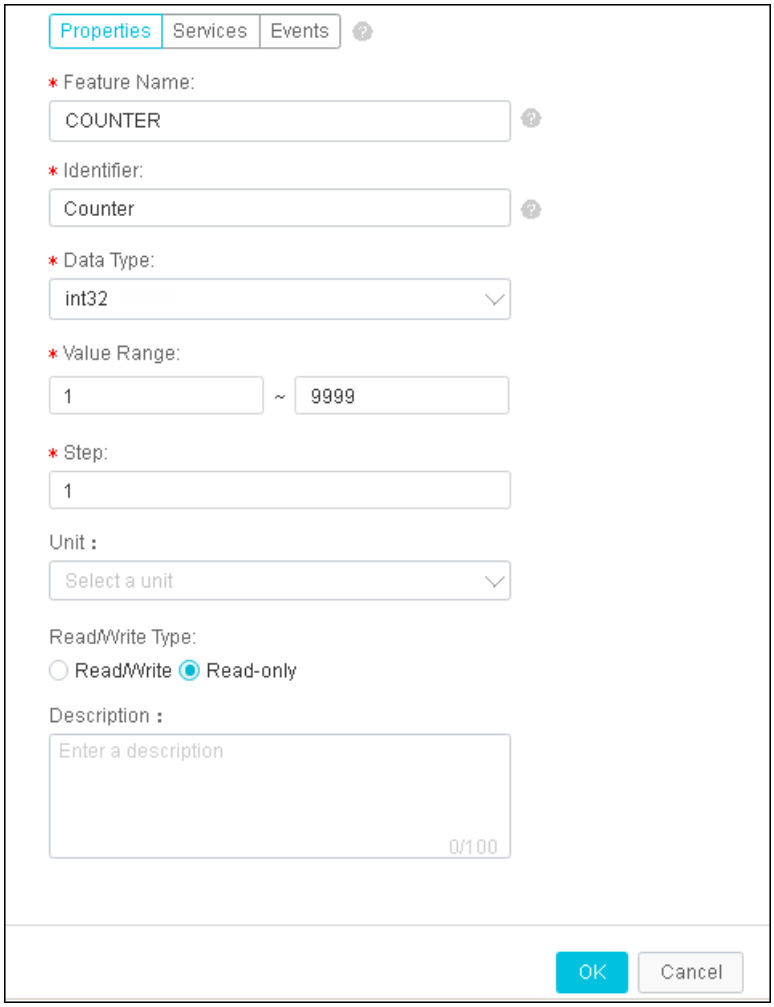
4. In the **Default Module** section of the **Define Feature** page, click **Add Self-defined Feature** to configure the TSL feature and then click **OK**.

   Configure the Main Light Switch and Current properties, as shown in the following figures.

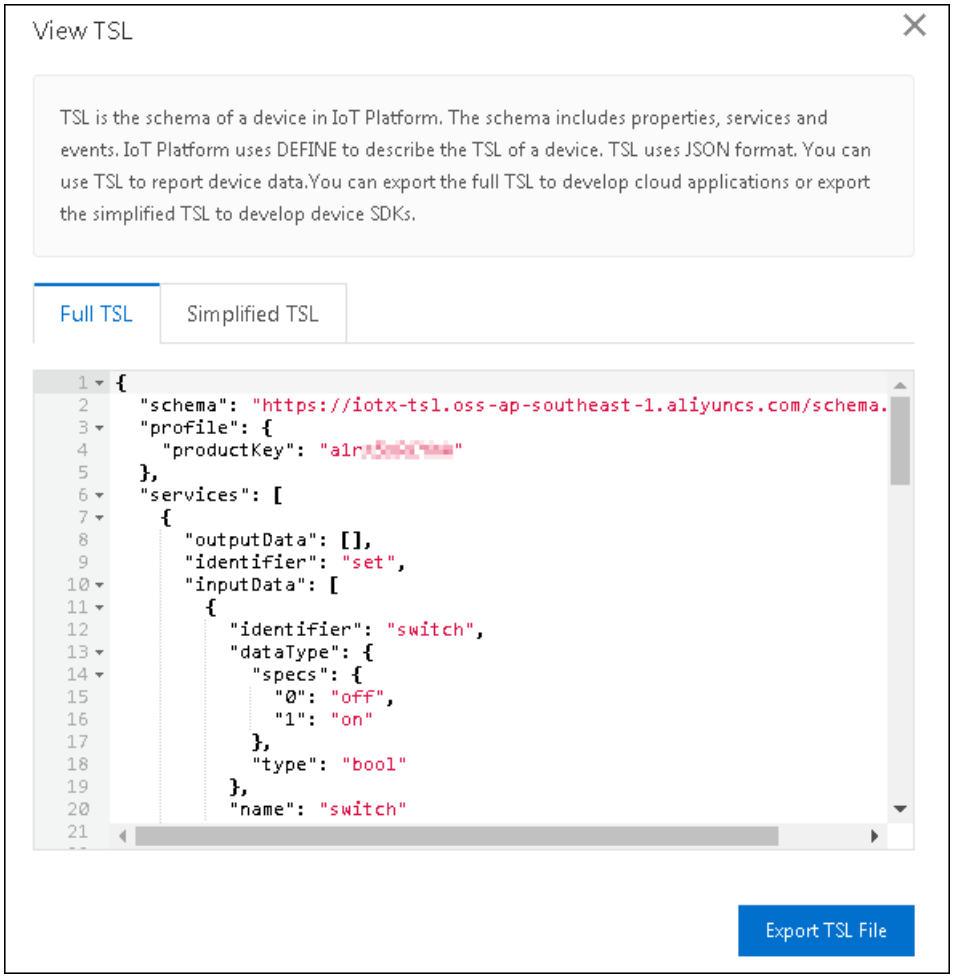
   - Main Light Switch property



- Current property



5. Optional. Click **TSL Model**. On the **Full TSL** tab, you can view the JSON file of the complete TSL model.



6. Release the TSL model.
   i. On the Edit Draft page, click **Release Online**. The **Release model online?** dialog box appears.
   ii. Optional. Click **+Add post notes**, and enter a version number and note.

Parameter	Description
Version Number	The version number of the TSL model. You can manage the TSL model based on the version number. The version number must be 1 to 16 characters in length, and can contain letters, digits, and periods (.).
Note	The description of the TSL model. The description can contain letters, digits, and special characters. It must be 1 to 100 characters in length.

   iii. If an online version is available, you must check the differences between the current version and the online version.

   Click **View Differences**. In the **View Differences** dialog box, you can view the differences. After you confirm your settings, click **Confirm**. In the **Release model online?** dialog box, the checkbox is automatically selected.

   iv. Click **OK** to release the TSL model.

What's next

Establish a connection between a device and IoT Platform

1.4. Establish a connection between a device and IoT Platform

Alibaba Cloud IoT Platform provides device SDKs that allow devices to connect with the platform. This article describes how to achieve communication between a device and IoT Platform by using the `data_model_basic_demo` sample code. A street lamp is developed in this example.

Context

- In this article, Link SDK for C is used on the Linux operating system. We recommend that you use 64-bit Ubuntu 16.04 to compile the SDK.
- The following software is required for SDK development and compilation:

  make (version 4.1 or later) and gcc (version 5.4.0 or later)

  Run the following command to install the software:

  ```
  sudo apt-get install -y build-essential make gcc
  ```

Procedure

1. For information about how to obtain Link SDK for C, see SDK Download.
2. Decompress Link SDK for C and modify the device certificate information in the SDK.

   IoT Platform provides the updated sample code based on this quick start. Click data_model_basic_demo.c to download the latest sample code. Then, go to the *\LinkSDK\demos* directory, and replace the *data_model_basic_demo.c* file with the downloaded file. You must modify the following parameters in the file:

   ```
   char *product_key       = "a2***";
   char *device_name       = "device1";
   char *device_secret     = "8c684ef***";
   ...
   ...
   char  *mqtt_host = "a2***.iot-as-mqtt.cn-shanghai.aliyuncs.com";
   ```

Parameter	Example	Description
url	a2***.iot-as-mqtt.cn-shanghai.aliyuncs.com	The MQTT endpoint. Format: `${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com`.
product_key	a2***	The device certificate information that is saved on premises after you add the device. You can also view the information on the Device Details page of the IoT Platform console.
device_name	device1	
device_secret	8c684ef***	

3. Log on to the Linux virtual machine and run the following command to install the required software:

```
sudo apt-get install -y build-essential make gcc
```

4. Upload the *LinkSDK* file that is modified in Step 2 to the development environment.
5. Go to the */LinkSDK* SDK root directory and run the `make` command to compile the sample code:

```
make clean
make
```

The generated `data-model-basic-demo` file is stored in the *./output* directory.

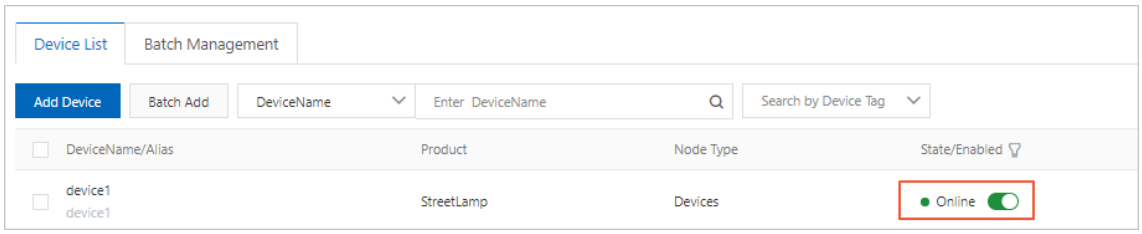
6. Run the sample code.

```
./output/data-model-basic-demo
```

The following figure shows a sample success response:

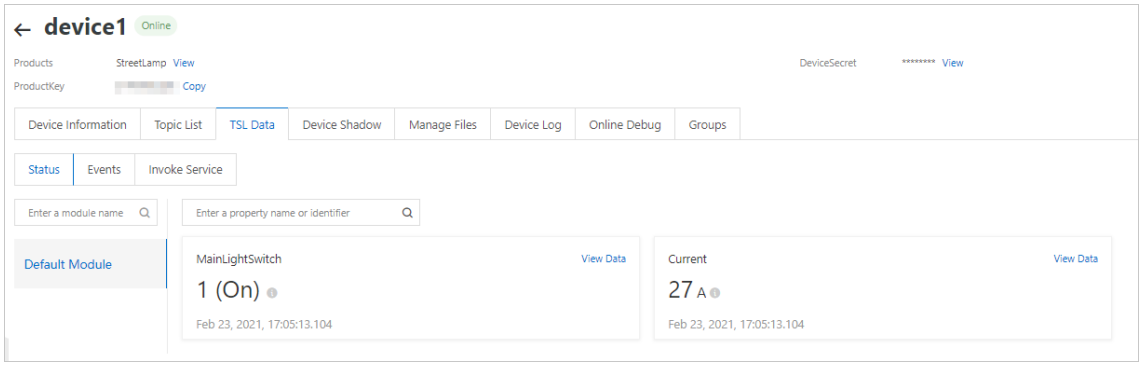
```
roo[redacted]:~/LinkSDK# ./output/data-model-basic-demo
[1637041632.700][LK-0313] MQTT user calls aiot_mqtt_connect api, connect
[1637041632.700][LK-032A] mqtt host: [redacted].iot-as-mqtt.cn-shanghai.aliyuncs.com
[1637041632.700][LK-0317] user name: device1&a[redacted]0
[1637041632.700][LK-0318] password: 86761[redacted]E21D
establish tcp connection with server(host='a[redacted]0.iot-as-mqtt.cn-shanghai.aliyuncs.com', port=[443])
success to establish tcp, fd=3
local port: 35604
[1637041632.711][LK-1000] establish mbedtls connection with server(host='a[redacted].iot-as-mqtt.cn-shanghai.aliyuncs.com', port=[443])
[1637041632.766][LK-1000] success to establish mbedtls connection, (cost 45252 bytes in total, max used 47988 bytes)
[1637041632.811][LK-0313] MQTT connect success in 108 ms
AIOT_MQTTEVT_CONNECT
[1637041632.811][LK-0309] pub: /sys/a[redacted]0/device1/thing/event/property/post

[LK-030A] > 7B 22 69 64 22 3A 22 31  22 2C 22 76 65 72 73 69 | {"id":"1","versi
[LK-030A] > 6F 6E 22 3A 22 31 2E 30  22 2C 22 70 61 72 61 6D | on":"1.0","param
[LK-030A] > 73 22 3A 7B 22 4C 69 67  68 74 43 75 72 72 65 6E | s":{"LightCurren
[LK-030A] > 74 22 3A 20 31 2E 35 7D  2C 22 73 79 73 22 3A 7B | t": 1.5},"sys":{
[LK-030A] > 22 61 63 6B 22 3A 31 7D  7D                      | "ack":1}}

[1637041632.833][LK-0309] pub: /sys/a[redacted]0/device1/thing/event/property/post_reply

[LK-030A] < 7B 22 63 6F 64 65 22 3A  32 30 30 2C 22 64 61 74 | {"code":200,"dat
[LK-030A] < 61 22 3A 7B 7D 2C 22 69  64 22 3A 22 31 22 2C 22 | a":{},"id":"1","
[LK-030A] < 6D 65 73 73 61 67 65 22  3A 22 73 75 63 63 65 73 | message":"succes
[LK-030A] < 73 22 2C 22 6D 65 74 68  6F 64 22 3A 22 74 68 69 | s","method":"thi
[LK-030A] < 6E 67 2E 65 76 65 6E 74  2E 70 72 6F 70 65 72 74 | ng.event.propert
[LK-030A] < 79 2E 70 6F 73 74 22 2C  22 76 65 72 73 69 6F 6E | y.post","version
[LK-030A] < 22 3A 22 31 2E 30 22 7D                          | ":"1.0"}

[1637041632.833][LK-0A08] DM recv generic reply
demo_dm_recv_handler, type = 0
msg_id = 1, code = 200, data = {}, message = success
```

7. View the online status and running status of the device.
   - In the left-side navigation pane, choose **Devices > Devices**. Then, find the device and view the online status. The value **Online** in the State column indicates that the device is connected to IoT Platform.



   - Click **View** in the Actions column to go to the **Device Details** page. Click the **TSL data** tab, and then click **Status** to view the running status of the device.

     In this example, the following operating current is submitted to IoT Platform by using the sample code in the *data_model_basic_demo.c* file.



What's next

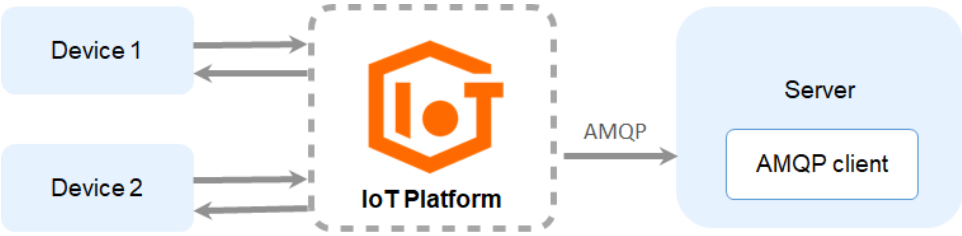
Subscribe to device messages from IoT Platform

1.5. Subscribe to device messages from IoT Platform

After a device is connected to IoT Platform, the device submits data to IoT Platform. Then, the data can be forwarded to your server by using the Advanced Message Queuing Protocol (AMQP). This article describes how to configure an AMQP server-side subscription. Your server can receive data from the street lamp by using an AMQP client.

Context

The following figure shows the process of obtaining device messages by configuring an AMQP server-side subscription.

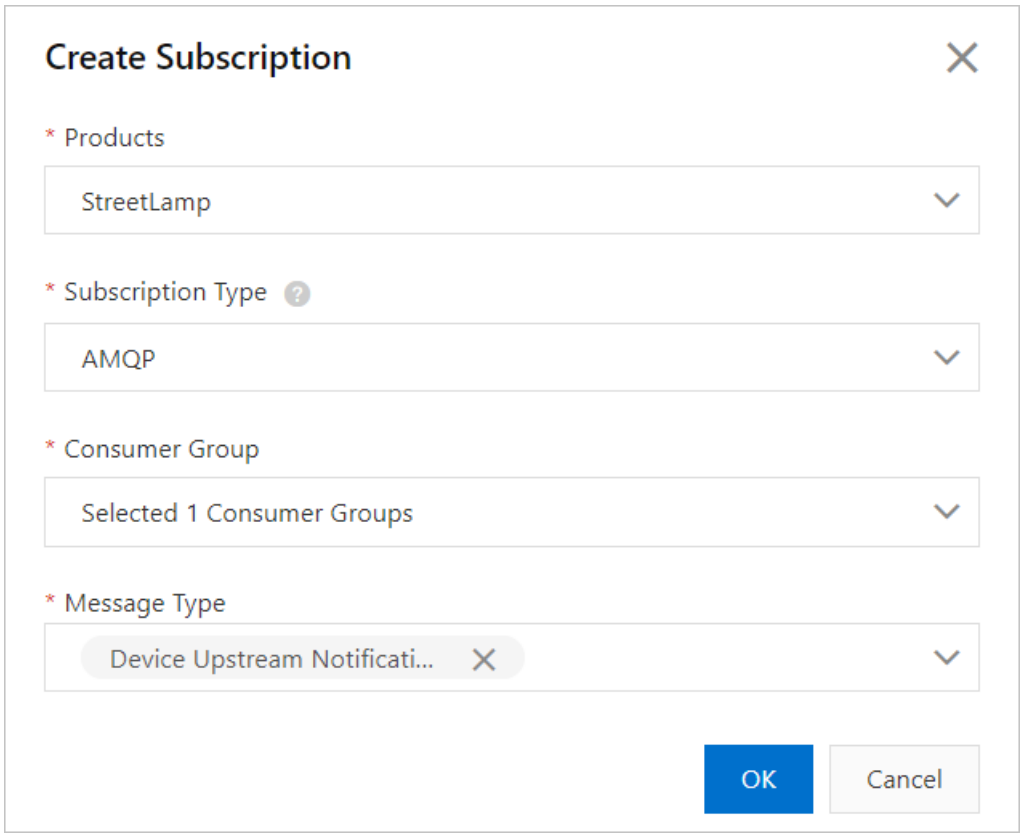


Procedure

1. Log on to the IoT Platform console.
2. Configure a consumer group to consume messages. Your server can obtain the messages by listening to the consumer group.
   i. In the left-side navigation pane, choose **Rules > Server-side Subscription** and click the **Consumer Groups** tab.
   ii. Click **Create Consumer Group**.
   iii. In the **Create Consumer Group** dialog box, set the consumer group name to StreetLampConsumerGroup and click **OK**.
3. Configure the server-side subscription for the product to which the street lamp device belongs. This allows your server to subscribe to various types of messages under the product.
   i. In the left-side navigation pane, choose **Rules > Server-side Subscription**.

ii. On the **Subscriptions** tab of the **Server-side Subscription** page, click **Create Subscription**.

iii. In the **Create Subscription** dialog box, set the parameters and click **OK**.



Parameter	Description
Products	Select StreetLamp.
Subscription Type	Select AMQP.
Consumer Group	Select StreetLampConsumerGroup that is created in the previous step.
Message Type	Select Device Upstream Notification.

4. Connect the AMQP client with IoT Platform.

   In this example, Java is used for development. We recommend that you use the Apache Qpid JMS client. To download the client and view the instructions, see Qpid JMS 0.57.0.

   In this example, the development environment consists of the following components:

   - Operating system: Windows 10 (64-bit)
   - Java Development Kit (JDK): JDK 8
   - Integrated development environment (IDE): IntelliJ IDEA Community Edition

   To connect the AMQP client to IoT Platform, perform the following steps:

   i. Download the demo package and decompress it.

   ii. Open IntelliJ IDEA and import the sample project *aiot-java-demo* in the demo package.

iii. In the *AmqpClient.java* file under the *src/main/java/com.aliyun.iotx.demo* directory, set the parameters to connect the JMS client with IoT Platform. The following table describes the parameters.

Parameter	Example	Description
accessKey	LTAI4GFGQvKuqHJhFa******	Log on to the IoT Platform console, move the pointer over the profile picture, and then click **AccessKey Management** to obtain the AccessKey ID and AccessKey secret. **Note** If you use a RAM user, you must attach the AliyunIOTFullAccess permission policy to the user. This policy allows the user to manage IoT Platform resources. Otherwise, the connection with IoT Platform fails. For more information about how to authorize a RAM user, see RAM user access.
accessSecret	iMS8ZhCDdfJbCMeA005sieKe******	
consumerGroupId	VWhGZ2QnP7kxWpeSSjt******	The ID of the consumer group that is created in Step 3. You can view the ID on the Public Instance page of the IoT Platform console. Choose **Rules > Server-side Subscription**, and then click the **Consumer Groups** tab. Find the consumer group and obtain the ID.
iotInstanceId	""	The ID of the instance. ■ If you use a public instance of the previous version, enter a null value. No instance ID exists.
clientId	12345	The ID of the client. We recommend that you use a unique identifier, such as the UUID, MAC address, or IP address of the client. The client ID must be 1 to 64 characters in length. Log on to the IoT Platform console. Choose **Rules > Server-side Subscription > Consumer Groups**, and click **View** next to the required consumer group. The **Consumer Group Details** page shows this parameter. This parameter allows you to identify clients.
connectionCount	4	The number of connections that are enabled on the AMQP client. Maximum value: 64. This parameter is used for the scale-out in real-time message pushing scenarios. On the **Consumer Group Details** page, each connected client is displayed in the format of `${clientId}+"-"+ a number`. The minimum number is 0.

Parameter	Example	Description
host	198426864******.iot-amqp.cn-shanghai.aliyuncs.com	The AMQP endpoint. Format: `${uid}.iot-amqp.${regionId}.aliyuncs.com`. ■ ${uid}: Replace this variable with the ID of your Alibaba Cloud account. You can log on to the IoT Platform console, and move the pointer over the profile picture to view the **account ID**. ■ ${YourRegionId}: Replace this variable with the ID of the region where your IoT Platform device resides. For more information about region IDs, see Regions and zones.

iv. After you run the sample code, the following log data is returned. The data indicates that the AMQP client is connected to IoT Platform and can receive messages.

> **Note** In this example, the `Thread.sleep(60 * 1000);` code snippet is added to end the program after the program starts and runs for 1 minute. You can set the running time based on your business requirements.

```
[main] INFO com.aliyun.iotx.demo.AmqpClient - amqp demo is started successfully, and will exit after 60s
[AmqpProvider :(2):[amqps://[redacted].aliyuncs.com:5671]] DEBUG org.apache.qpid.jms.provider.amqp.AmqpConsumer - Dispatching received message: JmsInboundMessageDispatch { sequenc
[JmsSession [ID:d8cd8ff1-5c3[redacted]y dispatcher] DEBUG org.apache.qpid.jms.provider.failover.FailoverProvider - Executing Failover Task: message acknowledge -> JmsInboundMessag
[AmqpProvider :(2):[amqps://[redacted].aliyuncs.com:5671]] DEBUG org.apache.qpid.jms.provider.amqp.AmqpConsumer - Delivered Ack of message: JmsInboundMessageDispatch { sequence = 1
[JmsSession [ID:d8cd8ff1-5c3[redacted]y dispatcher] DEBUG org.apache.qpid.jms.provider.failover.FailoverProvider - Executing Failover Task: message acknowledge -> JmsInboundMessag
[AmqpProvider :(2):[amqps://[redacted].aliyuncs.com:5671]] DEBUG org.apache.qpid.jms.provider.amqp.AmqpConsumer - Accepted Ack of message: JmsInboundMessageDispatch { sequence = 1
[pool-1-thread-1] INFO com.aliyun.iotx.demo.AmqpClient - receive message,
topic = /a1[redacted]/device1/thing/event/property/post,
messageId = 1380044478987006465,
content = {"deviceType":"CustomCategory","iotId":"nTT9[redacted]","requestId":"1617863381153","checkFailedData":{},"productKey":"a10[redacted]","gmtCreate":1617863218430,"deviceName":"device1","items":{"Light
```

For more information about how to configure a server-side subscription to receive device messages, see the following articles:

Related information

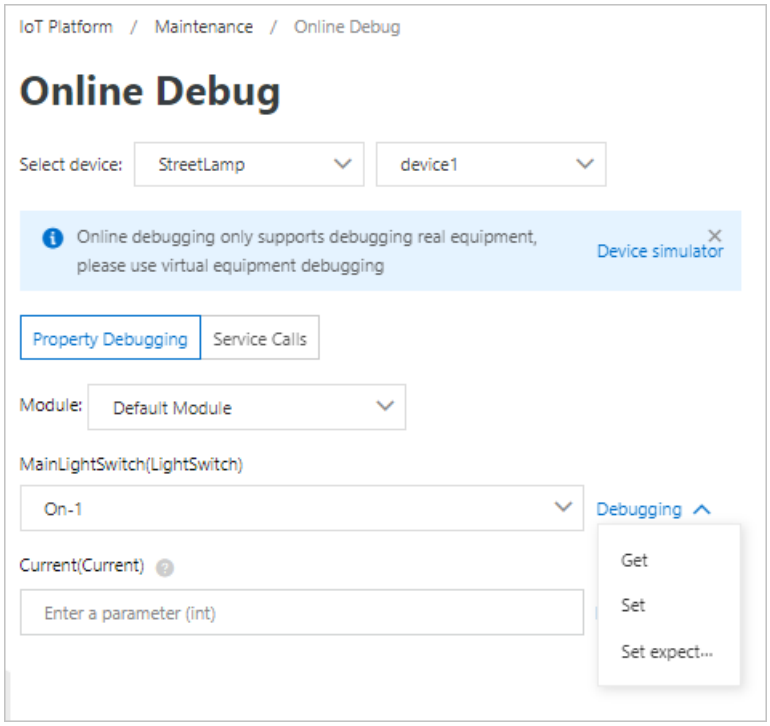
- Configure an AMQP server-side subscription
- Connect an AMQP client to IoT Platform

1.6. Send commands from IoT Platform to devices

After a device can submit data to IoT Platform, you can test downstream messaging by sending a command from IoT Platform to the device. This article describes how to set the properties of a street lamp device by using the online debugging feature in the IoT Platform console.

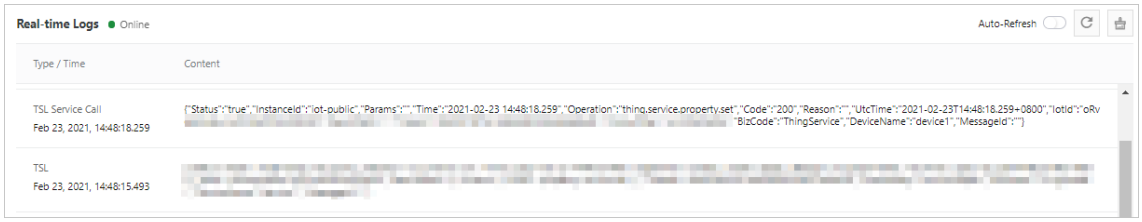
Procedure

1. Log on to the IoT Platform console.
2. In the left-side navigation pane, choose **Maintenance > Online Debug** .
3. On the **Online Debug** page, select the StreetLamp product and the device1 device.
4. On the **Property Debugging** tab, set the parameters to send a downstream command.



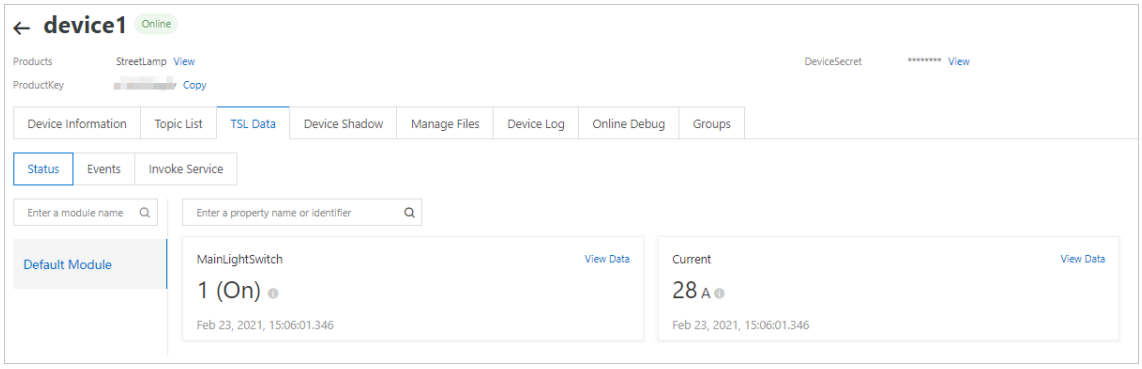
Scenario	Procedure
Module	Select **Default Module**.
Main Light Switch	Select **On-1**, and choose **Debugging** > **Set** on the right of the drop-down box.
Current	The Current property is read-only. To test downstream messaging, you do not need to set this property.

5. View the property status of the device.
   - In the Real-time Logs section of the **Online Debug** page, view the downstream and upstream data.



   - In the left-side navigation pane, choose **Devices** > **Devices**. Find the device, and then click **View** in the Actions column.

On the **Device Details** page, click the **TSL Data** tab. On this tab, click **Status** to view the running status of the device.



6. Optional. If you want to use your cloud-based application to set device properties, you can download the cloud SDK and call the specified API operation to send commands. For more information, see Cloud Developer Guide.

2.Connect a device to IoT Platform by using MQTT.fx

MQTT.fx is an Eclipse Paho-based Message Queuing Telemetry Transport (MQTT) client that is written in Java. It supports Windows, Mac, and Linux operating systems. It can be used to verify whether a device can connect to IoT Platform. MQTT.fx allows you to subscribe to and publish messages by using topics. This article describes how to connect a simulated device to IoT Platform over MQTT by using MQTT.fx on Windows.

Prerequisites

A product and a device are created in the IoT Platform console.The device certificate informationincluding ProductKey, DeviceName, and DeviceSecret is obtained. For more information, see the following articles:

- Create a product
- Create a device

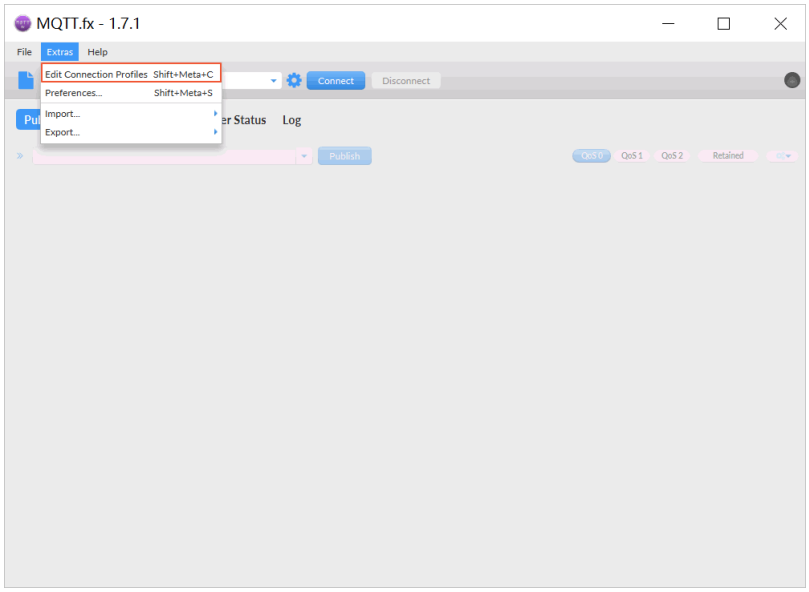
> **Notice** The MQTT.fx tool is used to simulate an online device and supports transmitting non-passthrough data. To transmit pass-through data, you can use an actual device or an SDK for testing.

Configure MQTT.fx

1. Download MQTT.fx v1.7.1 for Windows and install the MQTT.fx tool.

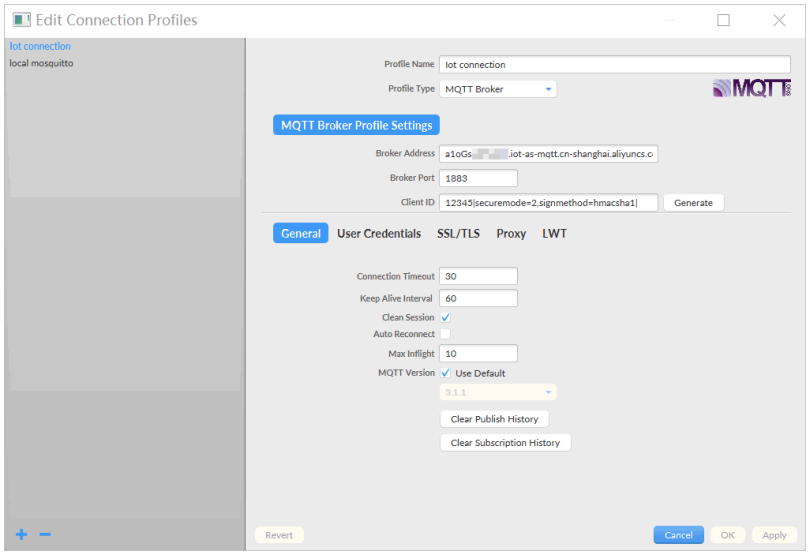
   For more information, visit the MQTT.fx official website.

2. Open the MQTT.fx tool, click **Extras** in the menu bar, and then select **Edit Connection Profiles**.



3. On the **Edit Connection Profiles** page, set the parameters.

   i. Edit the basic information.



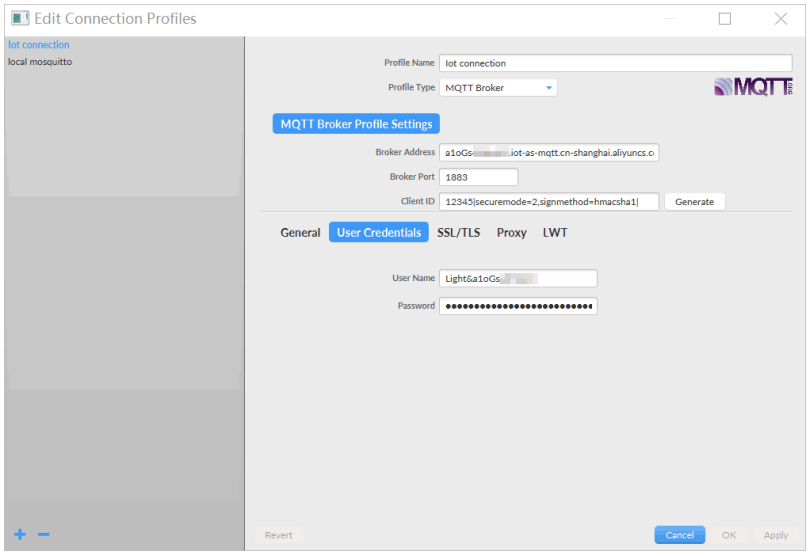
Parameter	Description	Example
Profile Name	Enter a custom name.	*iot connection*
Profile Type	Specify a connection mode. Select **MQTT Broker**.	
MQTT Broker Profile Settings		
Broker Address	Enter an endpoint. Format: `${YourProductKey}.iot-as-mqtt.${YourRegionId}.aliyuncs.com`. ■ *${YourProductKey}*: Replace this variable with the ProductKey of the product to which the device belongs. You can view the ProductKey on the **Device Details** page in the IoT Platform console. within the validity period before you enable the feature. ■ *${YourRegionId}*: Replace this variable with your region ID. For information about region IDs, see Regions and zones.	`a1oGs******.iot-as-mqtt.cn-shanghai.aliyuncs.com` ■ *a1oGs******* indicates the ProductKey. ■ *cn-shanghai* indicates the region ID.
Broker Port	Enter a port number. Set this parameter to *1883*.	

Parameter	Description	Example

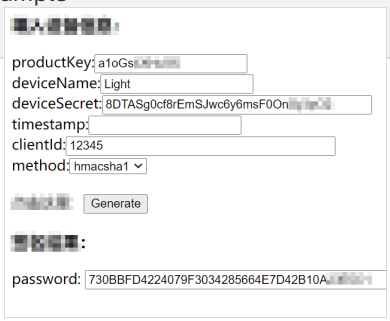
Client ID	Specify the fields in the MQTT protocol.
	Format: `${ClientId}\|securemode=${Mode},signmethod=${SignMethod}|`.
	Variables:
	■ *${ClientId}*: the ID of the client, such as a device, application, or web browser. You can specify a client ID as needed. The client ID must be 1 to 64 characters in length. In most cases, the client ID is the device ID. We recommend that you use the MAC address or serial number (SN) of the device as the client ID.
	■ *${Mode}*: the security mode.
	  ■ If you use a direct TCP connection, specify `securemode=3`. In this case, you do not need to set the SSL/TLS parameters.
	  ■ If you use a direct TLS connection, specify `securemode=2`. In this case, you must set the SSL/TLS parameters.
	■ *${SignMethod}*: the signature algorithm. Valid values: *hmacmd5* and *hmacsha1*.
	Notice
	■ Do not confuse the usage of the **Client ID** parameter and the *${ClientId}* variable.
	■ Do not omit the vertical bars (|) between and at the end of the parameters.
	■ When you set parameters, make sure that you remove all spaces from parameter values.
	■ After you specify the **Client ID** parameter, do not click **Generate**.
		In this example, the value of the ${ClientId} variable is *12345*. A direct TLS connection is used. The signature algorithm is *hmacsha1*.
		`12345|securemode=2,signmethod=hmacsha1|`

General	In this example, the default values of the parameters are used. You can set the parameters based on your business requirements.	

ii. Click the **User Credentials** tab. Set the **User Name** and **Password** parameters.

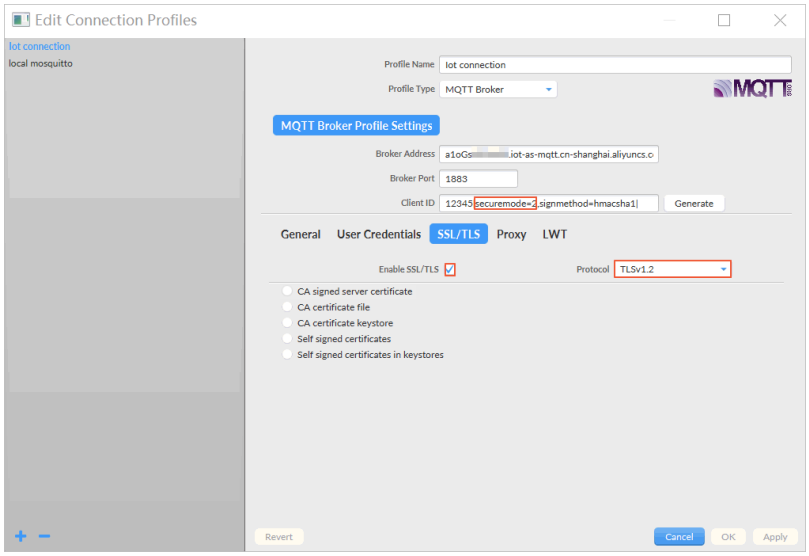


Parameter	Description	Example
User Name	The username consists of a DeviceName, ampersand (&), and ProductKey. Format: `${YourDeviceName}&${YourProductKey}`.	*Light&a1oGs****** ■ `Light` indicates the DeviceName of the device. ■ `a1oGs******` indicates the ProductKey of the device.
	To generate a password, you must select a signature algorithm, use the DeviceSecret of the device as a secret key, and then concatenate the required parameters and their values. **Notice** ■ Your MQTT.fx client may show a masked password. If a password is pasted, the pointer moves to the end of the password. In this case, you do not need to paste the password again. ■ Make sure that you specify valid uppercase and lowercase letters in parameter names and values. You can use one of the following methods to generate a password:	

Parameter	Description	Example
Password	■ Use a generation tool: Click MQTT_Password to download the tool. After you decompress the file, double-click the *sign.html* file and then set the parameters as prompted to generate a password. ■ productKey, deviceName, and deviceSecret: the information of the device certificate. You can view the certificate on the **Device Details** page of the IoT Platform console. ■ timestamp: the timestamp. This parameter is optional. ■ clientId: the ID of the device. The value of this parameter must be the same as the value of the *${ClientId}* variable that you set in the **Client ID** parameter. ■ method: the signature algorithm. The value of the parameter must be the same as the value of the *${SignMethod}* variable that you set in the **Client ID** parameter. ■ Manually generate a password by using an encryption function. The hmacsha1() function is used in this example. For more information about the sample code, see Appendix: Sample code for encryption. The field value varies based on the following operation types: ■ *${productKey}*, *${deviceName}*, and *${deviceSecret}*: Replace the variables with your device certificate information. ■ *${clientId}*: Replace this variable with the value of the *${ClientId}* variable that you set in the **Client ID** parameter.	■ Example of generating a password by using the tool  ■ Example of manually generating a password The timestamp parameter is not specified in this example. 

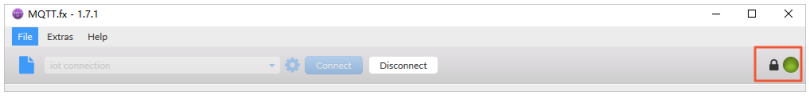
iii. If you use a TLS connection (securemode=2), click the **SSL/TLS** tab, select **Enable SSL/TLS**, and then set **Protocol** to **TLSv1.2**.

> **Notice** If you use a TCP connection (`securemode=3`), use the default settings on the SSL/TLS tab, and go to the next step.



4. Click **OK** in the lower-right corner.
5. Click **Connect**.

   If the indicator on the right side turns green, the connection is established.



   You can view the status of the device in the IoT Platform console.Choose **Devices > Devices**, select the product, and then find the device. The device is in the **Online** state.

In the following sections, downstream messaging and upstream messaging are tested to verify whether the MQTT.fx client is connected to IoT Platform. If your test results are different from the following sample results, the connection is not established. You must modify settings based on logs.

Test downstream messaging

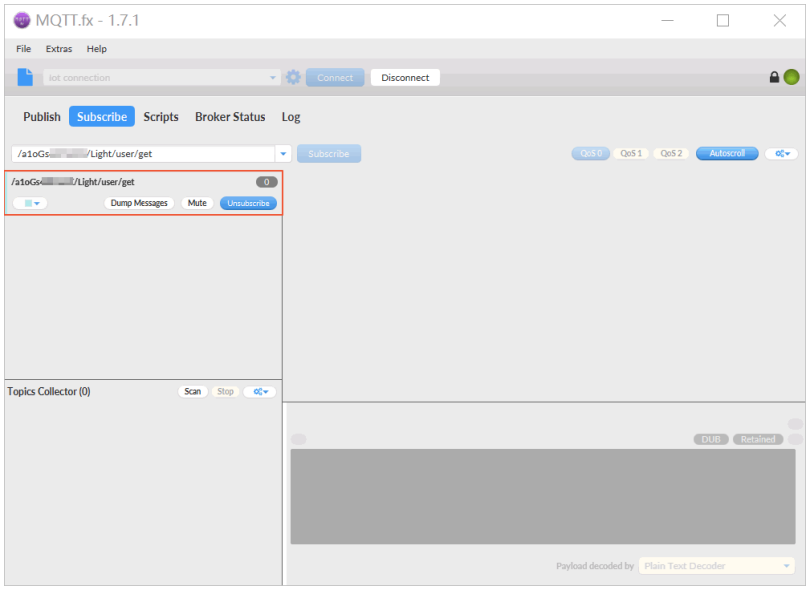
1. Log on to the IoT Platform console.On the Product Details page, choose **Topic Categories > Custom Topics**. Then, find a custom topic that has the **Subscribe** permission.

   The following topic is used in this example: `/a1oGs4X***/${deviceName}/user/get` . You must replace the `${deviceName}` variable with `Light` .

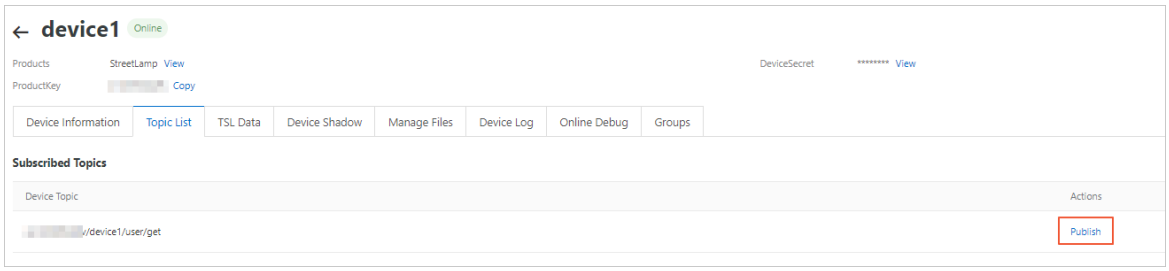
   For more information, see Custom topics.

2. In the MQTT.fx tool, click **Subscribe**. In the **Subscribe** field, enter the topic that is specified in the previous step, and then click **Subscribe**.

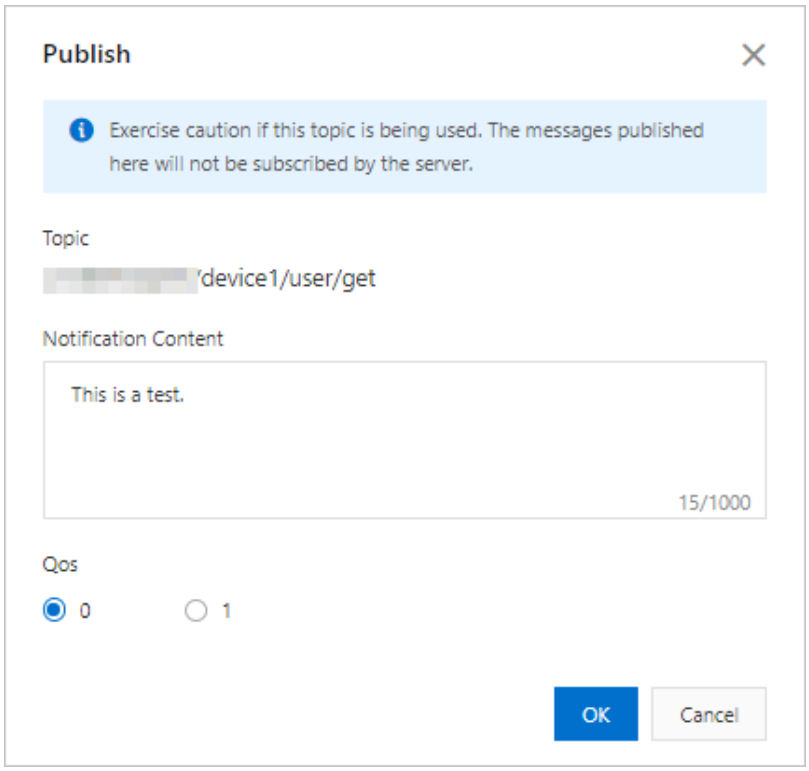
   After the subscription succeeds, the custom topic is displayed on the Subscribe tab.



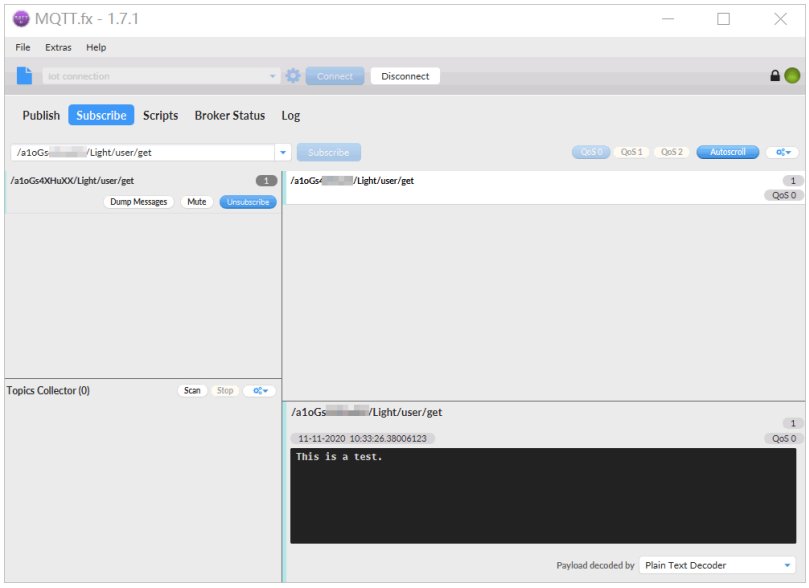
3. Go to the **Device Details** page in the IoT Platform console. On the **Topic List** tab, find the topic and click **Publish Message**.



4. Enter a message and click **OK**.



5. In the MQTT.fx tool, check whether the message is received.



6. Go to the **Device Details** page in the IoT Platform console. On the **Device Log** tab, click **View**. On the **Device Log** page, view cloud-to-device messages.

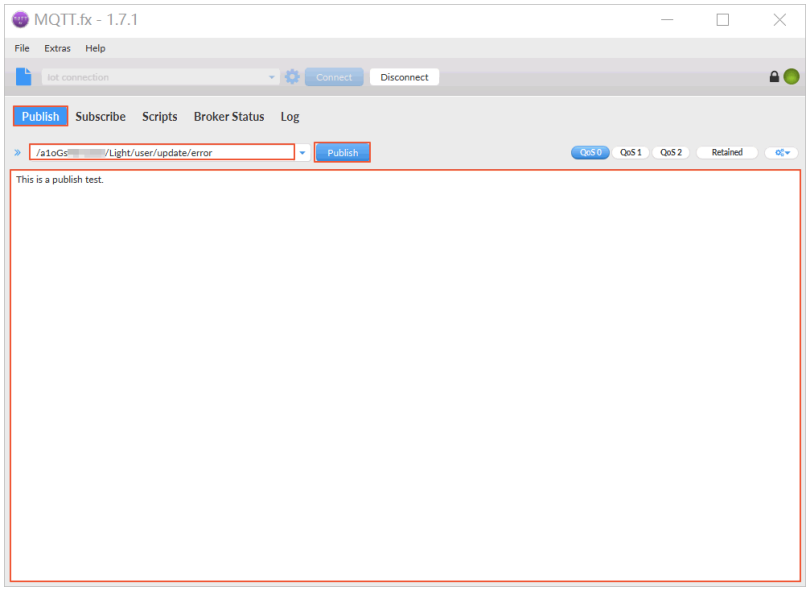
Test upstream messaging

1. Log on to the IoT Platform console.On the Product Details page, choose **Topic Categories > Custom Topics**. Then, find a custom topic that has the **Subscribe** permission.

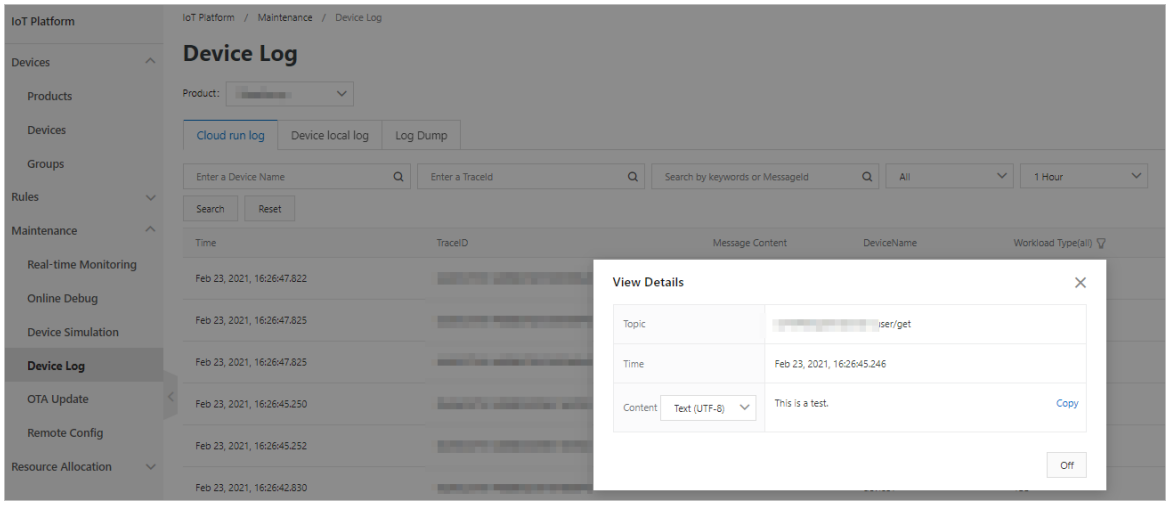
   The following topic is used in this example: `/a1oGs4X***/${deviceName}/user/get`. You must replace the `${deviceName}` variable with `Light`.

   For more information, see Custom topics.

2. In the MQTT.fx tool, click **Publish**. In the **Publish** field, enter the topic that is specified in the previous step. In the editor, enter the message to be sent and click **Publish**.

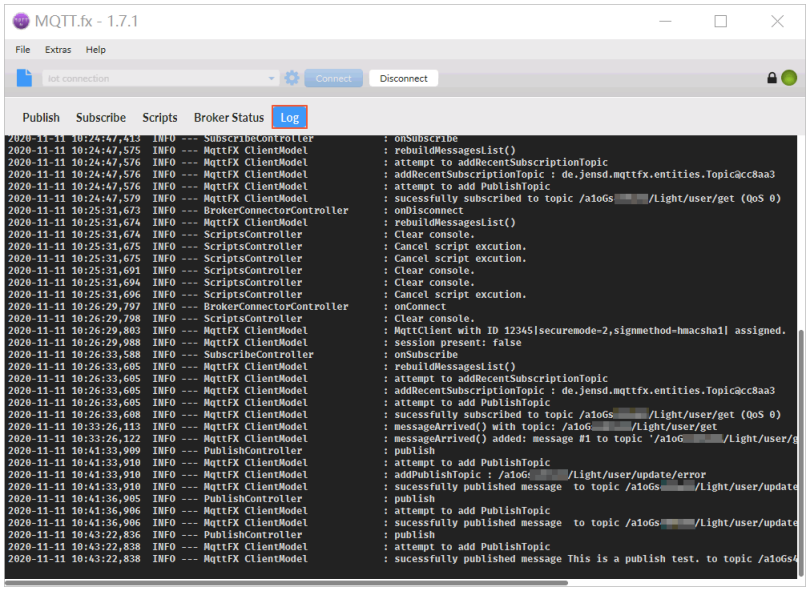


3. Go to the **Device Details** page in the IoT Platform console. On the **Device Log** tab, click **View**. On the **Device Log** page, view device-to-cloud messages.



View logs

In the MQTT.fx tool, click the **Log** tab. On the tab that appears, view operation logs and error logs.



Appendix: Sample code for encryption

```
package com.aliyun.iot.util;
import javax.crypto.Mac;
import javax.crypto.SecretKey;
import javax.crypto.spec.SecretKeySpec;
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;
public class SignDemo {
    public static String sign(Map<String, String> params, String deviceSecret, String signM
ethod) {
        // Sort the parameter keys in dictionary order.
        String[] sortedKeys = params.keySet().toArray(new String[] {});
        Arrays.sort(sortedKeys);
        // Generate a canonicalized request string.
        StringBuilder canonicalizedQueryString = new StringBuilder();
```

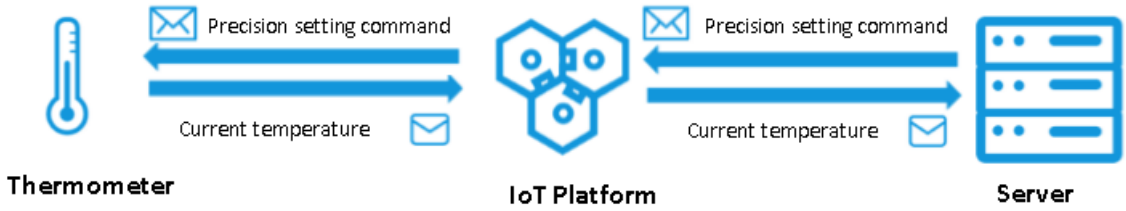
```
        for (String key : sortedKeys) {
            if ("sign".equalsIgnoreCase(key)) {
                continue;
            }
            canonicalizedQueryString.append(key).append(params.get(key));
        }
        try {
            String key = deviceSecret;
            return encryptHMAC(signMethod, canonicalizedQueryString.toString(), key);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
    public static String encryptHMAC(String signMethod, String content, String key) throws
Exception {
        SecretKey secretKey = new SecretKeySpec(key.getBytes("utf-8"), signMethod);
        Mac mac = Mac.getInstance(secretKey.getAlgorithm());
        mac.init(secretKey);
        byte[] data = mac.doFinal(content.getBytes("utf-8"));
        return bytesToHexString(data);
    }
    public static final String bytesToHexString(byte[] bArray) {
        StringBuffer sb = new StringBuffer(bArray.length);
        String sTemp;
        for (int i = 0; i < bArray.length; i++) {
            sTemp = Integer.toHexString(0xFF & bArray[i]);
            if (sTemp.length() < 2) {
                sb.append(0);
            }
            sb.append(sTemp.toUpperCase());
        }
        return sb.toString();
    }
    public static void main(String args[]) {
        Map<String, String> params = new HashMap<String, String>();
        params.put("productKey", "${productKey}");
        params.put("deviceName", "${deviceName}");
        params.put("clientId", "${clientId}");
        // The timestamp. This parameter is optional. When you specify this parameter, dele
te the forward slashes (//) before the following two lines of code.
        //String t = System.currentTimeMillis() + "";
        //params.put("timestamp", t);
        String sign = SignDemo.sign(params, "${deviceSecret}", "hmacsha1");
        System.out.print(sign);
    }
}
```

3.Use custom topics for communication

You can create custom topic categories in the IoT Platform console. Then, a device can send messages to a custom topic that belongs to a topic category. Your server can receive the messages by using an AMQP SDK. Your server can also call the Pub API operation to send commands to the device. Communication based on custom topics does not use the TSL model. In this case, you can define the data structure of the message.

Context

In this example, an electronic thermometer exchanges data with a server at a regular interval. The thermometer sends the real-time temperature data to the server, and the server sends the precision setting command to the thermometer.



Prepare the development environment

In this example, both devices and IoT Platform use SDKs for Java. You need to prepare the Java development environment first. You can download Java from the Java official website and install the Java development environment.

Create a product and devices

1. Log on to the IoT Platform console.
2.
3. In left-side navigation pane, choose **Devices > Products**.
4. Click **Create Product** to create a thermometer product. You must obtain the **productKey**, such as `a1uzcH0****`.

   For more information, see Create a product.
5. After the thermometer product is created, find it on the Products page and click **View** in the Actions column.
6. On the **Product Details** page, click the **Topic Categories** tab. On this tab, click **Topic Category** to add a custom topic category.

   For more information, see Custom topics.

   In this example, you must define the following two topic categories:

   - /a1uzcH0****/${deviceName}/user/devmsg: Devices send messages to this topic. Set the Publish permission for this topic category.
   - /a1uzcH0****/${deviceName}/user/cloudmsg: Devices subscribe to this topic. Set the Subscribe

permission for this topic category.

7. On the **Server-side Subscription** tab, click **Create Subscription** to create an AMQP server-side subscription. Set Message Type to **Device Upstream Notification** and Message Type to **Default Consumer Group**.

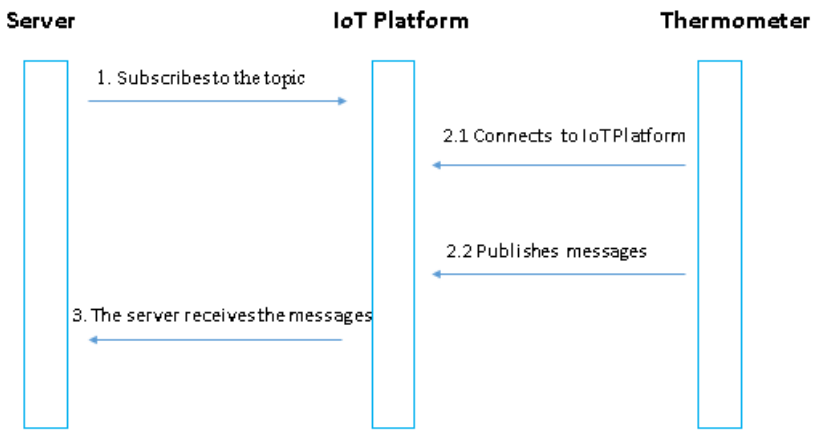
   For more information, see Configure an AMQP server-side subscription.

8. In the left-side navigation pane, click **Devices**. Then, add the **device1** device to the created thermometer product. Obtain the device certificate information including **ProductKey**, **DeviceName**, and **DeviceSecret**.

   For more information, see Create a device.

The device sends a message to the server

The following figure shows how the device sends a message to the server.



This section describes how to configure the server and the device to implement the process.

- The server receives messages by using an AMQP client. Therefore, you must configure the AMQP client to connect with IoT Platform and listen to device messages. For more information, see Connect a client to IoT Platform by using the SDK for Java.

  > **Notice** The consumer group in the AMQP server-side subscription must be in the same IoT Platform instance as the device.

- Configure the device SDK to connect the device with IoT Platform and enable the device to send messages.
  - Set the parameters to authenticate the device.

```
final String productKey = "a1uzcH0****";
final String deviceName = "device1";
final String deviceSecret = "uwMTmVAMnxB***";
final String region = "cn-shanghai";
```

- Set the parameters to initialize the connection, including the MQTT connection parameters, device parameters, and TSL model parameters.

```
LinkKitInitParams params = new LinkKitInitParams();
// Set the MQTT connection parameters. Link SDK uses MQTT as the underlying protocol.
IoTMqttClientConfig config = new IoTMqttClientConfig();
config.productKey = productKey;
config.deviceName = deviceName;
config.deviceSecret = deviceSecret;
config.channelHost = productKey + ".iot-as-mqtt." + region + ".aliyuncs.com:1883";
// Set the device parameters.
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
// Set the initial status of the device.
Map<String, ValueWrapper> propertyValues = new HashMap<String, ValueWrapper>();
params.mqttClientConfig = config;
params.deviceInfo = deviceInfo;
params.propertyValues = propertyValues;
```

- Initialize the connection.

```
// Initialize the connection and configure the callback function that is used after the
initialization succeeds.
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
     @Override
     public void onError(AError aError) {
         System.out.println("Init error:" + aError);
     }
     // Implement the callback function.
     @Override
     public void onInitDone(InitResult initResult) {
         System.out.println("Init done:" + initResult);
     }
 });
```

- Send a message from the device.

  After the device is connected to IoT Platform, you can send a message to the specified topic. Replace the content of the onInitDone function, as shown in the following example:

```
@Override
 public void onInitDone(InitResult initResult) {
     // Set the topic to which the message is published and the message content.
     MqttPublishRequest request = new MqttPublishRequest();
     request.topic = "/" + productKey + "/" + deviceName + "/user/devmsg";
     request.qos = 0;
     request.payloadObj = "{\"temperature\":35.0, \"time\":\"sometime\"}";
     // Publish the message and configure the callback functions that are used after th
e message is published.
     LinkKit.getInstance().publish(request, new IConnectSendListener() {
         @Override
         public void onResponse(ARequest aRequest, AResponse aResponse) {
             System.out.println("onResponse:" + aResponse.getData());
         }
         @Override
         public void onFailure(ARequest aRequest, AError aError) {
             System.out.println("onFailure:" + aError.getCode() + aError.getMsg());
         }
     });
 }
```

  The server receives the following message:

```
Message
{payload={"temperature":35.0, "time":"sometime"},
topic='/a1uzcH0****/device1/user/devmsg',
messageId='1131755639450642944',
qos=0,
generateTime=1558666546105}
```

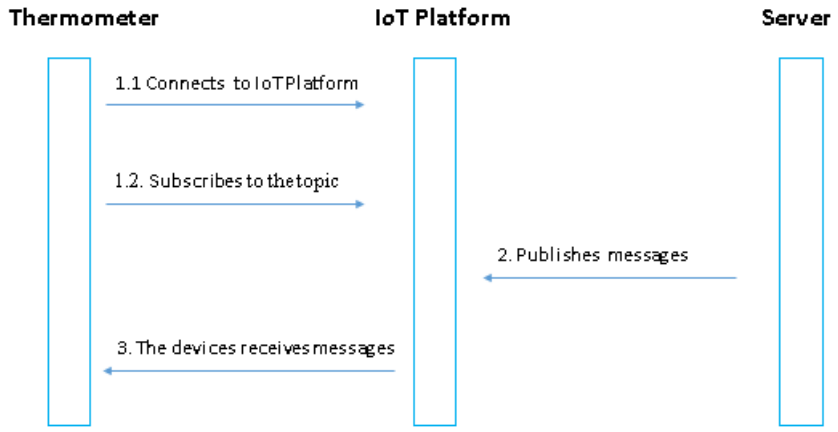
In actual business scenarios, you must modify the following parameter values.

Parameter	Example	Description
productKey	a1uzcH0****	The device authentication information. For more information, see Step 7 in the Create a product and devices section.
deviceName	device1	
deviceSecret	uwMTmVAMnxB***	
region	cn-shanghai	The ID of the region where your IoT Platform instance resides. For more information about region IDs, see Regions and zones.
request.topic	"/" + productKey + "/" + deviceName + "/user/devmsg"	The custom topic that has the Publish permission.

Parameter	Example	Description
request.payloadObj	"{\"temperature\":35.0, \"time\":\"sometime\"}"	The content of the custom message.

The server sends a message to the device

The following figure shows how the server sends a message to the device.



- Configure the device SDK to subscribe to a topic.

  For more information about how to specify the parameters to authenticate the device and initialize the connection, see the sample code in the The device sends a message to the server section.

  The device must subscribe to a specific topic before the device can receive messages sent by the server.

  The following example shows how to configure the device SDK to subscribe to a topic:

```
// Implement the callback function.
@Override
public void onInitDone(InitResult initResult) {
    // Set the topic to which the device subscribes.
    MqttSubscribeRequest request = new MqttSubscribeRequest();
    request.topic = "/" + productKey + "/" + deviceName + "/user/cloudmsg";
    request.isSubscribe = true;
    // Send a subscription request and configure the callback functions that are used aft
er the subscription succeeds or fails.
    LinkKit.getInstance().subscribe(request, new IConnectSubscribeListener() {
        @Override
        public void onSuccess() {
            System.out.println("");
        }
        @Override
        public void onFailure(AError aError) {
        }
    });
    // Set the listener that listens to downstream messages.
    IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
        // Configure the callback function that is used after the downstream messages are
received.
        @Override
        public void onNotify(String connectId, String topic, AMessage aMessage) {
            System.out.println(
                "received message from " + topic + ":" + new String((byte[])aMessage.getD
ata()));
        }
        @Override
        public boolean shouldHandle(String s, String s1) {
            return false;
        }
        @Override
        public void onConnectStateChange(String s, ConnectState connectState) {
        }
    };
    LinkKit.getInstance().registerOnNotifyListener(notifyListener);
}
```

You must change the value of the `request.topic` parameter to a custom topic that has the Subscribe permission.

- Configure IoT Platform SDK to call the Pub operation to publish a message. For more information about the parameters, see Pub. For more information about how to configure the SDK, see Use IoT Platform SDK for Java.
  - Set the parameters to authenticate the device.

```
String regionId = "XXXXXX";
String accessKey = "XXXXXX";
String accessSecret = "XXXXXXXXX";
final String productKey = "XXXXXX";
```

- Set the connection parameters.

```
// Set the parameters of the client.
DefaultProfile profile = DefaultProfile.getProfile(regionId, accessKey, accessSecret);
IAcsClient client = new DefaultAcsClient(profile);
```

- Set the parameters that are used to publish a message.

```
PubRequest request = new PubRequest();
request.setQos(0);
// Set the topic to which the message is published.
request.setTopicFullName("/" + productKey + "/" + deviceName + "/user/cloudmsg");
request.setProductKey(productKey);
// Set the MessageContent parameter. The message content must be encoded in Base64. Oth
erwise, the message content appears as garbled characters.
request.setMessageContent(Base64.encode("{\"accuracy\":0.001,\"time\":now}"));
```

- Publish the message.

```
try {
     PubResponse response = client.getAcsResponse(request);
     System.out.println("pub success?:" + response.getSuccess());
 } catch (Exception e) {
     System.out.println(e);
 }
```

The device receives the following message:

```
msg = [{"accuracy":0.001,"time":now}]
```

Appendix: Sample code

> **Note** In actual business scenarios, you must modify the parameter values in the code.

You can download the sample code in this article. To download the code, click PubSubDemo.

For more information about how to connect an AMQP client with IoT Platform, see the following articles:

- Connect a client to IoT Platform by using the SDK for Java
- Connect a client to IoT Platform by using the SDK for .NET
- Connect a client to IoT Platform by using the SDK for Node.js
- Connect a client to IoT Platform by using the SDK for Python 2.7
- Connect a client to IoT Platform by using the SDK for Python 3
- Connect a client to IoT Platform by using the SDK for PHP
- Connect a client to IoT Platform by using the SDK for Go