

ALIBABA CLOUD

Alibaba Cloud

物联网平台
Quick Start

Document Version: 20201126

 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 .
<code>Courier font</code>	Courier font is used for commands	Run the <code>cd /d C:/window</code> command to enter the Windows system folder.
<i>Italic</i>	Italic formatting is used for parameters and variables.	<code>bae log list --instanceid</code> <i>Instance_ID</i>
[] or [a b]	This format is used for an optional value, where only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	This format is used for a required value, where only one item can be selected.	<code>switch {active stand}</code>

Table of Contents

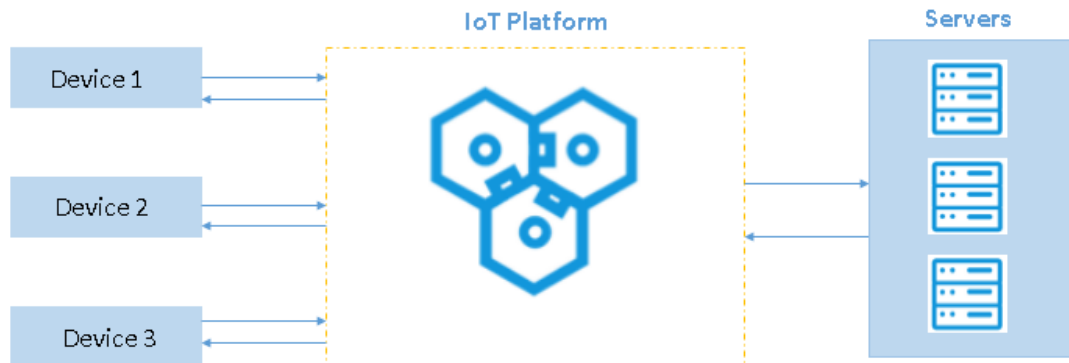
1. Use IoT Platform	05
1.1. Overview	05
1.2. Create products and devices	06
1.3. Define product features	07
1.4. Connect a device to IoT Platform	16
1.5. Subscribe to device messages from IoT Platform	18
1.6. Issue commands to devices from IoT Platform	22
2. Access IoT Platform by using MQTT.fx	25
3. Use custom topics for communication	33

1. Use IoT Platform

1.1. Overview

This topic describes how to use IoT Platform.

Data flow diagram



Prerequisites

Make sure that you have completed the following preparations before performing the operations described in this chapter.

- The **IoT Platform** service is active.
- The development environment for the C programming language is ready. In the example, development of the device is based on Linux. The Link Kit SDK in C programming language provided by Alibaba Cloud is used.
- The development environment for Java is ready. In the example, the Java SDK for server side provided by Alibaba Cloud is used to issue commands.

Procedure

1. **Create products and devices:** Register your device in IoT Platform and obtain a device certificate (including ProductKey, DeviceName, and DeviceSecret). Install the device certificate in the device. When the device connects to IoT Platform, it reports its certificate to IoT Platform for authentication.
2. **Define product features:** Define product features in terms of properties, services, and events. IoT Platform builds a TSL model based on the defined product features. The model helps with communication between the cloud and the device.
3. **Connect a device to IoT Platform:** Use device SDKs and pass in the certificate information of the device to connect the device to IoT Platform.
4. **Subscribe to device messages from IoT Platform:** Subscribe to messages of specific types from the device, and use AMQP clients to receive these messages.
5. **Issue commands to devices from IoT Platform:** Call the IoT Platform APIs to issue commands to the device.

1.2. Create products and devices

To start using IoT Platform, you need to create products and devices first. In IoT Platform, a product is a collection of devices that have the same features. You can create products to manage devices in batches.

Procedure

1. Log on to the [IoT Platform console](#).
2. Create a product.
 - i. In the left-side navigation pane, click **Devices > Product**. On the **Products** page, click **Create Product**.
 - ii. Specify the parameters as follows to create a test product.

For more information about product configurations, see [Create product](#).

Create a product (device model)

Product Information (Device TSL)

* Product Name
TestBulb

* Category ?
 Standard Category Custom Category

* Node Type
 Directly C... Gateway s... Gateway ...

Networking and Data Format

* Network Connection Method
WIFI

* Data Type
ICA Standard Data Format (Alink JSON)

* Authentication Mode
Device Secret

- iii. Click **OK**.

You can find the newly created product in the product list.

3. Create a device.
 - i. In the left-side navigation pane, click **Devices > Device**.

- ii. On the Devices page, click **Add Device**. Select the newly created product, enter a device name and alias, and click **OK**.

Add Device [Close]

Note: You do not need to specify DeviceName. If DeviceName is not specified, Alibaba Cloud issues a unique identifier as DeviceName.

Products
test1119 [Dropdown Arrow]

DeviceName [Device1]

Alias [test_device]

[OK] [Cancel]

- iii. Save the device certificate information.

The certificate information includes ProductKey, DeviceName, and DeviceSecret. The certificate information is used for identity authentication when the device connects to IoT Platform. Make sure to keep it confidential.

View Device Certificate [Close]

Note: Device certificate is used to authenticate devices connecting to the platform. Keep it in a safe place.

ProductKey	a1r0M4F0g	Copy
DeviceName	Light001	Copy
DeviceSecret	*****	Show

[Copy] [Close]

What's next

Define product features.

1.3. Define product features

IoT Platform allows you to define features for products. You can use a TSL model to describe product features, including properties, services, and events. The TSL model makes it easy to manage products and data transmission. After you create a product, you can define a TSL model to describe product features. Devices under this product automatically inherit its features.

Procedure

1. In the product list, select the product and click **View**.
2. On the Product Details page, click **Define Feature**.
3. In the **Self-Defined Feature** section, click **Add Self-defined Feature**.
4. As shown below, add a property to define a switch.

The screenshot shows a dialog box titled "Add self-defined feature" with a close button in the top right corner. The dialog contains the following fields and options:

- * Feature Type:** Three tabs labeled "Properties", "Services", and "Events". The "Properties" tab is selected.
- * Feature Name:** A text input field containing "PowerSwitch".
- * Identifier:** A text input field containing "PowerSwitch".
- * Data Type:** A dropdown menu with "bool" selected.
- * Boolean Value:** Two text input fields: "0 - OFF" and "1 - ON".
- Read/Write Type:** Two radio buttons: "ReadWrite" (selected) and "Read-only".
- Description :** A text area with the placeholder text "Enter a description" and a character count "0/100" at the bottom right.

At the bottom right of the dialog are two buttons: "OK" and "Cancel".

5. As shown below, add a property to define a counter.

The screenshot shows a configuration window with three tabs: 'Properties' (selected), 'Services', and 'Events'. The 'Properties' tab contains the following fields:

- * Feature Name:** A text input field containing 'COUNTER'.
- * Identifier:** A text input field containing 'Counter'.
- * Data Type:** A dropdown menu with 'int32' selected.
- * Value Range:** Two text input fields containing '1' and '9999', separated by a tilde '~'.
- * Step:** A text input field containing '1'.
- Unit :** A dropdown menu with 'Select a unit' selected.
- Read/Write Type:** Two radio buttons: 'Read/Write' (unselected) and 'Read-only' (selected).
- Description :** A text area with the placeholder text 'Enter a description' and a character count '0/100' at the bottom right.

At the bottom right of the window are two buttons: 'OK' and 'Cancel'.

6. As shown below, add a service to support numerical calculations.

* Feature Type:
 ?

* Feature Name:
 ?

* Identifier:
 ?

* Invoke Method::
 Asynchronous Synchronous ?

Input Parameters:

<input type="text" value="Parameter Name: ValueA"/>	Edit Delete
<input type="text" value="Parameter Name: ValueB"/>	Edit Delete

[+ Add Parameter](#)

Output Parameters:

<input type="text" value="Parameter Name: Result"/>	Edit Delete
---	---

[+ Add Parameter](#)

Description :

0/100

- o Value A is defined as follows:

Add Parameter ✕

* Parameter Name:

* Identifier:

* Data Type:

* Value Range:
 ~

* Step:

Unit :

- o Value B is defined as follows:

Add Parameter ✕

* Parameter Name:

* Identifier:

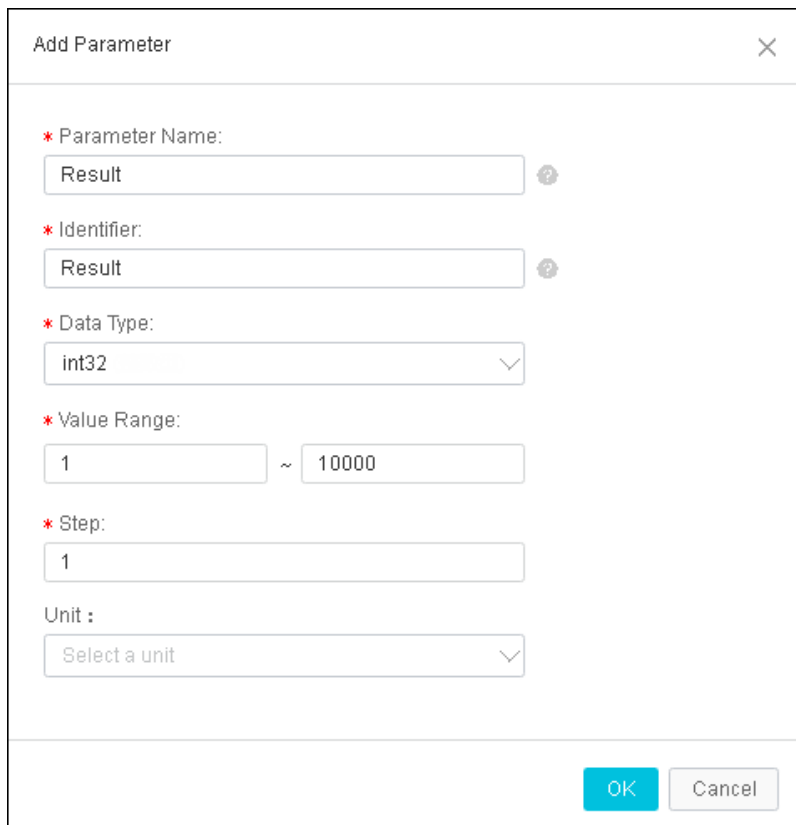
* Data Type:

* Value Range:
 ~

* Step:

Unit :

- The output parameter indicates the calculation result.



The screenshot shows a dialog box titled "Add Parameter" with a close button (X) in the top right corner. The dialog contains the following fields:

- * Parameter Name:** A text input field containing "Result".
- * Identifier:** A text input field containing "Result".
- * Data Type:** A dropdown menu showing "int32".
- * Value Range:** Two text input fields, the first containing "1" and the second containing "10000", separated by a tilde (~) symbol.
- * Step:** A text input field containing "1".
- Unit :** A dropdown menu showing "Select a unit".

At the bottom right of the dialog, there are two buttons: "OK" (highlighted in blue) and "Cancel".

7. As shown below, add an event to define a hardware error.

Add self-defined feature ✕

* Feature Type:
 Properties Services Events ⊕

* Feature Name:
 ⊕

* Identifier:
 ⊕

* Event Type:
 Info Alert Error ⊕

Output Parameters:
 Parameter Name: ErrorCode Edit Delete

[+ Add Parameter](#)

Description :

0/100

- o The output parameter indicates the error code.

Parameter Name: ErrorCode

Identifier: ErrorCode

Data Type: enum

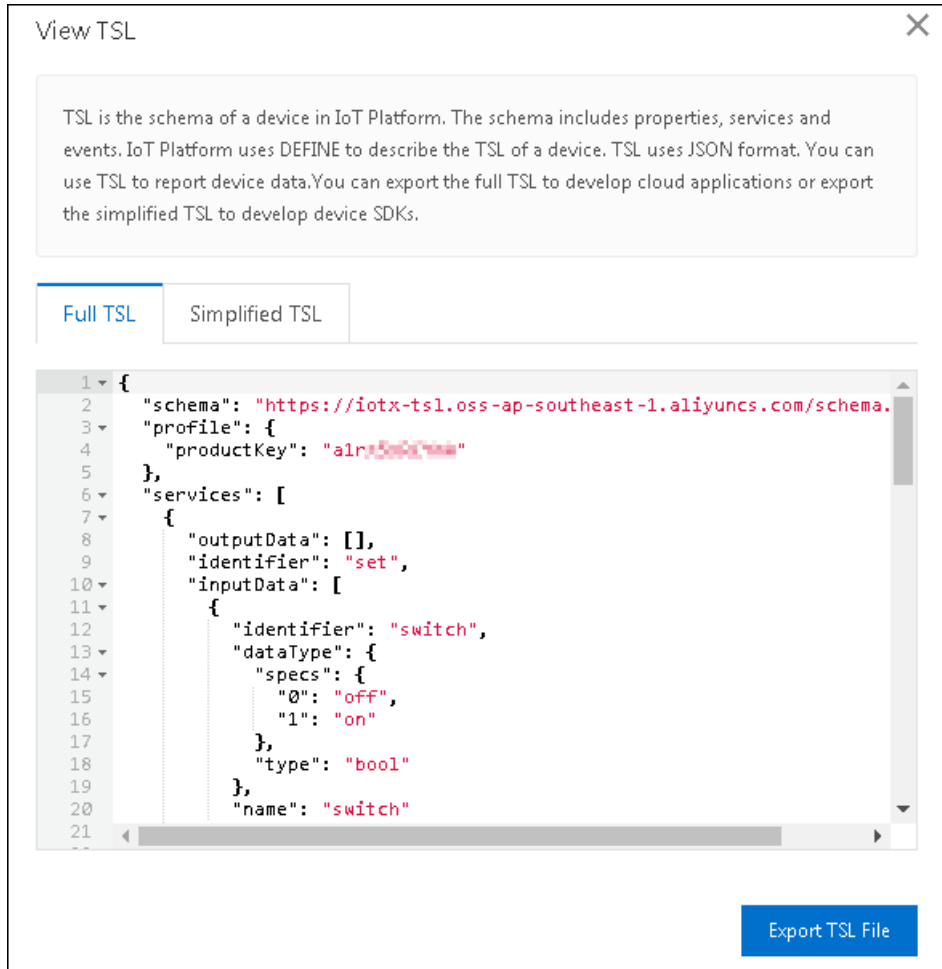
Enum Item:

Value	Description	
0	Error1	Delete
1	Error2	Delete
2	Error3	Delete

+ Add Enum Item

OK Cancel

8. Click **View TSL** and choose **Full TSL** to view the TSL definitions in JSON format.




9. 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 must be 1 to 100 characters in length, and can contain letters, digits, and special characters.

- 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. If the current version is configured as normal, click **Confirm**. In the **Release model online?** dialog box, the checkbox is automatically selected.

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

 **Note**

- A TSL model is applied to the product only after it is released.
- IoT Platform can save the latest 10 versions of a TSL model. Earlier versions are overwritten.

What's next

[Connect a device to IoT Platform](#)

1.4. Connect a device to IoT Platform

Alibaba Cloud IoT Platform provides device SDKs that you can use to connect devices to IoT Platform. This topic uses the sample program `linkkit-example-solo` to demonstrate how to connect a device to IoT Platform.

Context

- The following example uses a C SDK 3.0.1. We recommend that you use the SDK in a 64-bit Ubuntu 16.04 system.
- The following software is required:

make-4.1, git-2.7.4, gcc-5.4.0, gcov-5.4.0, lcov-1.12, bash-4.3.48, tar-1.28, and mingw-5.3.1.


You can run the following command to install the software:

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

Procedure

1. Log on to your Linux VM instance.
2. Download the [Link Kit SDK](#).
3. Use the unzip command to extract files from the package.
4. Before you call HAL to send device identity information to the SDK, you need to change the device certificate information in `wrappers/os/ubuntu/HAL_OS_linux.c` to the certificate information of the device added in [Create products and devices](#). Make sure to save your changes.

Enter the ProductKey, DeviceName, and DeviceSecret as follows. Device certificate information is used for identity authentication when the device connects to IoT Platform.

 **Note** In the quick start guide, the [Unique-certificate-per-device authentication](#) method is used and ProductSecret is optional.

```
#ifdef DEVICE_MODEL_ENABLED
char _product_key[IOTX_PRODUCT_KEY_LEN + 1] = "a1zlu*****";
char _product_secret[IOTX_PRODUCT_SECRET_LEN + 1] = "";
char _device_name[IOTX_DEVICE_NAME_LEN + 1] = "device1";
char _device_secret[IOTX_DEVICE_SECRET_LEN + 1] = "ynNuxxxxxxxxxx7RMUJD9WZhEvd7*****";
```


5. If your product is not created in the public instance in the China (Shanghai) region, find the following position in `src/dev_model/examples/linkkit_example_solo.c`.

```

/* post reply doesn't need */
post_reply_need = 1;
IOT_Ioctl(IOTX_IOCTL_RECV_EVENT_REPLY, (void *)&post_reply_need);
/* Create Master Device Resources */
g_user_example_ctx.master_devid = IOT_Linkkit_Open(IOTX_LINKKIT_DEV_TYPE_MASTER, &master_
meta_info);
if (g_user_example_ctx.master_devid < 0) {
    EXAMPLE_TRACE("IOT_Linkkit_Open Failed\n");
    return -1;
}

```

Add your MQTT endpoint information as follows in that position . Make sure to save your changes.

```

/* post reply doesn't need */
post_reply_need = 1;
IOT_Ioctl(IOTX_IOCTL_RECV_EVENT_REPLY, (void *)&post_reply_need);
// Enter the MQTT endpoint of your instance. In the left-side navigation pane of the IoT Platform console, click Instances. On the page that appears, click View in the Actions column of the instance. On the Instance Details page, you can view the endpoint.
char custom_domain[] = "${YourEndpoint}";
IOT_Ioctl(IOTX_IOCTL_SET_MQTT_DOMAIN, (void *)custom_domain);
/* Create Master Device Resources */
g_user_example_ctx.master_devid = IOT_Linkkit_Open(IOTX_LINKKIT_DEV_TYPE_MASTER, &master_
meta_info);
if (g_user_example_ctx.master_devid < 0) {
    EXAMPLE_TRACE("IOT_Linkkit_Open Failed\n");
    return -1;
}

```

6. In the root directory, use the make command to compile the sample program.

```

$ make distclean
$ make

```

The sample program `linkkit-example-solo` is saved under the directory `./output/release/bin`.

7. Run the sample program.

```

./output/release/bin/linkkit-example-solo

```

In the IoT Platform console, the device status is **online**, the device is connected to IoT Platform.

After the device is connected to IoT Platform, it automatically reports messages to IoT Platform. You can check log files for message details.

What's next

Device messages subscribed by the server

1.5. Subscribe to device messages from IoT Platform

After a device is connected to IoT Platform, the device directly reports data to IoT Platform. Then, the data is forwarded to your server over AMQP. This topic describes how to configure the server-side subscription function..

Context



Procedure

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose **Rules > Server-side Subscription**.
3. On the **Subscriptions** tab of the **Server-side Subscription** page, click **Create Subscription**.
4. In the **Create Subscription** dialog box that appears, set the following parameters and click **OK**.
5. Develop an AMQP client to receive messages.

AMQP SDKs are open-source SDKs. We recommend that you use the Apache Qpid JMS client for the Java development environment. You can visit [Qpid JMS 0.47.0](#) to download the client and read its instructions.

Add the Maven dependency

```
<!-- amqp 1.0 qpid client -->
<dependency>
  <groupId>org.apache.qpid</groupId>
  <artifactId>qpid-jms-client</artifactId>
  <version>0.47.0</version>
</dependency>
<!-- util for base64-->
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.10</version>
</dependency>
```

Sample code:

For information about the parameters in the following demo, see [Connect an AMQP client to IoT Platform](#).

```
import java.net.URI;
import java.util.Hashtable;
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.apache.commons.codec.binary.Base64;
import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionListener;
import org.apache.qpid.jms.message.JmsInboundMessageDispatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class AmqpJavaClientDemo {
    private final static Logger logger = LoggerFactory.getLogger(AmqpJavaClientDemo.class);
    public static void main(String[] args) throws Exception {
        //For information about parameters, see AMQP client access instructions.
```

```

String accessKey = "${YourAccessKeyId}";
String accessSecret = "${YourAccessKeySecret}";
String consumerGroupId = "${YourConsumerGroupId}";
long timeStamp = System.currentTimeMillis();
//Signature method: hmacmd5, hmacsha1, or hmacsha256.
String signMethod = "hmacsha1";
//The value of the clientId parameter is displayed as the client ID on the consumer group status page
for server-side subscription in the console.
//We recommend that you use a unique identifier, such as the UUID, MAC address, or IP address as th
e clientId value. This allows you to identify clients.
String clientId = "${YourClientId}";
//For information about how to obtain the value of UserName, see AMQP client access instructions.
String userName = clientId + "| authMode = aksign"
+ ",signMethod=" + signMethod
+ ",timestamp=" + timeStamp
+ ",authId=" + accessKey
+ ",consumerGroupId=" + consumerGroupId
+ "|";
//For information about how to obtain the value of password, see AMQP client access instructions.
String signContent = "authId=" + accessKey + "& timestamp=" + timeStamp;
String password = doSign(signContent,accessSecret, signMethod);
//Construct the connection URL based on the rules described in Qpid JMS documentation.
String connectionUrl = "failover:(amqps://${uid}.iot-amqp.${regionId}.aliyun.com:5671? amqp.idle
Timeout=80000)"
+ "? failover.reconnectDelay=30";
Hashtable<String, String> hashtable = new Hashtable<>();
hashtable.put("connectionfactory.SBCF",connectionUrl);
hashtable.put("queue.QUEUE", "default");
hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFa
ctory");
Context context = new InitialContext(hashtable);
ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
Destination queue = (Destination)context.lookup("QUEUE");
// Create Connection
Connection connection = cf.createConnection(userName, password);
((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
// Create Session
// Session. Client_acknowledgment: After you receive the message, manually call message.acknowle
dge().
// Session. Auto_acknowledgment: The SDK is automatically acknowledged. (recommended)
Session session = connection.createSession (false, Session.Auto_acknowledgment );

```

```

connection.start();
// Create Receiver Link
MessageConsumer consumer = session.createConsumer(queue);
consumer.setMessageListener(messageListener);
}
private static MessageListener messageListener = new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            byte[] body = message.getBody(byte[].class);
            String content = new String(bytes);
            String topic = message.getStringProperty("topic");
            String messageId = message.getStringProperty("messageId");
            logger.info("receive message"
                + ", topic = " + topic
                + ", messageId = " + messageId
                + ", content = " + content);
            //If you select Session.CLIENT_ACKNOWLEDGE when creating a session, manual acknowledgment is required.
            //message.acknowledge();
            //Ensure that no time-consuming logic exists during message processing. If processing received messages takes a long period of time, initiate asynchronous requests.
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
};
private static JmsConnectionListener myJmsConnectionListener = new JmsConnectionListener() {
    /**
     * The connection is successfully established.
     */
    @Override
    public void onConnectionEstablished(Uri remoteUri) {
        logger.info("onConnectionEstablished, remoteUri:{}", remoteUri);
    }
    /**
     * The connection fails after the maximum number of retries.
     */
    @Override
    public void onConnectionFailure(Throwable error) {
        logger.error("onConnectionFailure. {}", error.getMessage());
    }
}

```

```

    logger.info("onConnectionInterrupted, remoteUri:{}", remoteUri);
}
/**
 * The connection is interrupted.
 */
@Override
public void onConnectionInterrupted(Uri remoteUri) {
    logger.info("onConnectionInterrupted, remoteUri:{}", remoteUri);
}
/**
 * The connection is interrupted and automatically restored.
 */
@Override
public void onConnectionRestored(Uri remoteUri) {
    logger.info("onConnectionRestored, remoteUri:{}", remoteUri);
}
@Override
public void onInboundMessage(JmsInboundMessageDispatch envelope) {}
@Override
public void onSessionClosed(Session session, Throwable cause) {}
@Override
public void onConsumerClosed(MessageConsumer consumer, Throwable cause) {}
@Override
public void onProducerClosed(MessageProducer producer, Throwable cause) {}
};
/**
 * To obtain the string to sign in the value of password, see AMQP client access instructions.
 */
private static String doSign(String toSignString, String secret, String signMethod) throws Exception {
    SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(), signMethod);
    Mac mac = Mac.getInstance(signMethod);
    mac.init(signingKey);
    Byte [] rawHmac = mac.doFinal (toSignString.getBytes ());
    return Base64.encodeBase64String(rawHmac);
}
}

```

1.6. Issue commands to devices from IoT Platform

After devices successfully report messages to IoT Platform, you can try to issue commands to devices from IoT Platform. This topic describes how to call the SetDeviceProperty operation to configure device properties by issuing commands from IoT Platform to devices.

Procedure

1. Import the SDK dependency into a maven project.

This example shows how to import the IoT Platform Java SDK dependency into a maven project.

```
<!-- https://mavenrepository.com/artifact/com.aliyun/aliyun-java-sdk-iot -->
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-iot</artifactId>
  <version>6.8.0</version>
</dependency>
```

Import the core module of the SDK.

```
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <version>3.5.1</version>
</dependency>
```

2. Initialize the SDK.


The region ID in the endpoint must be consistent with the region of the device. In this example, the region ID is cn-shanghai.

```
String accessKey = "<your accessKey>";
String accessSecret = "<your accessSecret>";
DefaultProfile.addEndpoint("cn-shanghai", "cn-shanghai", "iot", "iot.cn-shanghai.aliyuncs.com");
IClientProfile profile = DefaultProfile.getProfile("cn-shanghai", accessKey, accessSecret);
DefaultAcsClient client = new DefaultAcsClient(profile);
```

3. Call the SetDeviceProperty operation to send a request to a device and set its LightSwitch property to 1.

Example:

```
SetDevicePropertyRequest request = new SetDevicePropertyRequest();
request.setProductKey("a1zluR09W76");
request.setDeviceName("device1");
JSONObject itemJson = new JSONObject();
itemJson.put("PowerSwitch", 1);
request.setItems(itemJson.toString());
try {
    SetDevicePropertyResponse response = client.getAcsResponse(request);
    System.out.println(response.getRequestId() + ", success: " + response.getSuccess());
} catch (ClientException e) {
    e.printStackTrace();
}
```

 **Note** For more information about the SetDeviceProperty operation, see [SetDeviceProperty](#).

Result

If the device successfully receives the request, the log output is as follows:

```
<{
< "method": "thing.service.property.set",
< "id": "432801169",
< "params": {
< "PowerSwitch": 1
< },
< "version": "1.0.0"
<}
user_report_reply_event_handler. 94: Message Post Reply Received, Message ID: 646, Code: 200, Reply: {}
user_property_set_event_handler. 114: Property Set Received, Request: {"PowerSwitch":1}
>{
> "id": "647",
> "version": "1.0",
> "params": {
> "PowerSwitch": 1
> },
> "method": "thing.event.property.post"
>}
```


2. Access IoT Platform by using MQTT.fx

This article uses MQTT.fx as an example to describe the method for using a third-party MQTT client to connect to IoT Platform. MQTT.fx is a MQTT client that is written in Java language and based on Eclipse Paho. It supports subscribing to messages and publishing messages through topics.

Prerequisites

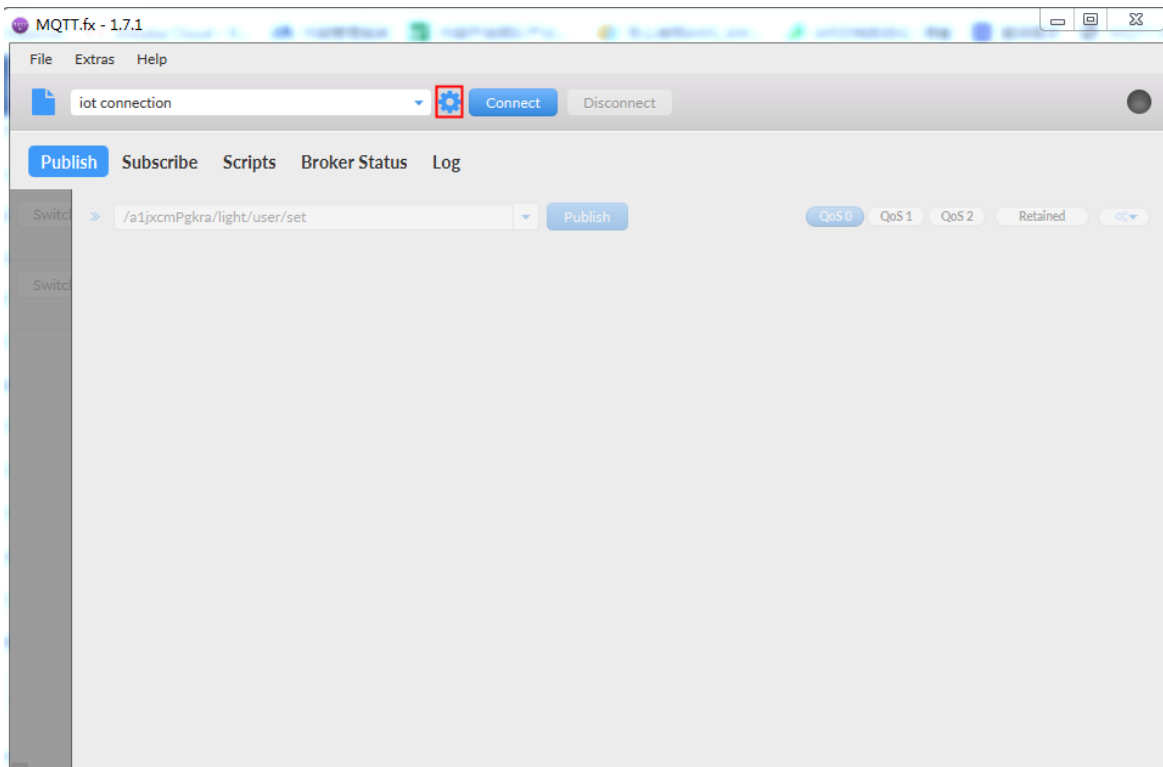
You have created products and devices in the [IoT Platform console](#), and have got the ProductKey, DeviceName, and DeviceSecret of the devices. When you set the connection parameters for MQTT.fx, you will use the values of the ProductKey, DeviceName, and DeviceSecret. See [Create a product](#), [Create a device](#), and [Create multiple devices at a time](#) for help when creating products and devices.

Procedure

1. Download and install the MQTT.fx software.

Download the MQTT.fx software for Windows from [MQTT.fx website](#).

2. Open MQTT.fx, and click the settings icon.



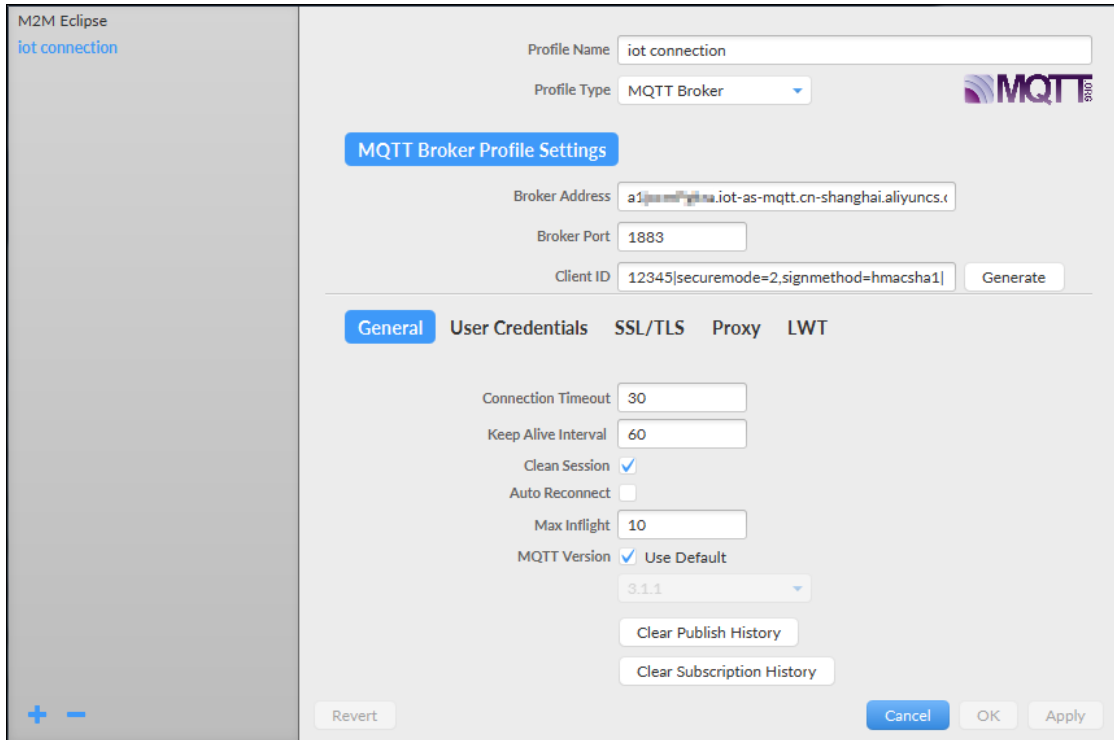
3. Set the connection parameters.

Currently, two types of connection modes are supported: TCP and TLS. These two modes only differ in settings of Client ID and SSL/TLS.

The procedure is as follows:

- i. Enter basic information. See the following table for parameter descriptions.

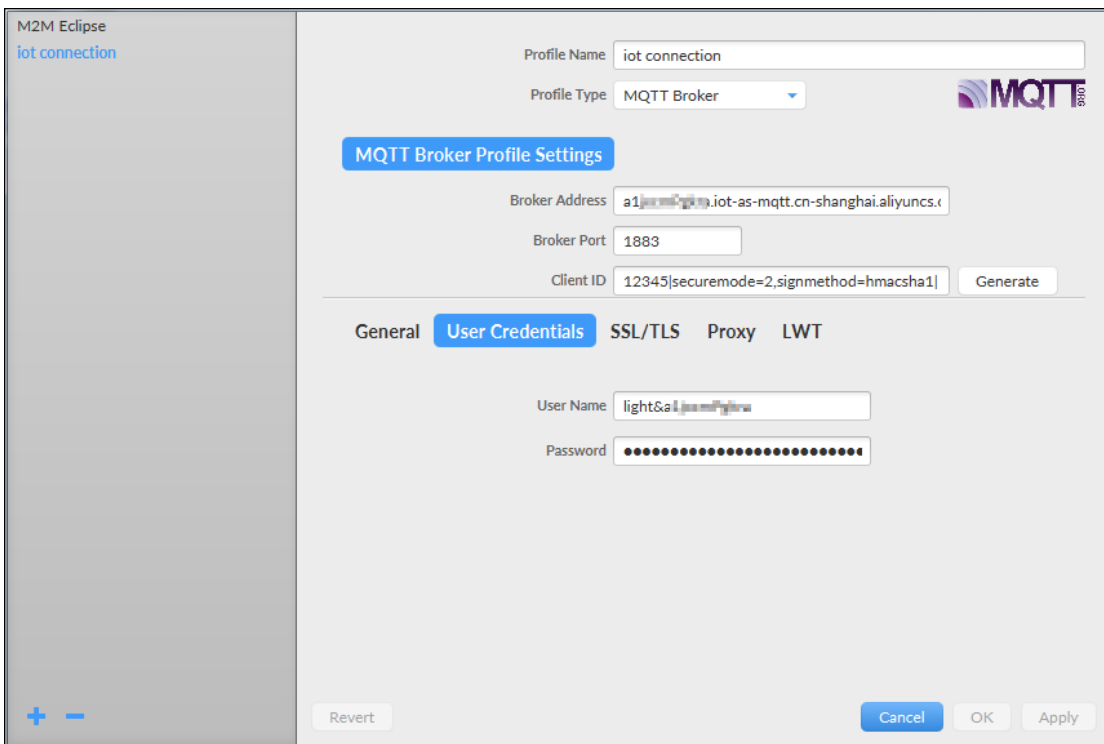
You can keep the default parameters for **General**, or set the values according to your needs.



Parameter	Description
Profile Name	Enter a custom profile name.
Profile Type	Select MQTT Broker .
Broker Address	<ul style="list-style-type: none"> To view the endpoint of the instance that you purchased, perform the following steps: Log on to the IoT Platform console. In the left-side navigation pane, click Instances. On the page that appears, click View in the Actions column of the instance. On the Instance Details page, you can view the endpoint. The endpoint for public instances is <code>\${YourProductKey}.iot-as-mqtt.\${YourRegionId}.aliyuncs.com</code>. <ul style="list-style-type: none"> <code>\${YourProductKey}</code>: Replace this variable with the ProductKey of the product to which the device belongs. You can obtain the ProductKey on the Device Details page of the IoT Platform console. <code>\${YourRegionId}</code>: Replace this variable with your region ID. For information about region IDs, see Regions and zones.
Broker Port	Set to <code>1883</code> .

Parameter	Description
Client ID	<p>Enter a value in the format of <code>`\${clientId}`securemode=3,signmethod=hmacsha1</code>. Example: <code>12345 securemode=3,signmethod=hmacsha1</code>. The parameters are described as follows:</p> <ul style="list-style-type: none"> <code>`\${clientId}`</code> is a custom client ID. It can be any value within 64 characters. We recommend that you use the MAC address or SN code of the device as the value of client ID. <code>securemode</code> is the security mode of the connection. If you use the TCP mode, set it as <code>securemode=3</code>; if you use the TLS mode, set it as <code>securemode=2</code>. <code>signmethod</code> is the signature method that you want to use. IoT Platform supports <code>hmacmd5</code> and <code>hmacsha1</code>. <p>Note Do not click Generate after you enter the Client ID information.</p>

ii. Click **User Credentials**, and enter your **User Name** and **Password**.

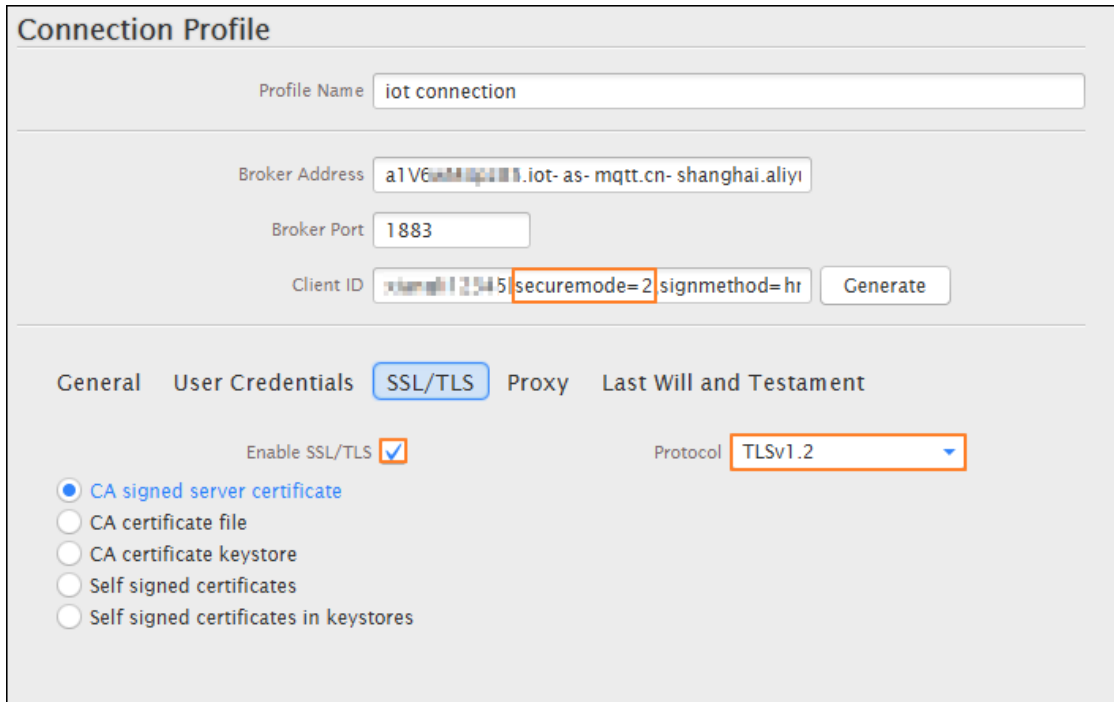


Parameter	Description
User Name	It must be the device name directly followed by the character "&" and the product key. Format: <code>`\${YourDeviceName}`&`\${YourProductKey}`</code> . For example, <code>device&fOAt5H5TOWF</code> .

Parameter	Description
Password	<p>You must enter an encrypted value of the input parameters.</p> <ul style="list-style-type: none"> IoT Platform provides a Password Generator for you to generate one easily. <p>Parameters in the password generator:</p> <ul style="list-style-type: none"> productKey: The unique identifier of the product to which the device belongs. You can view this information on the device details page in the console. deviceName: The name of the device. You can view this information on the device details page in the console. deviceSecret: The device secret. You can view this information on the device details page in the console. timestamp: (Optional) Timestamp of the current system time. clientId: The custom client ID, which must be the same as the value of <code>clientId</code> in Client ID. method: The signature algorithm, which must be the same as the value of <code>signmethod</code> in Client ID. <ul style="list-style-type: none"> You can also encrypt a password by yourself. <p>Generate a password manually:</p> <ol style="list-style-type: none"> Sort and join the parameters. <ul style="list-style-type: none"> Sort and join the parameters <code>clientId</code>, <code>deviceName</code>, <code>productKey</code>, and <code>timestamp</code> in a lexicographical order. (If you have not set a timestamp, do not include timestamp in the string.) Joint string example: <code>clientId12345deviceNamedeviceproductKeyfOAt5H5TOWF</code> Encrypt. <ul style="list-style-type: none"> Use the <code>deviceSecret</code> of the device as the secret key to encrypt the joint string by the signature algorithm defined in Client ID. Suppose the <code>deviceSecret</code> of the device is <code>abc123</code>, the encryption format is <code>hmacsha1(abc123,clientId12345deviceNamedeviceproductKeyfOAt5H5TOWF)</code>.

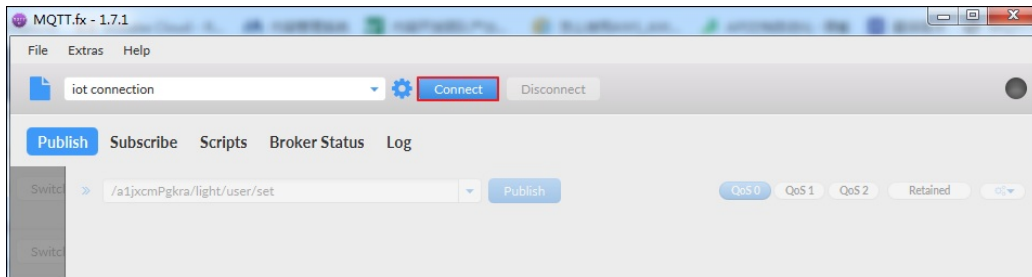
- iii. If you use TLS connection mode, you are required to set information for SSL/TLS. SSL/TLS settings are not required when the connection mode is TCP.

Check the box for **Enable SSL/TLS**, and select **TLSv1** as the protocol.



- iv. Enter all the required information, and then click OK.

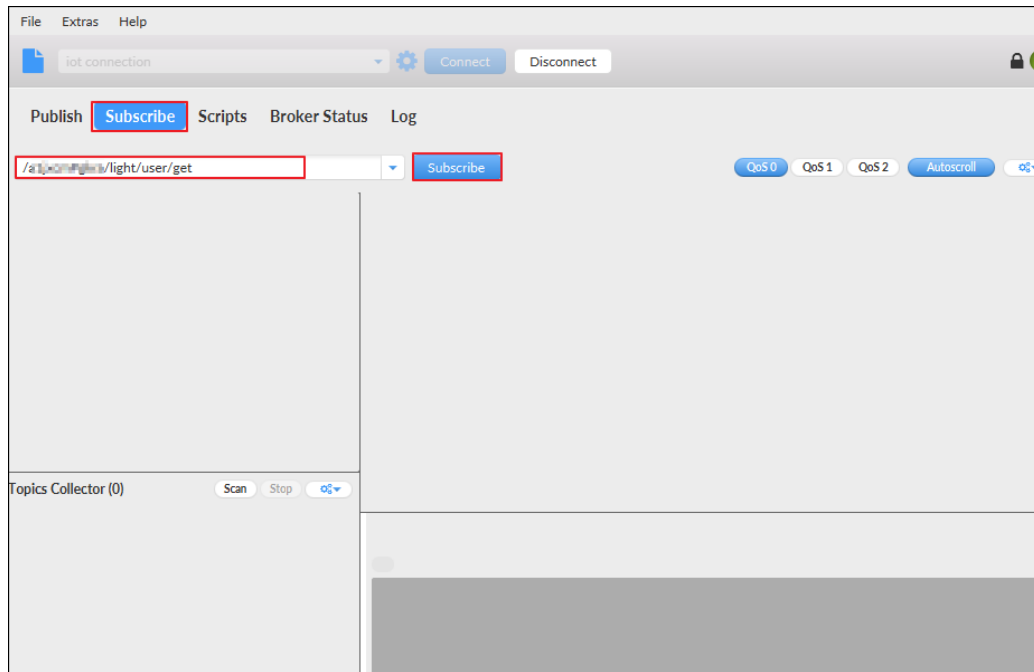
4. Click **Connect** to connect to IoT Platform.



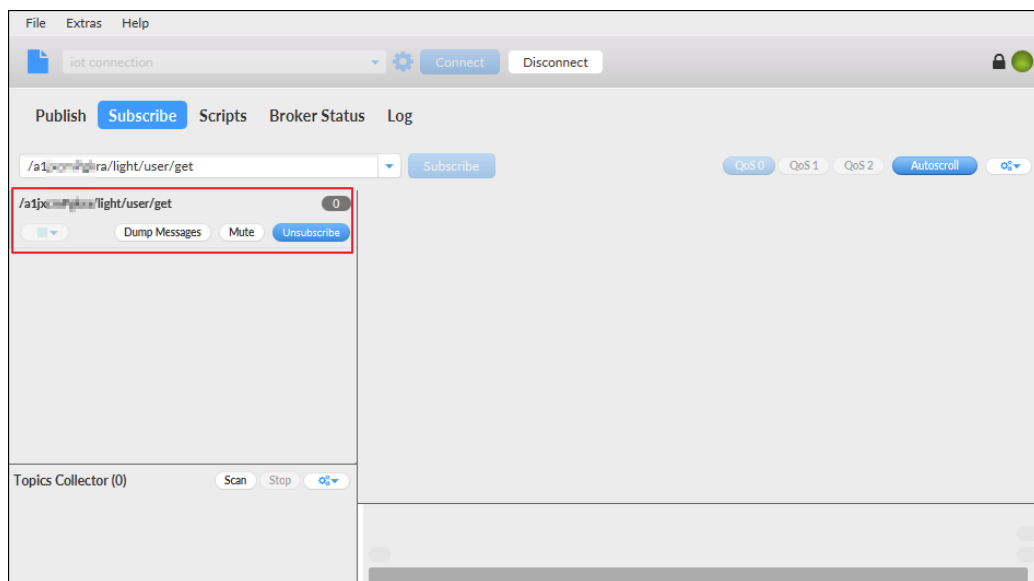
Message communication test

Test whether MQTT.fx and IoT Platform are successfully connected.

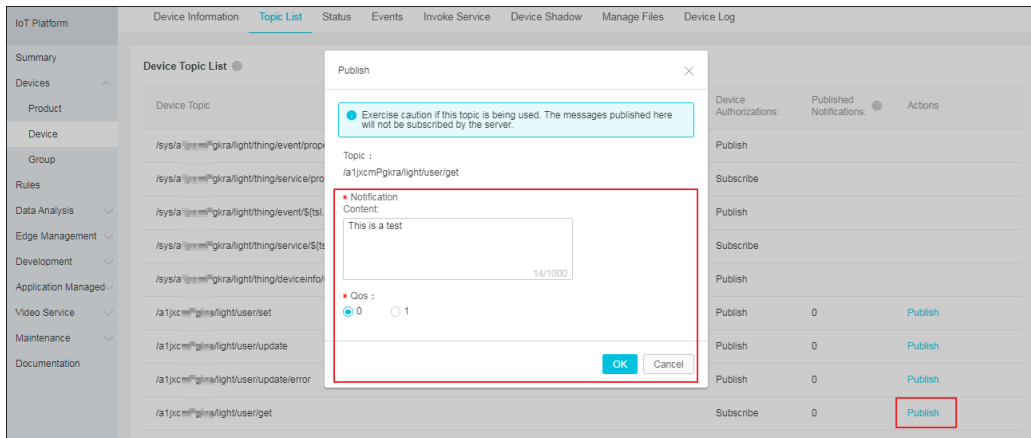
1. In MQTT.fx, click **Subscribe**.
2. Enter a topic of the device, and then click **Subscribe**.



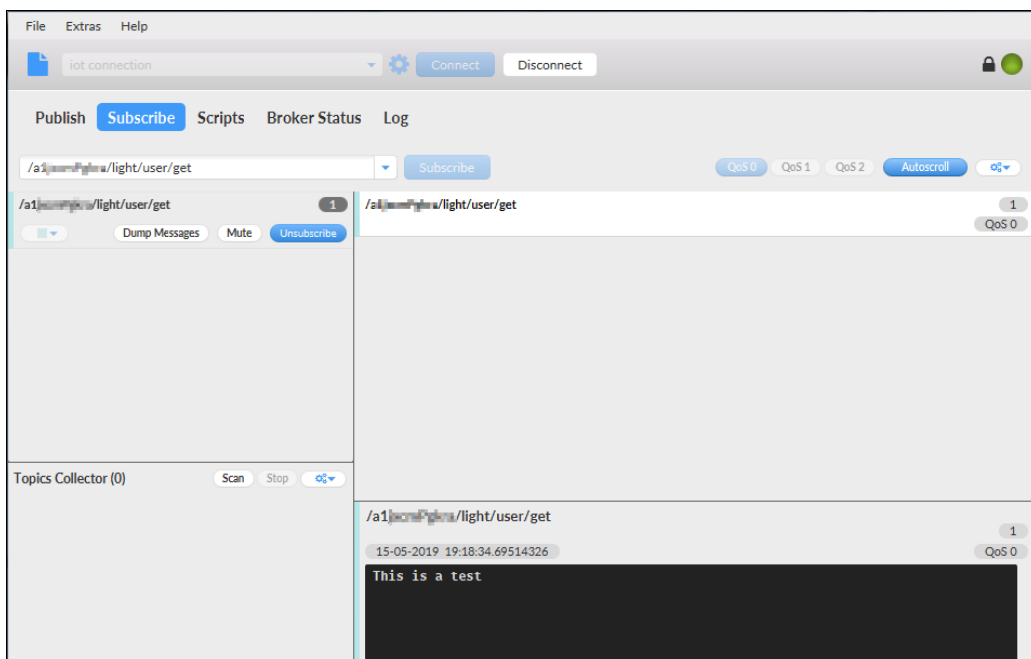
After you have successfully subscribed to a topic, it is displayed in the topic list.



3. In the **IoT Platform console**, in the **Topic List** of the **Device Details** page, click the **Publish** button of the topic that you have subscribed to.
4. Enter message content, and click **OK**.

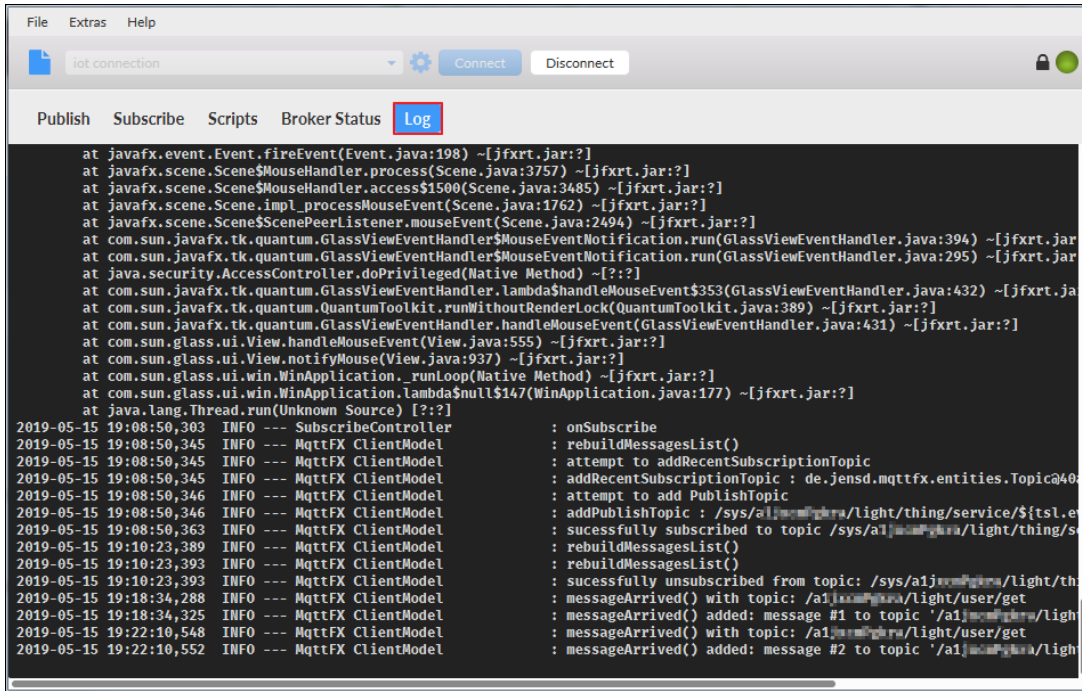


5. Go back to MQTT.fx to check if the message has been received.



View logs

In MQTT.fx, click **Log** to view the operation logs and error logs.

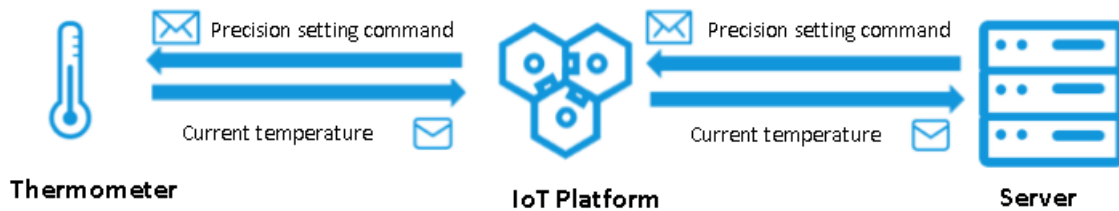


3. Use custom topics for communication

You can define custom topic categories in IoT Platform. Then, a device can send messages to a custom topic, and your server can receive the messages using an AMQP SDK. Your server can also call the API operation `Pub` to send commands to the device.

Scenario

In this example, an electronic thermometer periodically exchanges data with a server. The thermometer sends the current temperature to the server, and the server sends the precision setting command to the thermometer.



Prepare the development environment

In this example, both the devices and the server use Java SDKs, so you need to prepare the Java development environment first. You can download Java tools at [Java official website](#) and install the Java development environment.

Add the following Maven dependencies to import the device SDK (Link Kit Java SDK) and IoT SDK:

```
<dependencies>
<dependency>
  <groupId>com.aliyun.alink.linksdk</groupId>
  <artifactId>iot-linkkit-java</artifactId>
  <version>1.2.0.1</version>
  <scope>compile</scope>
</dependency>
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-core</artifactId>
  <version>3.7.1</version>
</dependency>
<dependency>
  <groupId>com.aliyun</groupId>
  <artifactId>aliyun-java-sdk-iot</artifactId>
  <version>6.9.0</version>
</dependency>
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>iot-client-message</artifactId>
  <version>1.1.2</version>
</dependency>
</dependencies>
```

Create a product and a device

First, you need to create a product, define custom topic categories, define the TSL model, configure server-side subscription, and create a device in the IoT Platform console.

1. Log on to the [IoT Platform console](#).
2. In the left-side navigation pane, choose **Devices > Products**.
3. Click **Create Product** to create a thermometer product.

For more information, see [Create a product](#).

4. After the product is created, find the product and click **View**.
5. On the **Topic Categories** tab of the **Product Details** page, add custom topic categories.

For more information, see [Customize a topic category](#).

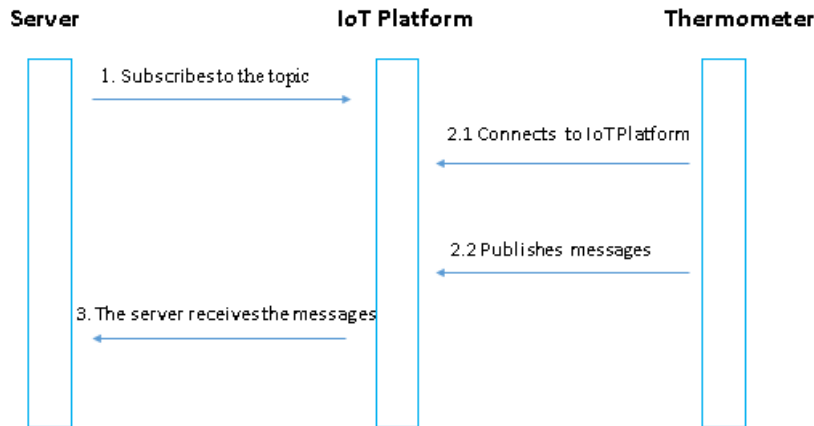
In this example, add the following two topic categories:

- `/${productKey}/${deviceName}/user/devmsg`: used by devices to publish messages. Set Device Operation Authorizations to Publish for this topic category.
- `/${productKey}/${deviceName}/user/cloudmsg`: used by devices to receive subscribed messages. Set Device Operation Authorizations to Subscribe for this topic category.

6. On the **Server-side Subscription** page, set the type of messages to be pushed to the AMQP SDK to **Device Upstream Notification**. For more information, see [Configure AMQP server-side subscriptions](#)
7. In the left-side navigation pane, choose **Devices** and add a thermometer device under the thermometer product that has been created. For more information, see [Create a device](#).

The server receives messages from the device

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



- Configure the AMQP SDK, which will be installed in the server, to receive messages from IoT Platform.
 - Add the following dependency to install [Qpid JMS 0.47.0](#).

```

<!-- amqp 1.0 qpid client -->
<dependency>
  <groupId>org.apache.qpid</groupId>
  <artifactId>qpid-jms-client</artifactId>
  <version>0.47.0</version>
</dependency>
<!-- util for base64 -->
<dependency>
  <groupId>commons-codec</groupId>
  <artifactId>commons-codec</artifactId>
  <version>1.10</version>
</dependency>
    
```

- Connect the AMQP SDK to IoT Platform.

For information about the parameters in the following demo, see [Connect an AMQP client to IoT Platform](#).

```

import java.net.URI;
import java.util.Hashtable;
import javax.crypto.Mac;
    
```

```
import javax.crypto.spec.SecretKeySpec;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.Destination;
import javax.jms.Message;
import javax.jms.MessageConsumer;
import javax.jms.MessageListener;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.naming.Context;
import javax.naming.InitialContext;
import org.apache.commons.codec.binary.Base64;
import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionFactory;
import org.apache.qpid.jms.message.JmsInboundMessageDispatch;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
public class AmqpJavaClientDemo {
    private final static Logger logger = LoggerFactory.getLogger(AmqpJavaClientDemo.class);
    public static void main(String[] args) throws Exception {
        //For information about parameters, see AMQP client access instructions.
        String accessKey = "${YourAccessKeyId}";
        String accessSecret = "${YourAccessKeySecret}";
        String consumerGroupId = "${YourConsumerGroupId}";
        long timeStamp = System.currentTimeMillis();
        //Signature method: hmacmd5, hmacsha1, or hmacsha256.
        String signMethod = "hmacsha1";
        //The value of the clientId parameter is displayed as the client ID on the consumer group status page for server-side subscription in the console.
        //We recommend that you use a unique identifier, such as the UUID, MAC address, or IP address as the clientId value. This allows you to identify clients.
        String clientId = "${YourClientId}";
        //For information about how to obtain the value of UserName, see AMQP client access instructions.
        String userName = clientId + "| authMode = aksign"
            + ",signMethod=" + signMethod
            + ",timestamp=" + timeStamp
            + ",authId=" + accessKey
            + ",consumerGroupId=" + consumerGroupId
            + "|";
        //For information about how to obtain the value of password, see AMQP client access instructions.
        String signContent = "authId =" + accessKey + "& timestamp =" + timeStamp;
```

```

String password = doSign(signContent,accessSecret, signMethod);
//Construct the connection URL based on the rules described in Qpid JMS documentation.
String connectionUrl = "failover:(amqps://${uid}.iot-amqp.${regionId}.aliyuncs.com:5671? amqp.idleTimeout=80000)"
    + "? failover.reconnectDelay=30";
Hashtable<String, String> hashtable = new Hashtable<>();
hashtable.put("connectionfactory.SBCF",connectionUrl);
hashtable.put("queue.QUEUE", "default");
hashtable.put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");
Context context = new InitialContext(hashtable);
ConnectionFactory cf = (ConnectionFactory)context.lookup("SBCF");
Destination queue = (Destination)context.lookup("QUEUE");
// Create Connection
Connection connection = cf.createConnection(userName, password);
((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
// Create Session
// Session. Client_acknowledgment: After you receive the message, manually call message.acknowledge().
// Session. Auto_acknowledgment: The SDK is automatically acknowledged. (recommended)
Session session = connection.createSession (false, Session.Auto_acknowledgment );
connection.start();
// Create Receiver Link
MessageConsumer consumer = session.createConsumer(queue);
consumer.setMessageListener(messageListener);
}
private static MessageListener messageListener = new MessageListener() {
    @Override
    public void onMessage(Message message) {
        try {
            byte[] body = message.getBody(byte[].class);
            String content = new String(bytes);
            String topic = message.getStringProperty("topic");
            String messageId = message.getStringProperty("messageId");
            logger.info("receive message"
                + ", topic = " + topic
                + ", messageId = " + messageId
                + ", content = " + content);
            //If you select Session.CLIENT_ACKNOWLEDGE when creating a session, manual acknowledgment is required.

```

```
//message.acknowledge();
//Ensure that no time-consuming logic exists during message processing. If processing received
messages takes a long period of time, initiate asynchronous requests.
} catch (Exception e) {
    e.printStackTrace();
}
}
};
private static JmsConnectionListener myJmsConnectionListener = new JmsConnectionListener() {
    /**
     * The connection is successfully established.
     */
    @Override
    public void onConnectionEstablished(Uri remoteUri) {
        logger.info("onConnectionEstablished, remoteUri:{}", remoteUri);
    }
    /**
     * The connection fails after the maximum number of retries.
     */
    @Override
    public void onConnectionFailure(Throwable error) {
        logger.error("onConnectionFailure, {}", error.getMessage());
    }
    /**
     * The connection is interrupted.
     */
    @Override
    public void onConnectionInterrupted(Uri remoteUri) {
        logger.info("onConnectionInterrupted, remoteUri:{}", remoteUri);
    }
    /**
     * The connection is interrupted and automatically restored.
     */
    @Override
    public void onConnectionRestored(Uri remoteUri) {
        logger.info("onConnectionRestored, remoteUri:{}", remoteUri);
    }
    @Override
    public void onInboundMessage(JmsInboundMessageDispatch envelope) {}
    @Override
    public void onSessionClosed(Session session, Throwable cause) {}
}
```

```
@Override
public void onConsumerClosed(MessageConsumer consumer, Throwable cause) {}
@Override
public void onProducerClosed(MessageProducer producer, Throwable cause) {}
};
/**
 * To obtain the string to sign in the value of password, see AMQP client access instructions.
 */
private static String doSign(String toSignString, String secret, String signMethod) throws Exception {
    SecretKeySpec signingKey = new SecretKeySpec(secret.getBytes(), signMethod);
    Mac mac = Mac.getInstance(signMethod);
    mac.init(signingKey);
    Byte [] rawHmac = mac.doFinal (toSignString. getBytes ());
    return Base64.encodeBase64String(rawHmac);
}
}
```

- Configure the device SDK to send a message.
 - Configure device authentication information.

```
final String productKey = "XXXXXX";
final String deviceName = "XXXXXX";
final String deviceSecret = "XXXXXXXXXX";
final String region = "XXXXXX";
```

- Set connection initialization parameters, including MQTT connection information, device information, and initial device status.

```
LinkKitInitParams params = new LinkKitInitParams();
// Configure MQTT connection information. Link Kit 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";
// Configure device information.
DeviceInfo deviceInfo = new DeviceInfo();
deviceInfo.productKey = productKey;
deviceInfo.deviceName = deviceName;
deviceInfo.deviceSecret = deviceSecret;
// Register the initial device status.
Map<String, ValueWrapper> propertyValues = new HashMap<String, ValueWrapper>();
params.mqttClientConfig = config;
params.deviceInfo = deviceInfo;
params.propertyValues = propertyValues;
```

- Initialize the connection.

```
// Initialize the connection and set the callback that is called when the initialization is successful.
LinkKit.getInstance().init(params, new ILinkKitConnectListener() {
    @Override
    public void onError(AError aError) {
        System.out.println("Init error:" + aError);
    }
    // Set the callback that is called when the initialization is successful.
    @Override
    public void onInitDone(InitResult initResult) {
        System.out.println("Init done:" + initResult);
    }
});
```


- Send a message from the device.

After connecting to IoT Platform, the device sends a message to the specified topic. Replace the content of the `onInitDone` callback like the following example:

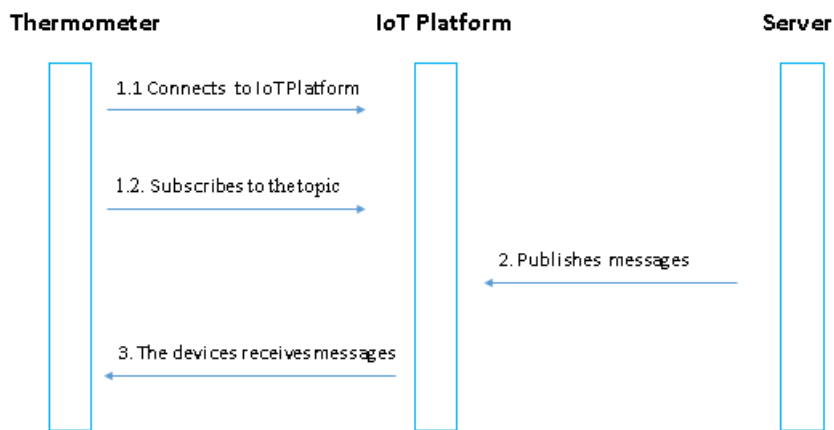
```
@Override
public void onInitDone(InitResult initResult) {
    // Set the topic to which the message is published and the message content.
    MqttPublishRequest request = new MqttPublishRequest();
    request.topic = "/" + productKey + "/" + deviceName + "/user/devmsg";
    request.qos = 0;
    request.payloadObj = "{\"temperature\":35.0, \"time\":\"sometime\"}";
    // Publish the message and set the callback that is called when the 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}
```

The server sends messages 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 configure device authentication information, set connection initialization parameters, and initialize the connection, see the device SDK configuration in the previous section.

The device needs to subscribe to a specific topic to receive messages sent by the server.

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

```
// Set the callback that is called when the initialization is successful.
@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 set the callbacks that are called when the subscription succeeds and
    // fails, respectively.
    LinkKit.getInstance().subscribe(request, new IConnectSubscribeListener() {
        @Override
        public void onSuccess() {
            System.out.println("");
        }
        @Override
        public void onFailure(AError aError) {
        }
    });
    // Set the listener for listening to subscribed messages.
    IConnectNotifyListener notifyListener = new IConnectNotifyListener() {
        // Define the callback that is called when a subscribed message is received.
        @Override
        public void onNotify(String connectId, String topic, AMessage aMessage) {
            System.out.println(
                "received message from " + topic + ":" + new String((byte[])aMessage.getData());
        }
        @Override
        public boolean shouldHandle(String s, String s1) {
            return false;
        }
        @Override
        public void onConnectStateChange(String s, ConnectState connectState) {
        }
    };
    LinkKit.getInstance().registerOnNotifyListener(notifyListener);
}
```

- Configure the IoT SDK to call the **Pub** operation to publish a message.

- Configure identity verification information.

```
String regionId = "The ID of the region where the device is located";
String accessKey = "The AccessKey ID of your Alibaba Cloud account";
String accessSecret = "The AccessKey Secret of your Alibaba Cloud account";
final String productKey = "The key of the product";
```

- Set connection parameters.

```
// Construct a client.
DefaultProfile profile = DefaultProfile.getProfile(regionId, accessKey, accessSecret);
IAcsClient client = new DefaultAcsClient(profile);
```

- Set the parameters for publishing 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 message content. The message content must be encoded in Base64. Otherwise, the message
content will be 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: demo

Click [here](#) to download and view the complete demo for this example.

For more information about server-side subscription, see:

[Configure AMQP server-side subscriptions](#)

[Connect an AMQP client to IoT Platform](#)

[Java SDK access example](#)

[Node.js SDK access example](#)

[.NET SDK access example](#)

[Python 2.7 SDK access example](#)