IoT Device Access

Service Overview

Issue	1.0
Date	2024-11-06

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions

HUAWEI and other Huawei trademarks are the property of Huawei Technologies Co., Ltd.
All other trademarks and trade names mentioned in this document are the property of their respective holders.

Notice

The purchased products, services and features are stipulated by the contract made between Huawei Cloud and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

Contents

1 Quick Device Access - Property Reporting and Command Receiving.....1
1.1 Subscribing to IoTDA.....1
1.2 Connecting a Smart Smoke Detector to the Platform (Quick Usage).....2
1.3 Registering a Simulated Smart Street Light Device.....6
1.4 Using MQTT.fx to Simulate Communication Between the Smart Street Light and the Platform.....15
1.5 Using a Virtual Smart Street Light to Communicate with the Platform (Java SDK).....22
1.6 Using a Virtual Smart Street Light to Communicate with the Platform (C SDK).....34
2 Quick Device Access - Message Sending and Receiving.....45
2.1 Overview.....45
2.2 Subscribing to IoTDA.....47
2.3 Registering a Device.....48
2.4 Using MQTT.fx to Send and Receive Messages.....50
2.5 Using Device SDKs to Send and Receive Messages.....56
3 Quick Application Access.....60

1 Quick Device Access - Property Reporting and Command Receiving

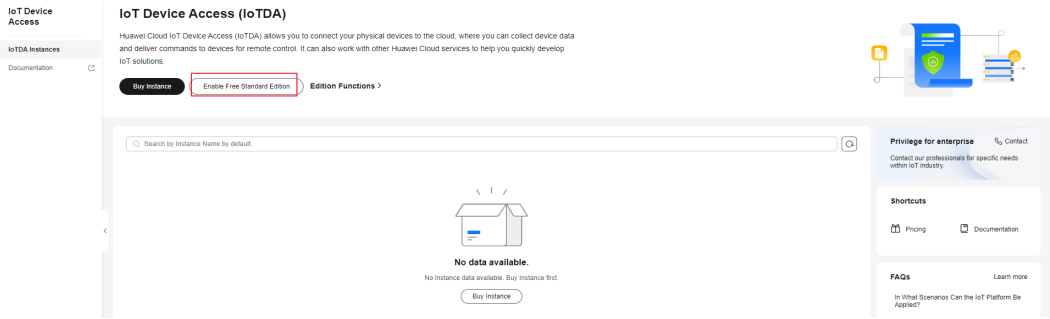
1.1 Subscribing to IoTDA

This section describes how to subscribe to a free IoTDA instance of the standard edition in the CN-Hong Kong region.

Step 1 Access the **IoTDA** service page and click **Access Console**.

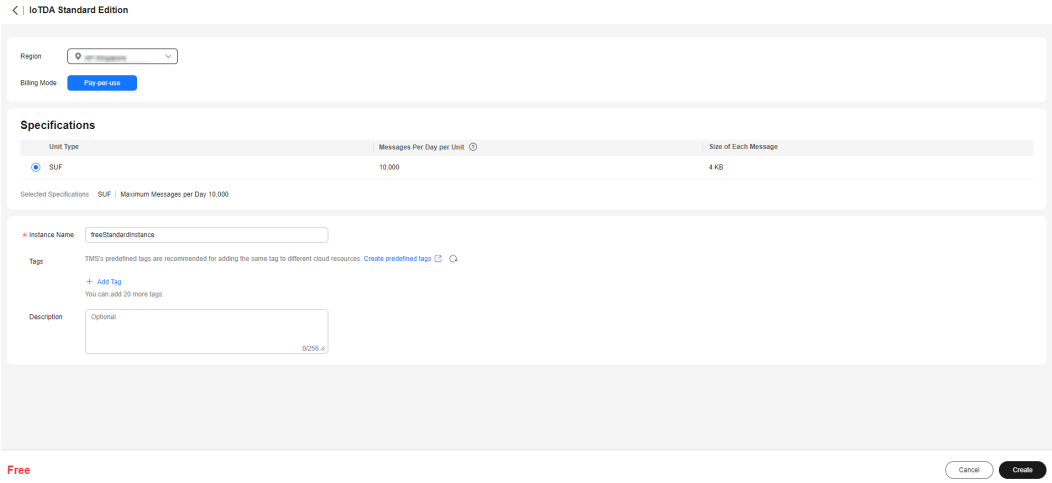
Step 2 In the navigation pane, choose **IoTDA Instances** and click the button for subscribing to a free instance.

Figure 1-1 Standard edition - Enabling free instances



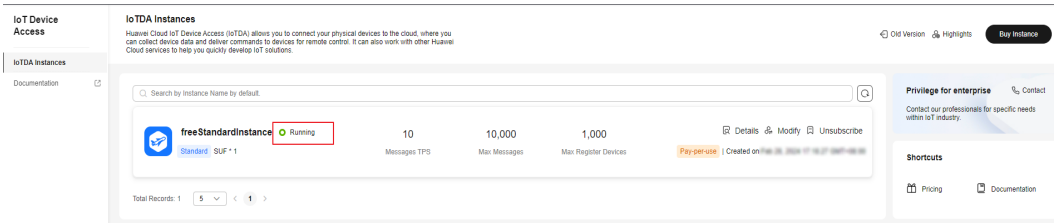
Step 3 Configure as required (recommended: default value).

Figure 1-2 Instance - Configuring a free instance



Step 4 Click **Create**, and the instance page is displayed. Refresh the page and wait until the instance status changes to **Running**, indicating that the free instance has been created.

Figure 1-3 Instance - Free instance created



----End

1.2 Connecting a Smart Smoke Detector to the Platform (Quick Usage)

Scenarios

You can use device data collection and command receiving with a Windows or Linux PC as a virtual device. This section uses a virtual smart smoke detector as an example to describe how to connect a device to the platform, enable the device to report data, and deliver commands from the platform to the device.

Prerequisites

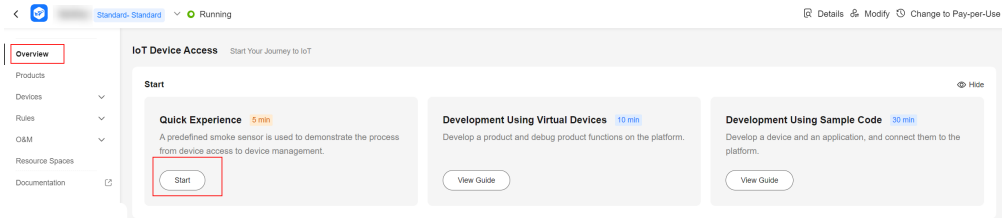
- You have registered a Huawei Cloud account. If you have not registered, click **here** to complete the registration.
- You have subscribed to the IoTDA service. If you have not subscribed to the service, go to the **IoTDA** service page, and click **Access Console** to subscribe to the service.

Procedure

Step 1 Access the **IoTDA** service page and click **Access Console**. Click the target instance card.

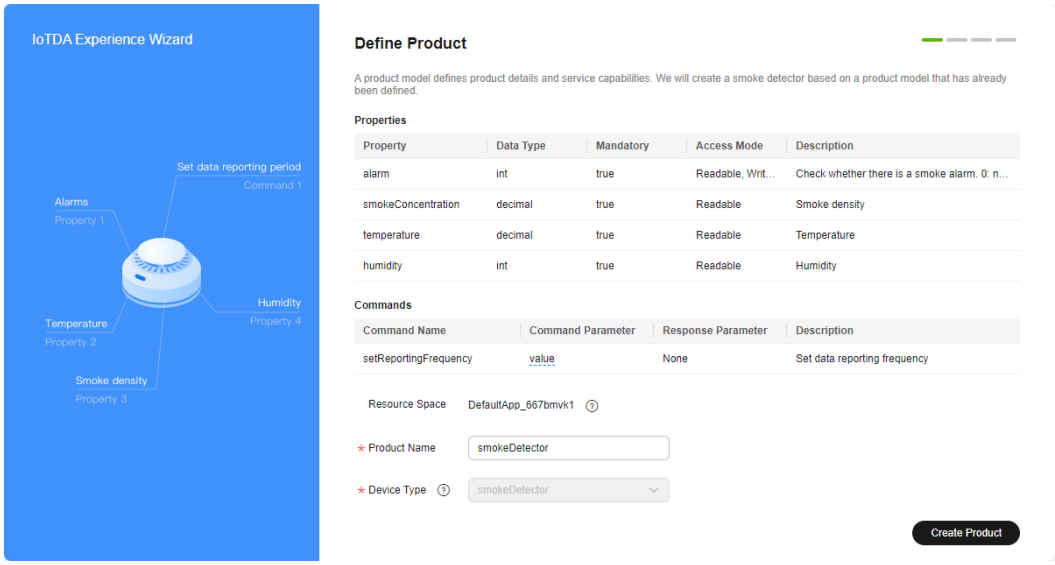
Step 2 In the navigation pane, choose **Overview** and click **Start** to start the experience.

Figure 1-4 Start



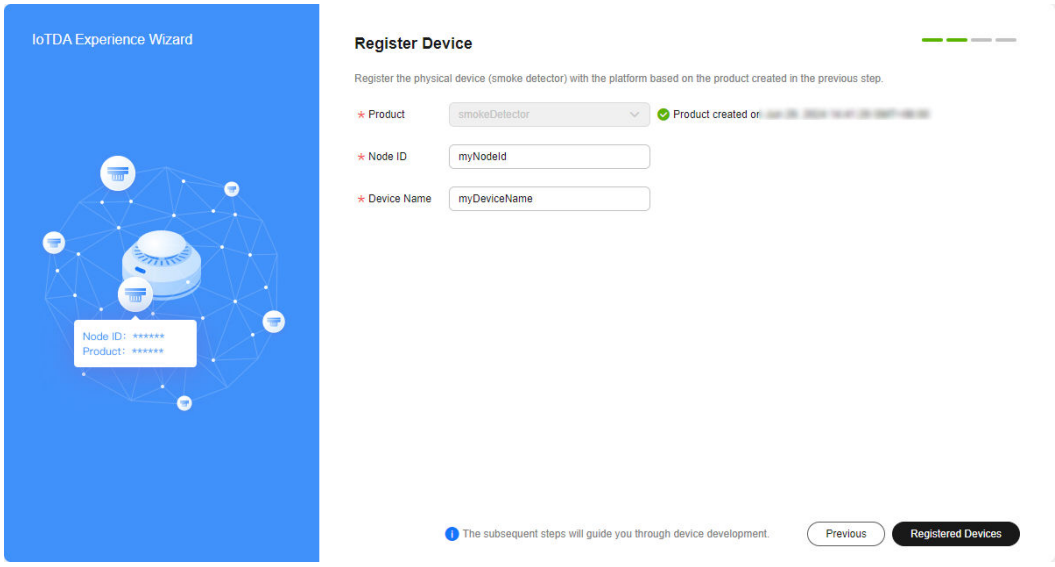
Step 3 A smart smoke detector model has been predefined for the quick usage. On the displayed page, check the properties and commands of the model and click **Create Product**.

Figure 1-5 Quick experience - Creating a product



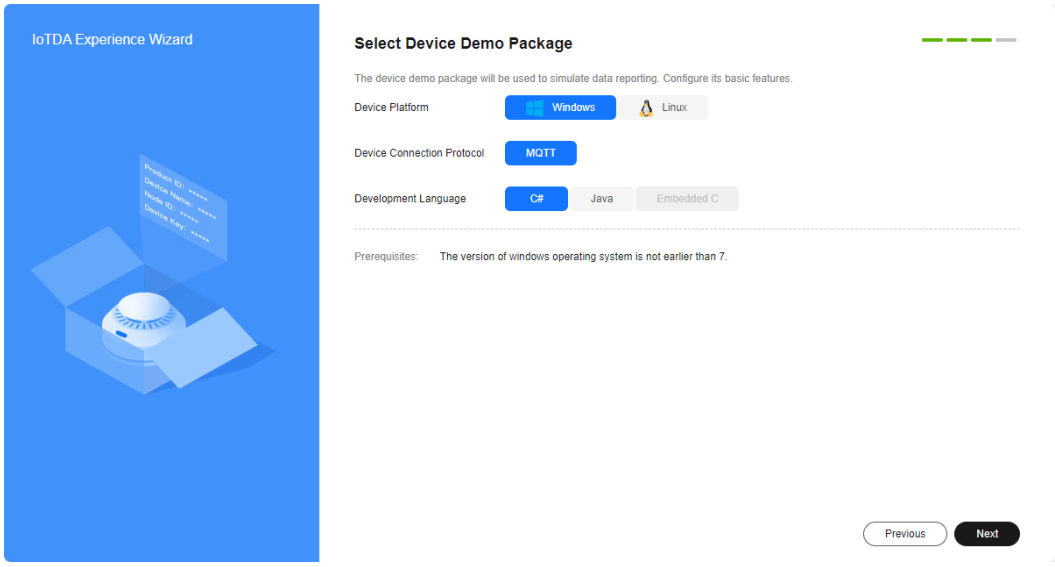
Step 4 Create a virtual smart smoke detector. Customize the node ID and device name. Click **Register Device**.

Figure 1-6 Quick experience - Registering a device



Step 5 Select a device demo package as required.

Figure 1-7 Quick experience - Selecting a device demonstration package



Step 6 Click **Download Demo Package** and decompress the package as prompted. Run the **huaweicloud-iot-device-quickstart.exe** command. If the device status changes from **Inactive** to **Online** and properties such as the temperature have reported values, the device has been connected to the platform.

Figure 1-8 Quick experience - Configuring the simulated device

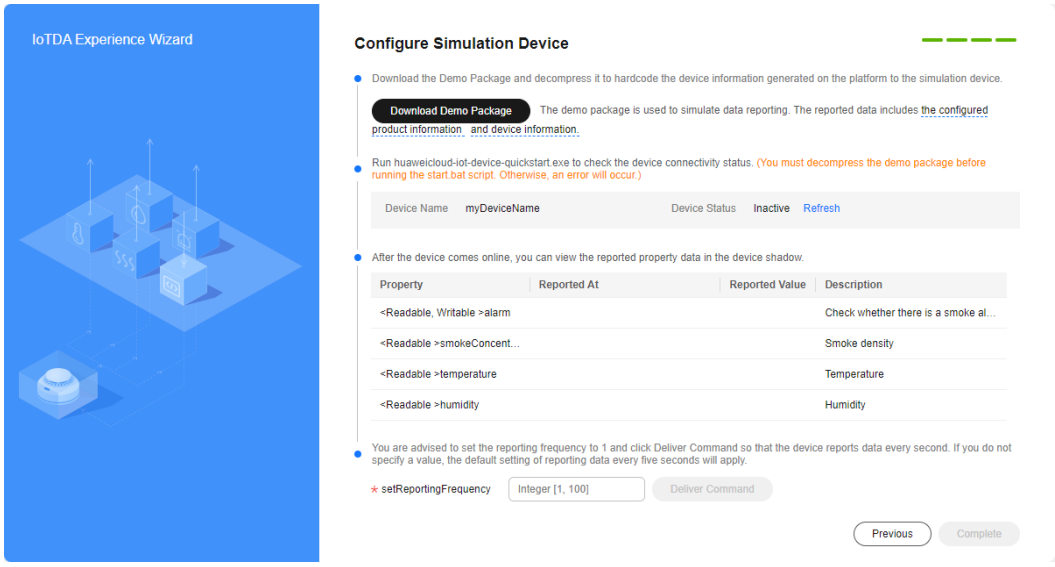
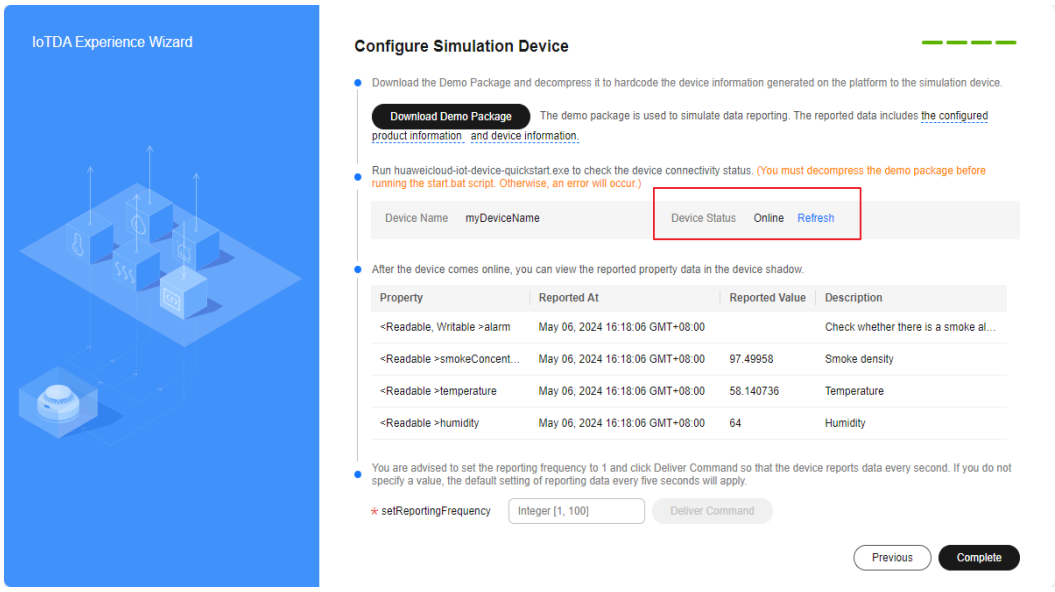
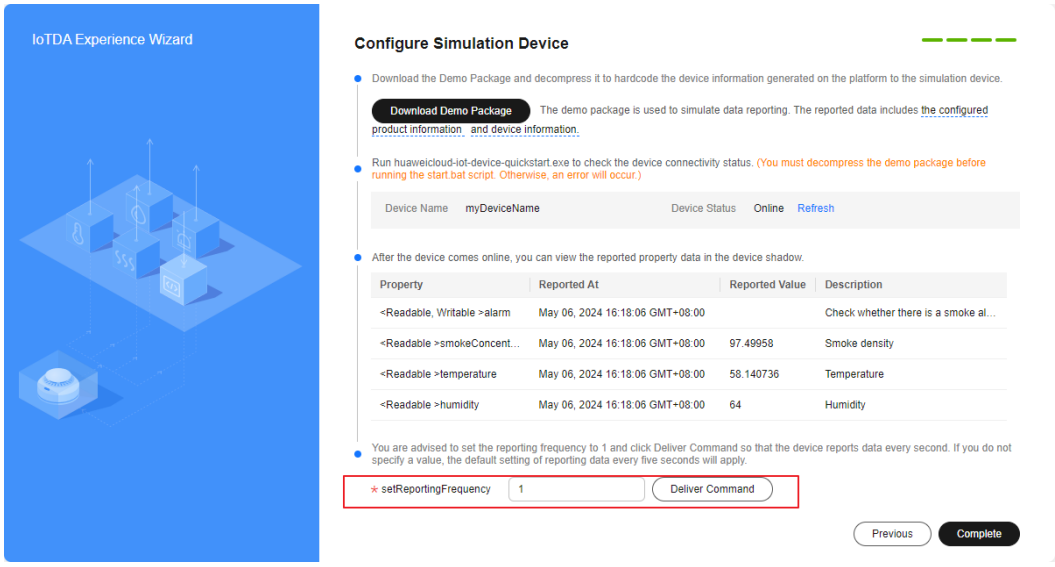


Figure 1-9 Quick experience - Simulating device status changes



Step 7 Set **setReportingFrequency** (property reporting frequency). Click **Deliver Command** to deliver the new reporting frequency to the device. Check the update speed of the reported property values before and after the setting.

Figure 1-10 Quick experience - Setting device property reporting frequency



----End

Advanced Experience

By following the instructions in this section, you may have understood how to connect a device to the platform and related concepts.

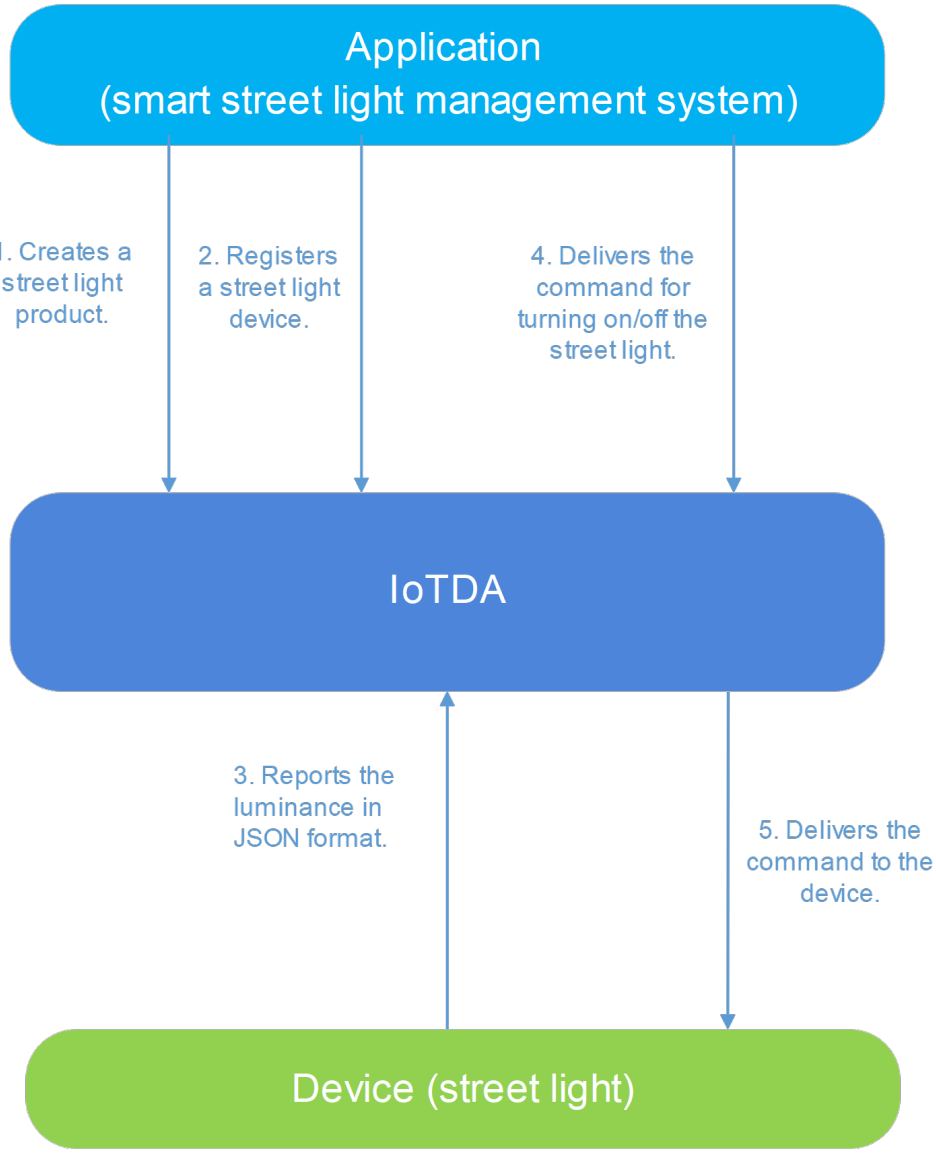
You can customize a product and use a virtual device and simulated application to use the basic functions and perform development on the platform. For details, see **Registering a Simulated Smart Street Light Device**.

1.3 Registering a Simulated Smart Street Light Device

Scenarios

This section describes how to use the MQTT.fx tool to simulate a smart street light. You can connect the simulated light to IoTDA, report the light intensity data to the platform, and deliver a light-on command to the light.

Figure 1-11 Communications between a simulated smart street light and the platform



Prerequisites

- You have registered a Huawei Cloud account. If you have not registered, click **here** to complete the registration.
- You have subscribed to the IoTDA service. If you have not subscribed to the service, go to the **IoTDA** service page, and click **Access Console** to subscribe to the service.

Service Process

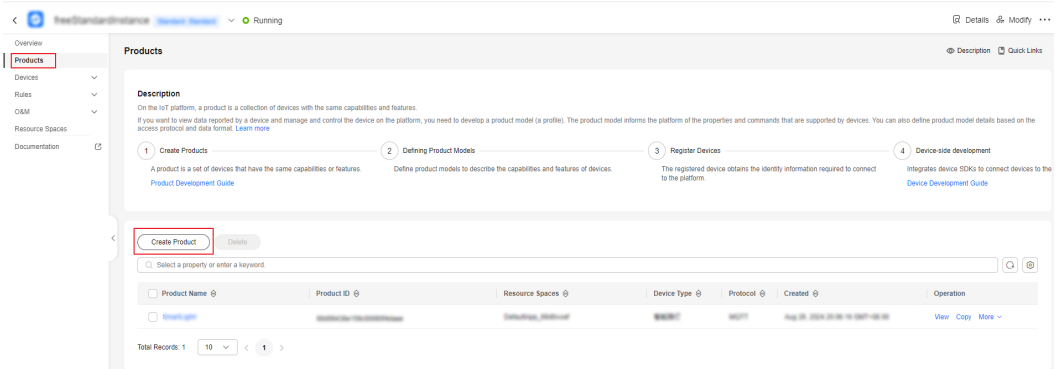
The MQTT.fx simulator is used as an example to describe data reporting and command delivery. Download **MQTT.fx** (64-bit OS) or **MQTT.fx** (32-bit OS) and install it. Overall service process:

1. Product creation: Create an MQTT smart street light product on the console. Define a product model to develop a street light that supports light intensity reporting and status control commands.
2. Device registration: Register an MQTT smart street light on the console.
3. Device connection establishment: Use MQTT.fx to simulate a smart street light, complete connection authentication, and activate the device registered on IoTDA.
4. Data reporting: Use MQTT.fx to simulate a smart street light to report light intensity data to IoTDA.
5. Command delivery: Deliver a street light switch command on the console to remotely control MQTT.fx to simulate a smart street light.

Creating a Product

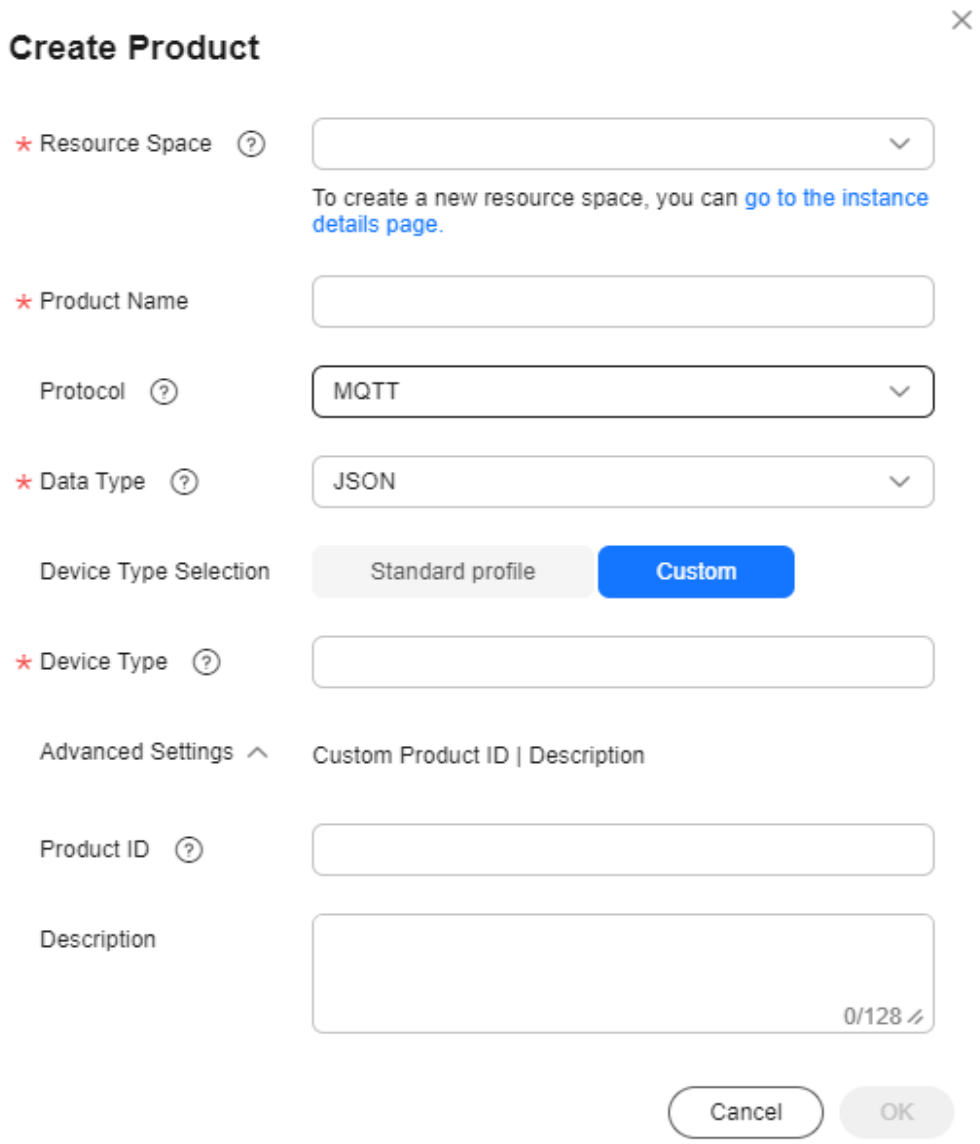
Step 1 Log in to the **console**, choose **Products** in the navigation pane, and click **Create Product** on the left.

Figure 1-12 Creating a product



Step 2 Create a product whose protocol type is MQTT and device type is **StreetLamp**, set parameters as prompted, and click **OK**.

Figure 1-13 Creating a product - MQTT



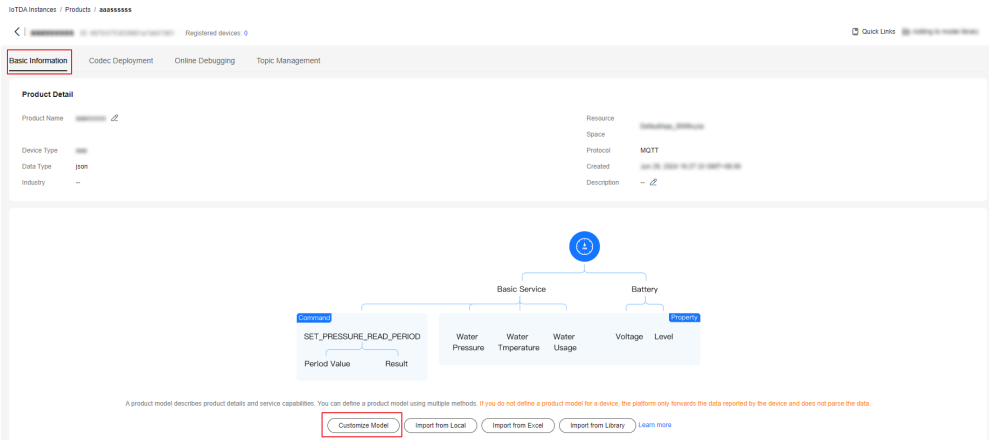
----End

Developing a Product Model

Step 1 Click the created product. The product details page is displayed.

Step 2 On the **Model Definition** tab page, click **Customize Model** to add services of the product.

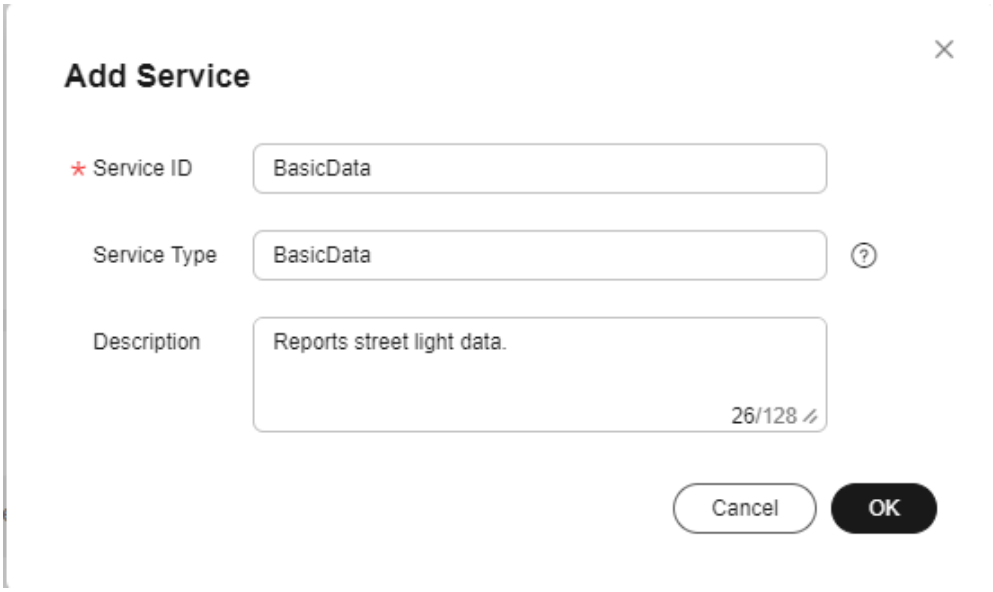
Figure 1-14 Custom model - MQTT



Step 3 Add the **BasicData** service.

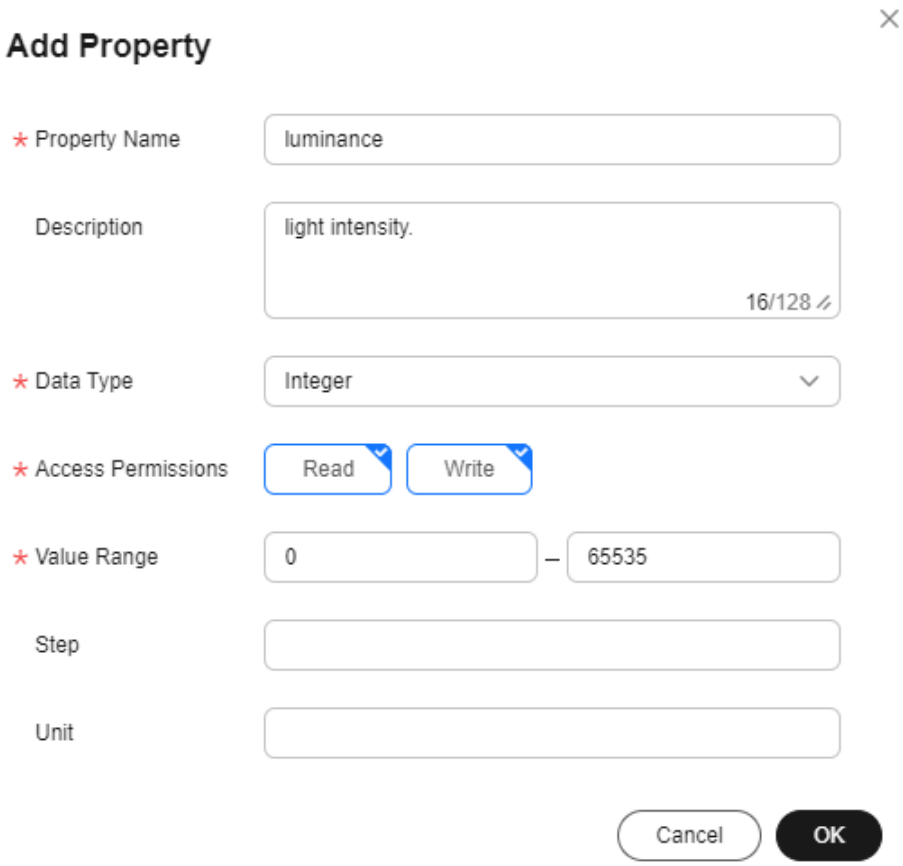
1. On the **Add Service** page, specify **Service ID**, **Service Type**, and **Description**, and click **OK**.

   Figure 1-15 Adding a service - BasicData



2. In the **BasicData** service list on the right, click **Add Property**, enter related information, and click **OK**.

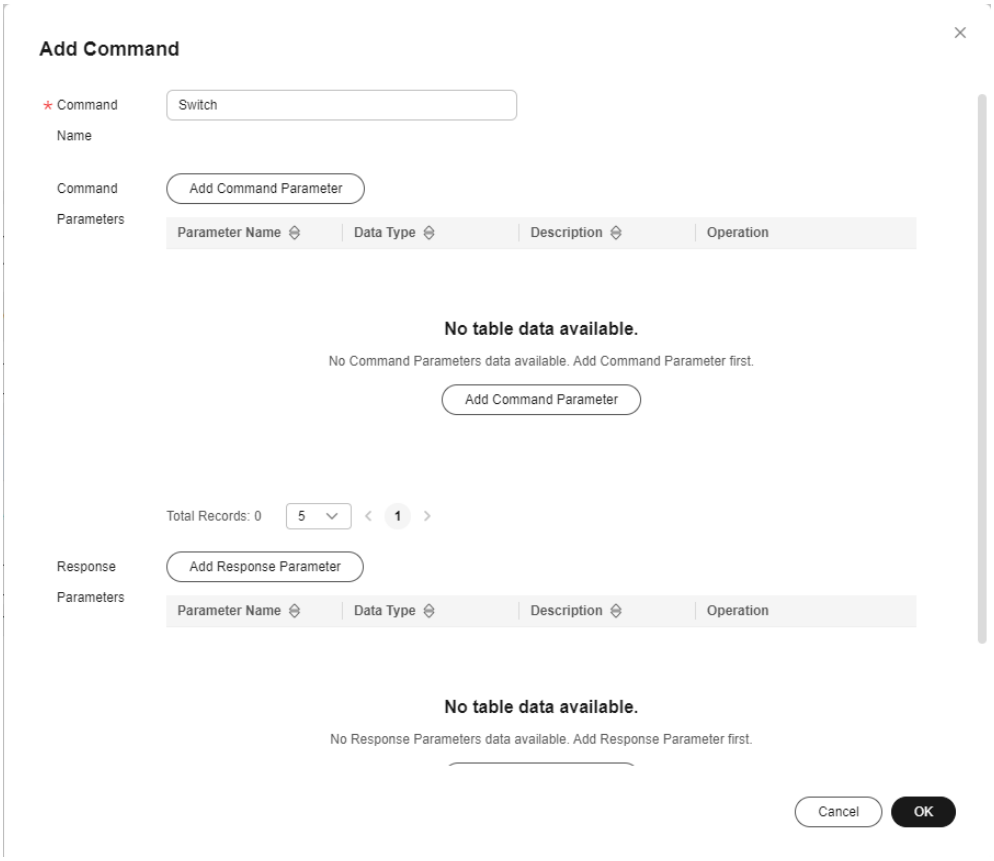
Figure 1-16 Adding a property - luminance



Step 4 Add the **LightControl** service.

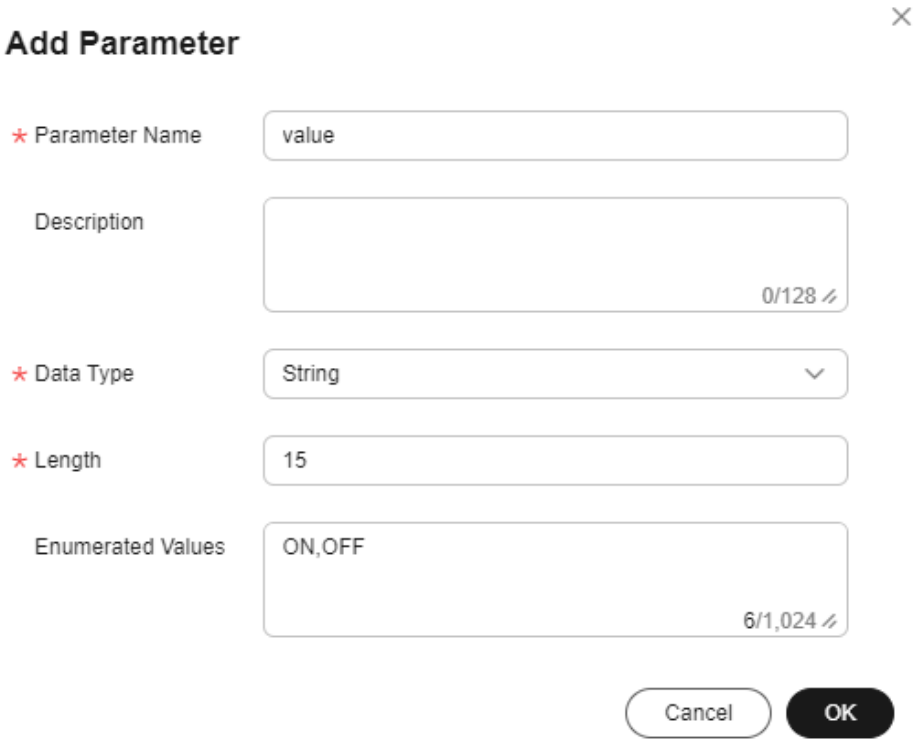
1. Click **Add Service** on the **Model Definition** tab page, set parameters as prompted, and click **OK**.
   - **Service ID**: Enter **LightControl**.
   - **Service Type**: You are advised to set this parameter to the same value as **Service ID**.
   - **Description**: Enter **Controls the street light**.
2. Choose **LightControl**, click **Add Command**, and enter the command name **Switch**.

Figure 1-17 Adding a command - Switch



3. Click **Add Command Parameter**, enter related information, and click **OK**.

Figure 1-18 Adding a command parameter - -value

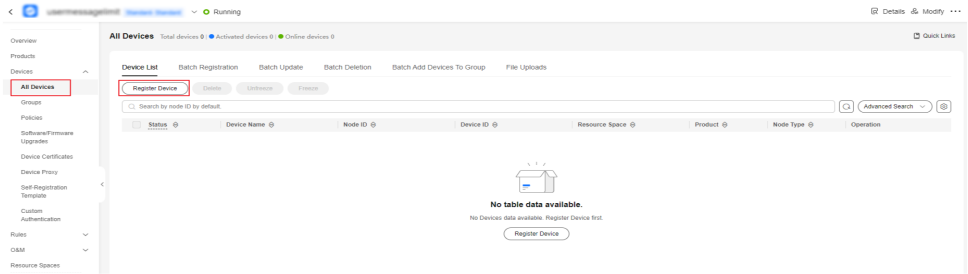


----End

Registering a Device

Step 1 On the IoTDA console, choose **Devices** > **All Devices** in the navigation pane, and click **Register Device** in the upper right corner.

Figure 1-19 Registering a device

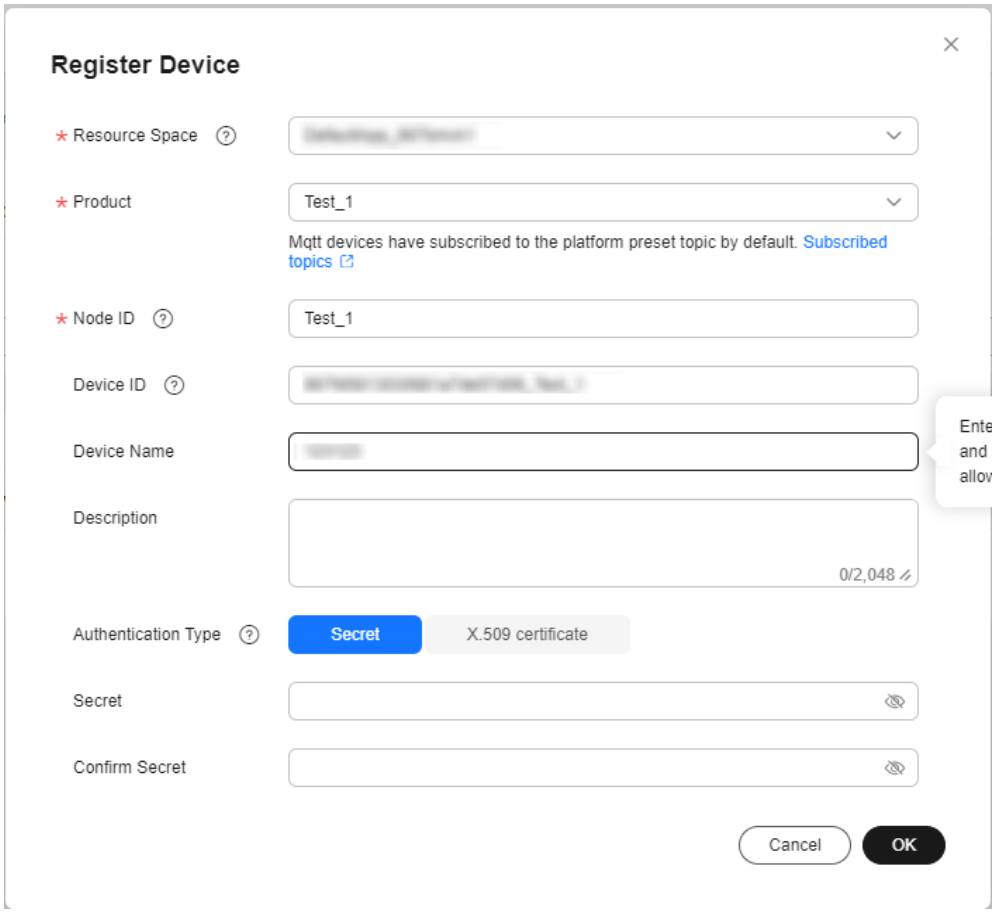


Step 2 Set the parameters as prompted and click **OK**.

Parameter	Description
Resource Space	Ensure that the device and its associated product belong to the same resource space.
Product	Select a corresponding product.

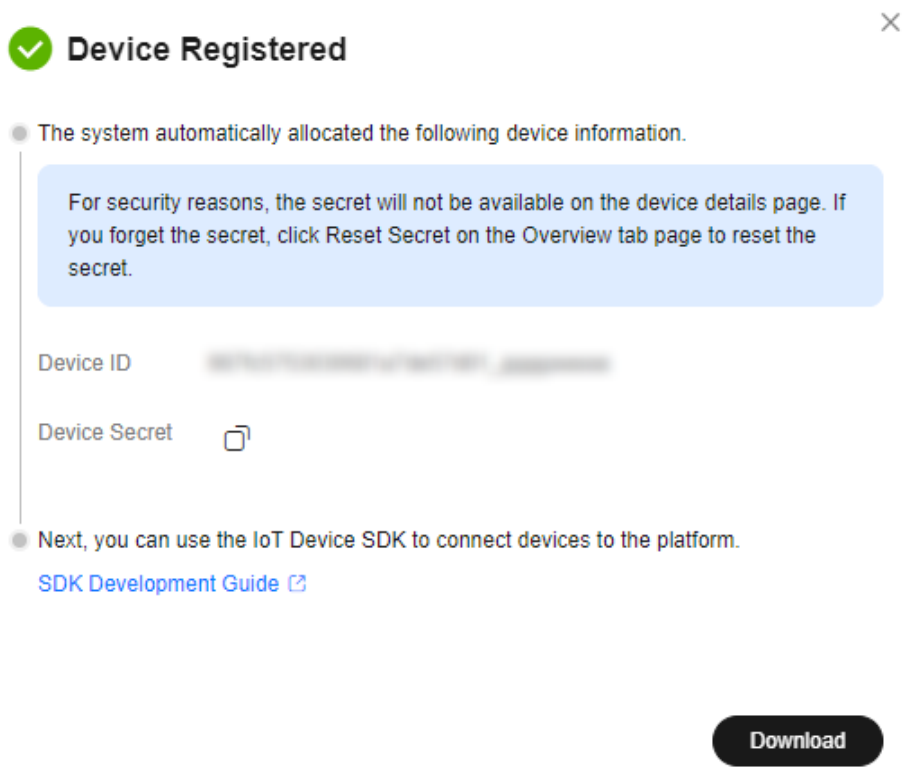
Parameter	Description
Node ID	Customize a unique physical identifier for the device. The value consists of letters and digits.
Device Name	Customize the device name.
Authentication Type	Select **Secret**.
Secret	If you do not set this parameter, IoTDA automatically generates a value.

Figure 1-20 Registering a device - MQTT



Step 3 After the device is registered, the platform automatically generates a device ID and secret. Save the device ID and secret for device access.

Figure 1-21 Device registered



----End

1.4 Using MQTT.fx to Simulate Communication Between the Smart Street Light and the Platform

Connecting the Simulated Street Light to the Platform

Use MQTT.fx to activate the device registered on IoTDA.

Step 1 Download **MQTT.fx** (64-bit OS) or **MQTT.fx** (32-bit OS) and install it.

Step 2 Go to the device details page, find **MQTT Connection Parameter**, and click **View** to check the clientId, username, password, and hostname.

Figure 1-22 Device - Device details

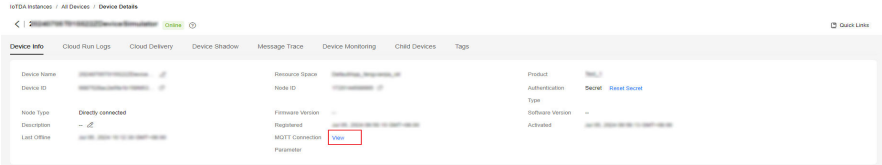
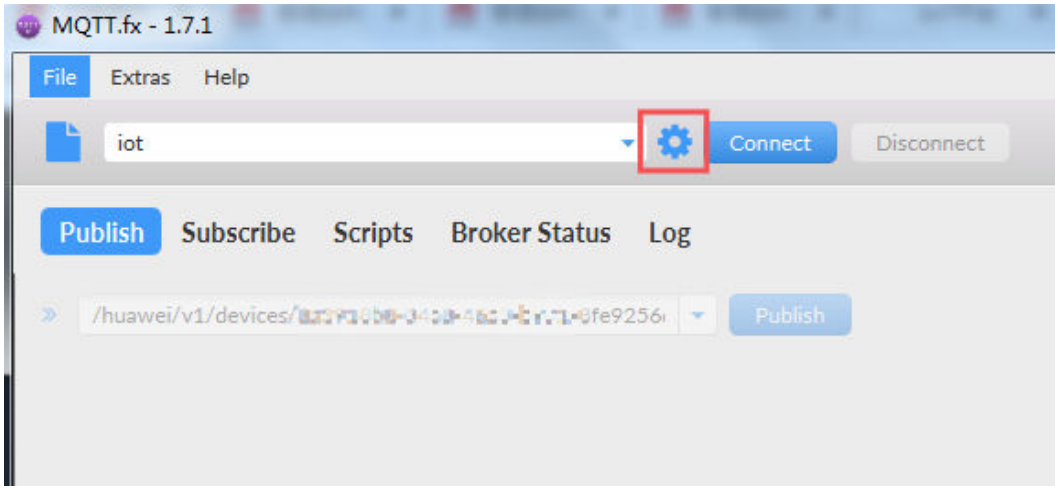


Figure 1-23 Device - Device details - MQTT connection parameters

Step 3 Open MQTT.fx and click the setting icon.

Figure 1-24 MQTT.fx Settings



Step 4 Click the **User Credentials** tab and set authentication parameters by referring to the following table.

Figure 1-25 Configuring authentication parameters

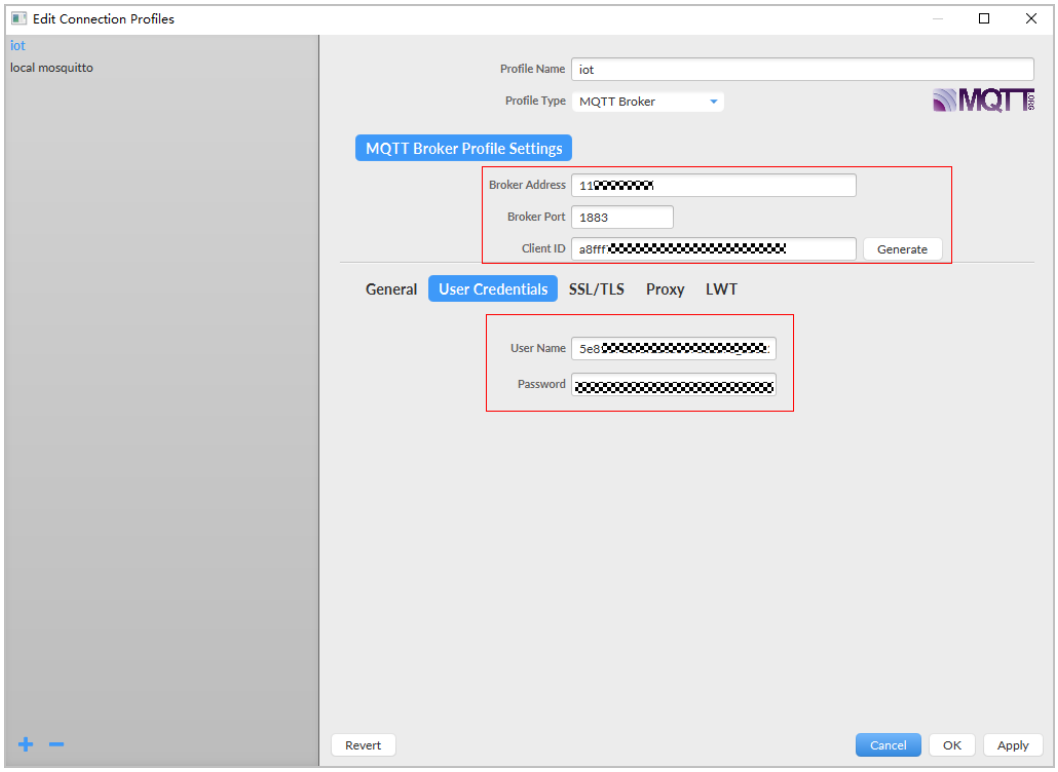
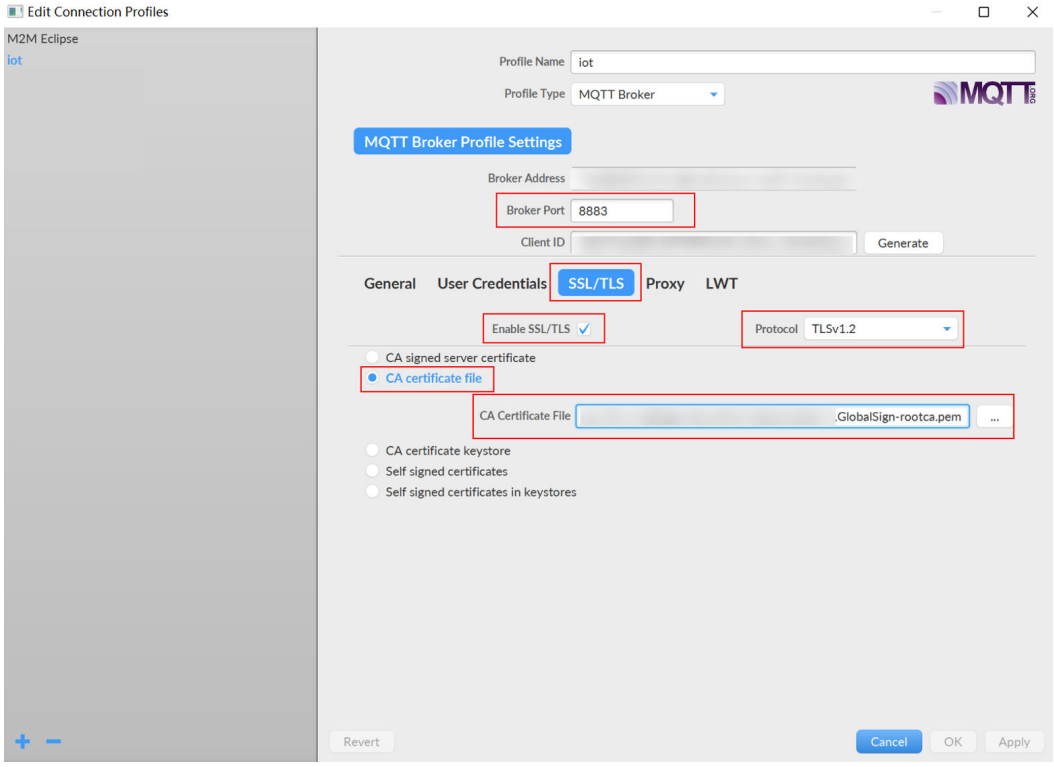


Table 1-1 Parameter description

Parameter	Description
Broker Address	Host name, which is obtained in 2. The access address is a domain name. For devices that cannot be connected to the platform using a domain name, run the **ping** *Domain name* command in the CLI to obtain the IP address. The IP address is variable and needs to be set using a configuration item.
Broker Port	8883. In this example, port 8883 is used for secure connection.
Client ID	Enter the device client ID obtained in 2.
User Name	Enter the device ID obtained in 2.
Password	Enter the encrypted device secret obtained in 2.

Step 5 Click the **SSL/TLS** tab and then select **Enable SSL/TLS**. Recommended: Set **Protocol** to **TLSv1.2**. Click **CA certificate file**, go to the **certificate resources** page to download the certificate file of the corresponding region and instance version, and enter the complete local path of the certificate file in the text box. Click **Apply**, and then click **Cancel** to exit the configuration page.

Figure 1-26 Setting SSL/TLS parameters



Step 6 Click **Connect**. If the icon in the upper right corner turns green, the device simulator has been authenticated and connected. The device status displayed on the platform is **Online**.

Figure 1-27 Device simulator connected

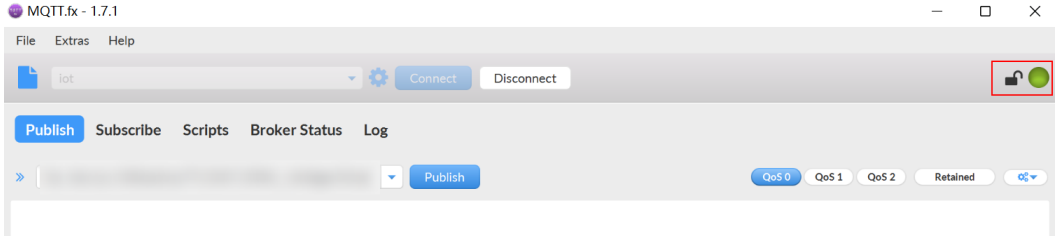
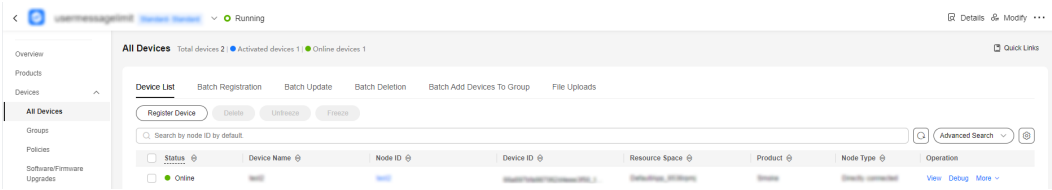


Figure 1-28 Device online status



----End

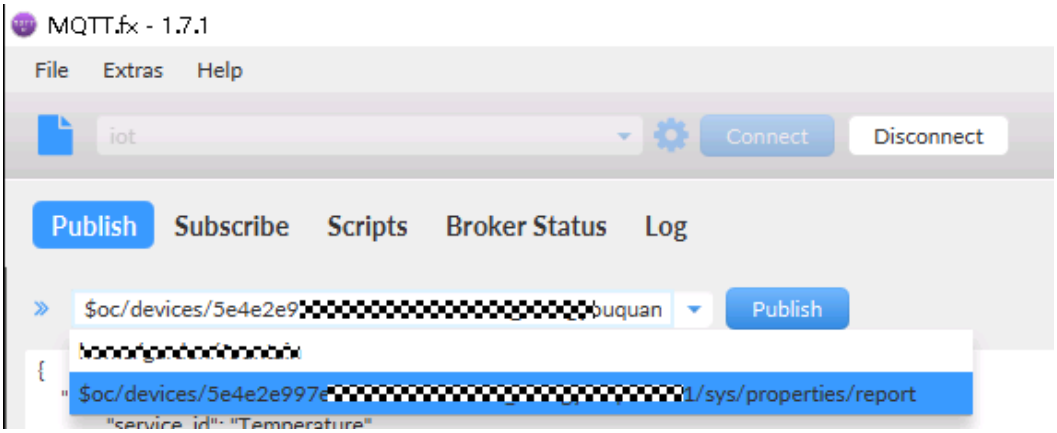
Reporting Light Intensity Data

Use MQTT.fx to report light intensity data to the IoT platform. If a device reports data through the MQTT channel, the data needs to be sent to a specific topic in the format **$oc/devices/***{device_id}***/sys/properties/report**. For devices that each

has a different secret, set *device_id* to the device ID returned upon successful device registration.

Step 1 Enter the API URL, for example, **$oc/devices/***{device_id}***/sys/properties/report**.

Figure 1-29 Entering the API address



Step 2 Enter the data to report in the blank area in the middle of the tool and click **Publish**.

Table 1-2 Service data list

Field	Mandatory/ Optional	Type	Description
services	Mandatory	List<ServiceProperty>	Service data list. (For details, see the **ServiceProperty** structure below.)

Table 1-3 ServiceProperty structure

Field	Mandatory/ Optional	Type	Description
service_id	Mandatory	String	Service ID.
properties	Mandatory	Object	Service properties, which are defined in the product model associated with the device.

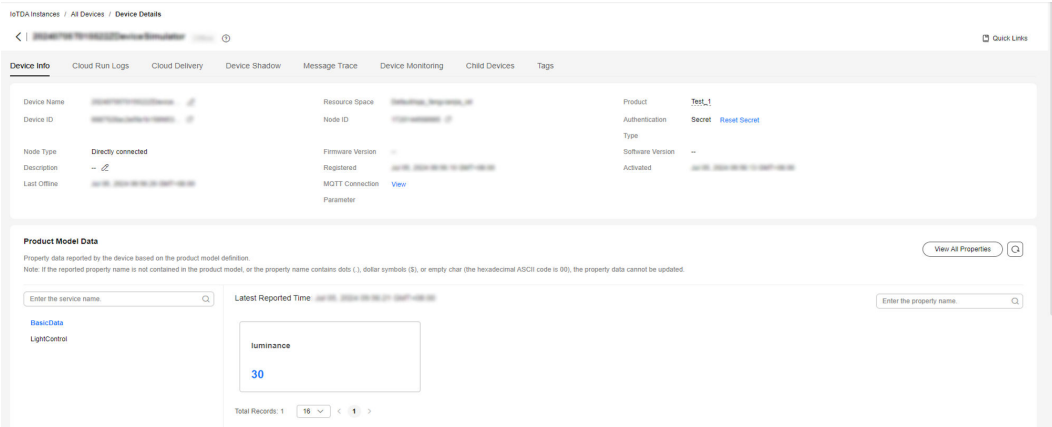
Field	Mandatory/ Optional	Type	Description
eventTime	Optional	String	UTC time when the device reports data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is not carried in the reported data or is in incorrect format, the time when IoTDA receives the data is used.

Request example:

```
{
   "services": [{
          "service_id": "BasicData",
          "properties": {
             "luminance": 30
       }
     }
  ]
}
```

Step 3 Check whether the device successfully reports data on the device details page. As shown in the following figure, the luminance is updated to 30.

Figure 1-30 Viewing reported data - MQTT



----End

Remotely Delivering Commands for Turning On the Light

Deliver a command on the console to remotely control smart street lights.

Step 1 In the navigation pane, choose **Devices** > **All Devices**, locate the target device, and click **View** to access its details page.

Step 2 Click the **Cloud Delivery** tab, click **Deliver Command**, set **Command** to **LightControl: Switch**, and set **Value** to **ON** to deliver a command for turning on the light.

Figure 1-31 Command delivery - Synchronous command delivery

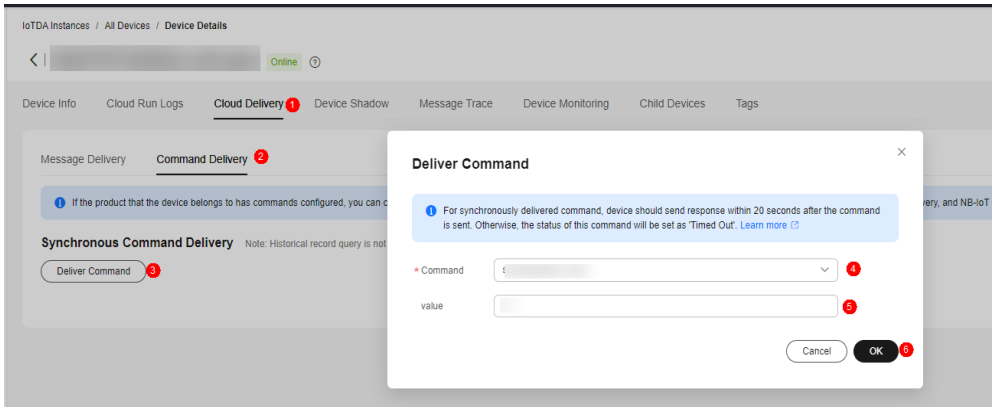
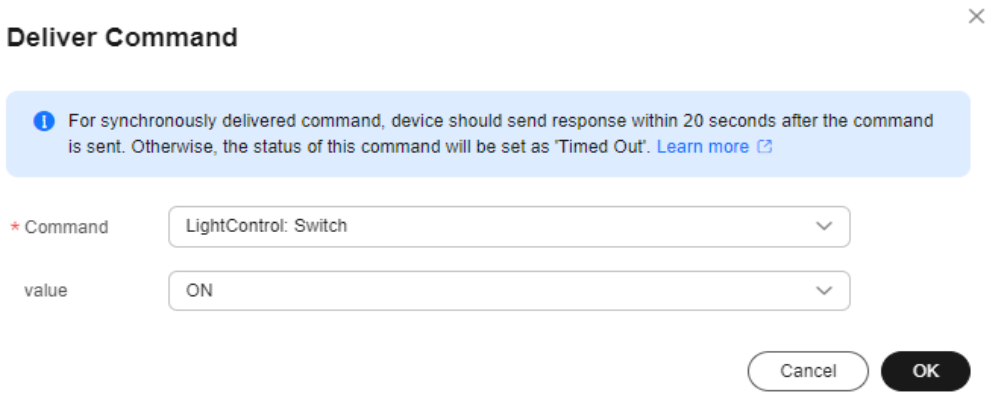


Figure 1-32 Command delivery - LightControl

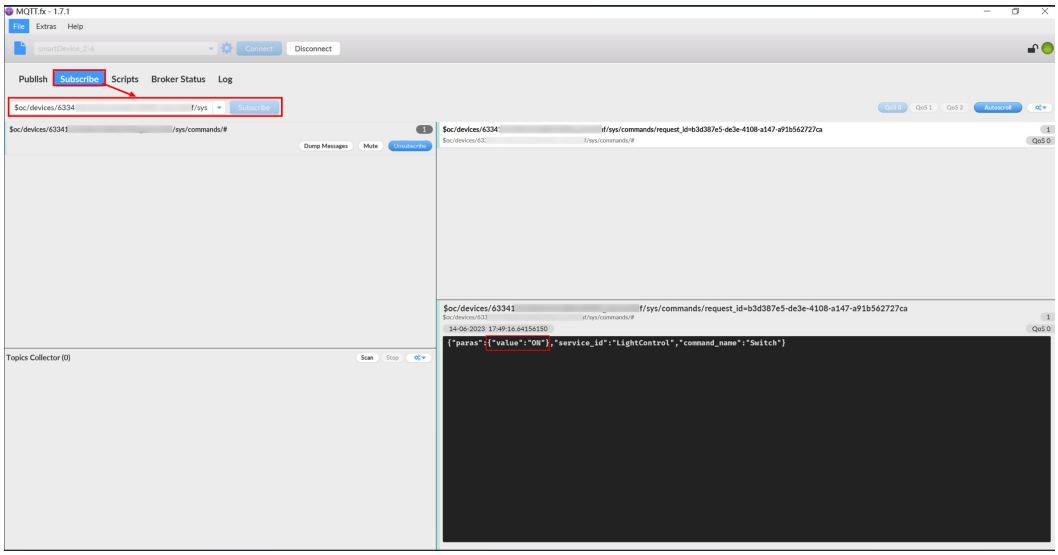


NOTE

MQTT devices support only synchronous command delivery. NB-IoT devices support only asynchronous command delivery.

Step 3 In the MQTT.fx simulator, click the **Subscribe** tab and enter the command delivery topic. After the subscription, check the delivered command parameters. The format of the command delivery topic is **$oc/devices/***{device_id}***/sys/commands/#**. As shown in the following figure, the MQTT.fx simulator has received the command whose **command_name** is **Switch** and value is **ON**.

Figure 1-33 Checking the delivered command parameters



NOTE

The device needs to respond to the delivered synchronous command in a timely manner. However, MQTT.fx does not automatically report the command response. Therefore, the console page may display a message indicating that the command request times out. For details about the command response, see **Platform Delivering a Command**.

----End

1.5 Using a Virtual Smart Street Light to Communicate with the Platform (Java SDK)

Overview

This section describes how to connect a device to Huawei Cloud IoTDA through MQTTS/MQTT using Java code, implement southbound data reporting and command delivery using **platform APIs**, and receive messages subscribed by the northbound server using the application-side sample code. Taking a smart street light as an example, the device reports information such as luminance to IoTDA, and an application receives device data pushed by IoTDA.

Prerequisites

- You have installed JDK 1.8 or later.
- You have installed IntelliJ IDEA. If you have not installed IntelliJ IDEA, visit the **IntelliJ IDEA official website** to download and install it.

Uploading a Product Model

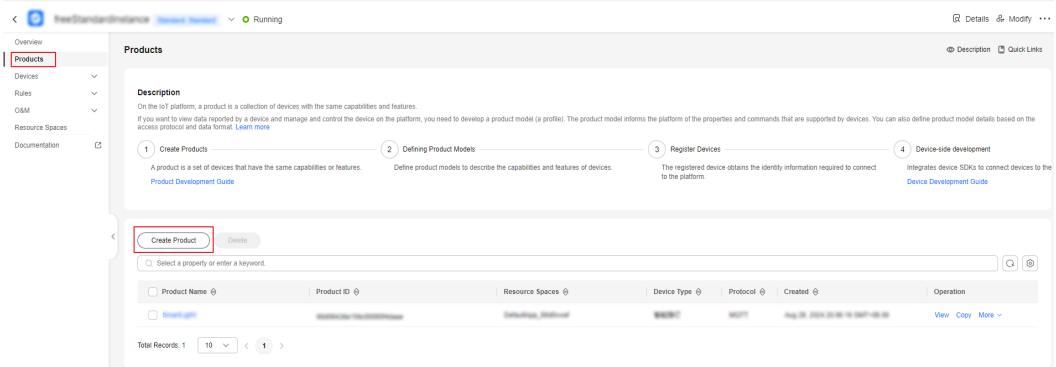
A product model is a JSON file that describes device capabilities. It defines basic device properties and message formats for data reporting and command delivery. Defining a product model is to construct an abstract model of a device in the platform to enable the platform to understand the device function.

Procedure

Step 1 Access the **IoTDA** service page and click **Access Console**.

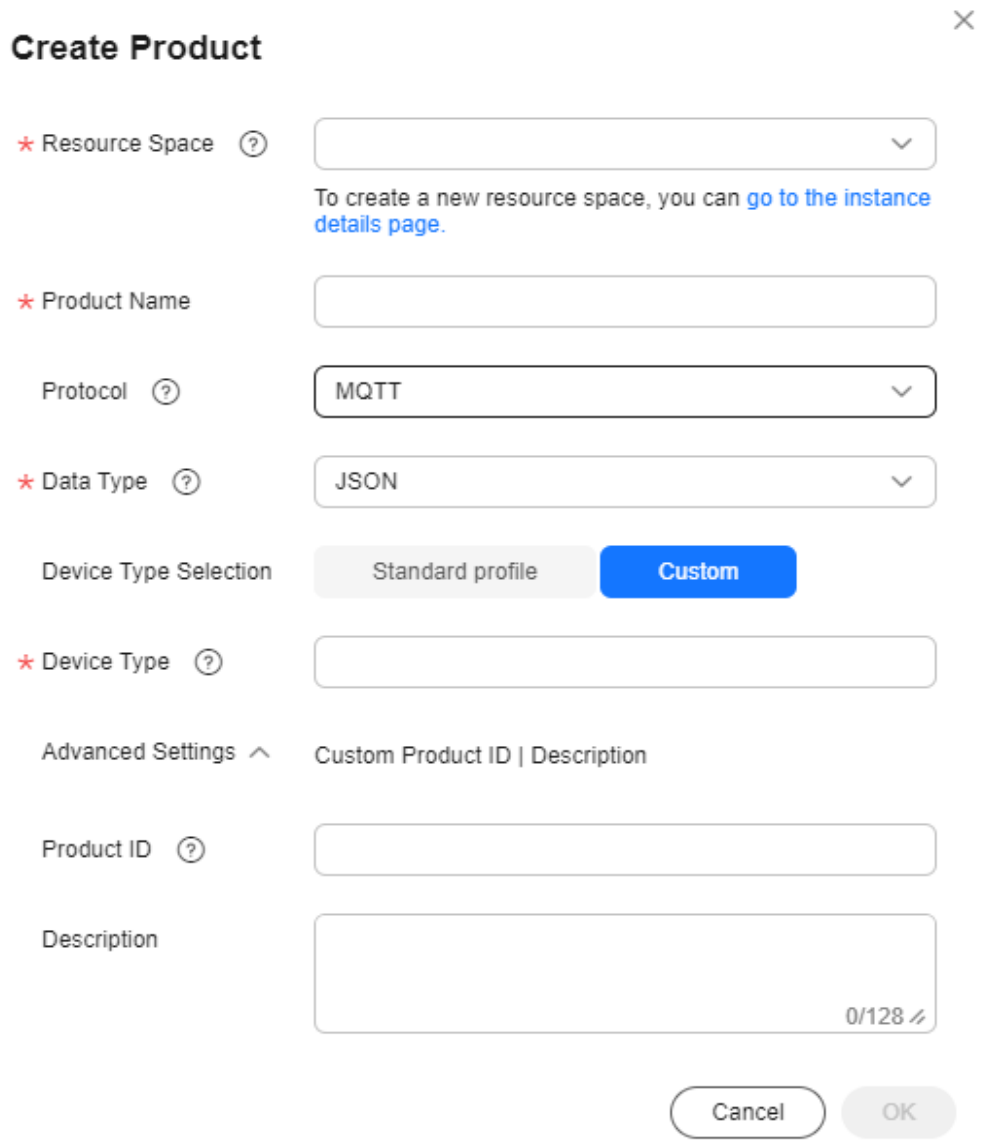
Step 2 Choose **Products** in the navigation pane and click **Create Product**.

Figure 1-34 Creating a product



Step 3 In the displayed dialog box, set parameters based on your requirements.

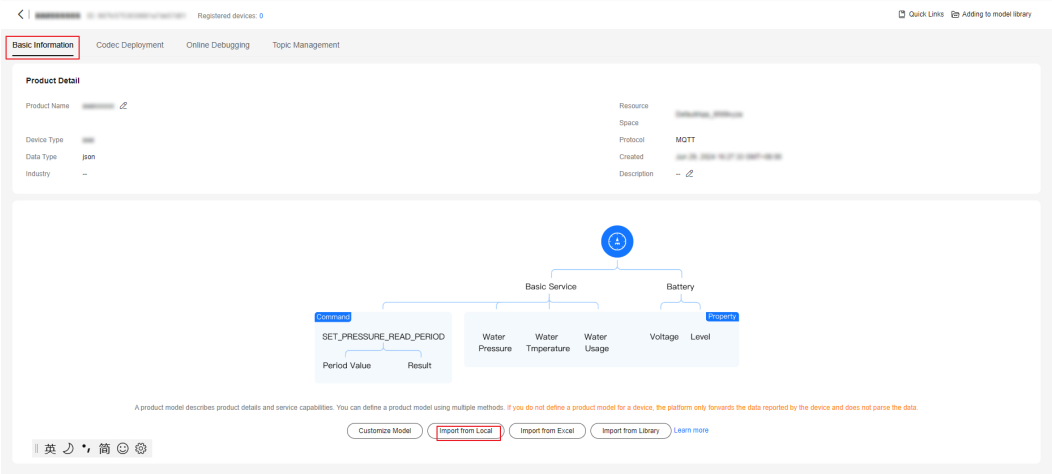
Figure 1-35 Creating a product - MQTT



Step 4 Download the **model file**. For details about the development process, see **Developing a Product Model Online**.

Step 5 After the product is created, click the product, and then click **Import from Local** to upload the downloaded model file. The model file does not need to be decompressed, and the package name cannot contain brackets.

Figure 1-36 Uploading a product model - MQTT

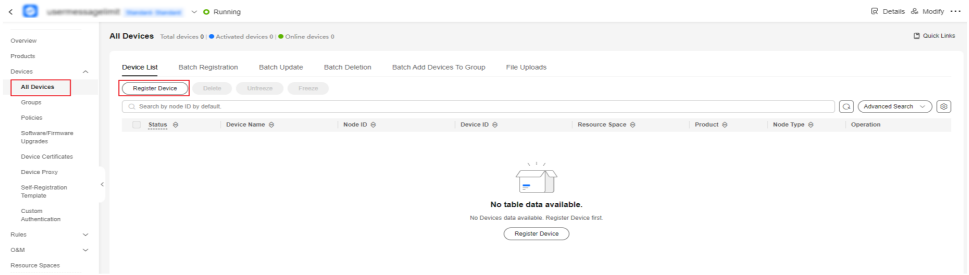


----End

Creating a Device

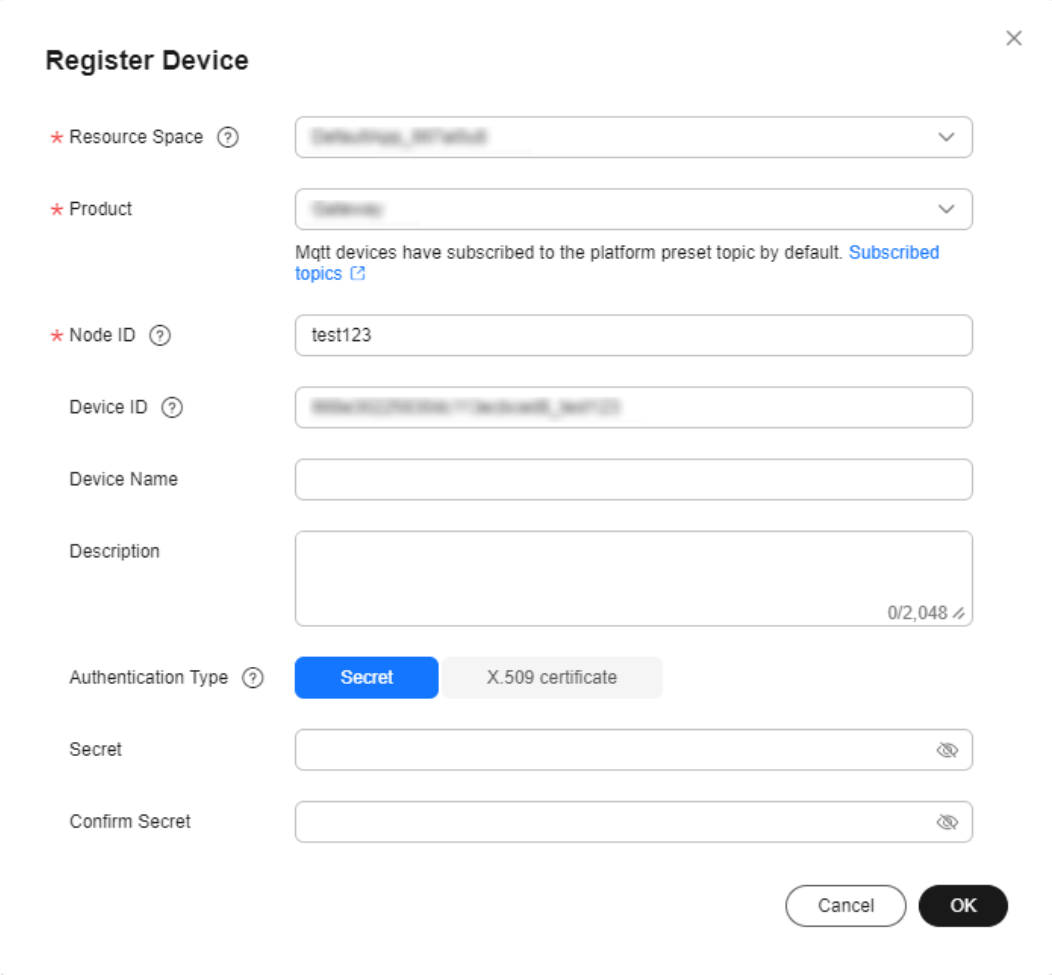
Step 1 In the navigation pane, choose **Devices** > **All Devices**, and click **Register Device**.

Figure 1-37 Registering a device



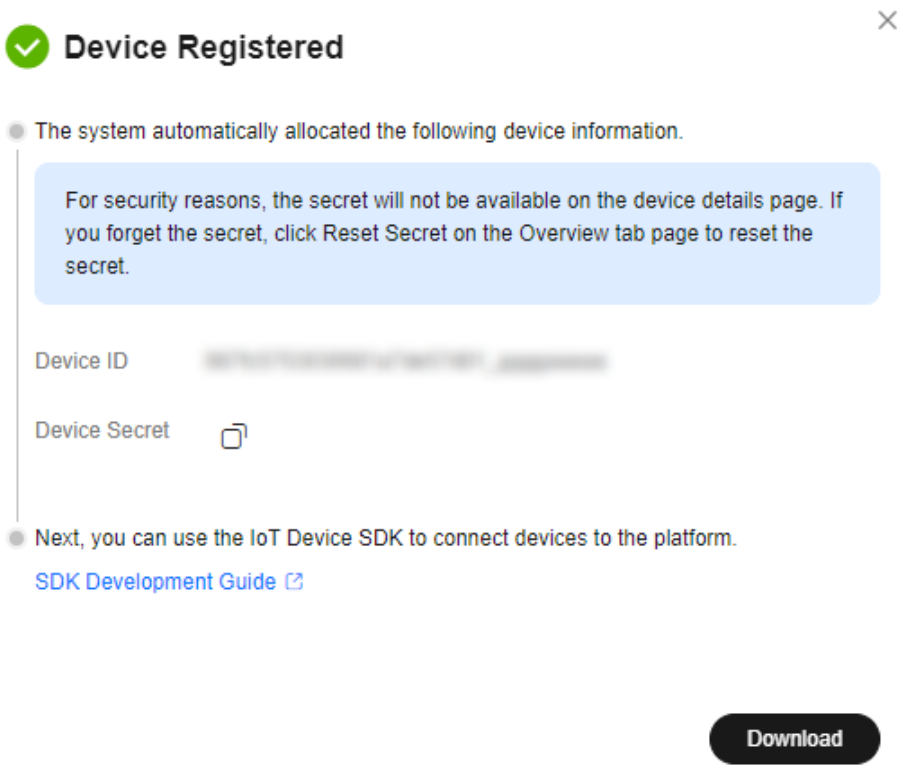
Step 2 In the displayed dialog box, configure the parameters by referring to the following figure (select the created product), and click **OK**. If you do not specify **Secret**, a secret will be automatically generated by the platform. In this example, the secret is automatically generated.

Figure 1-38 Registering a device (test123)



Step 3 After the device is created, save the device ID and secret, which will be used for device connection.

Figure 1-39 Device registered

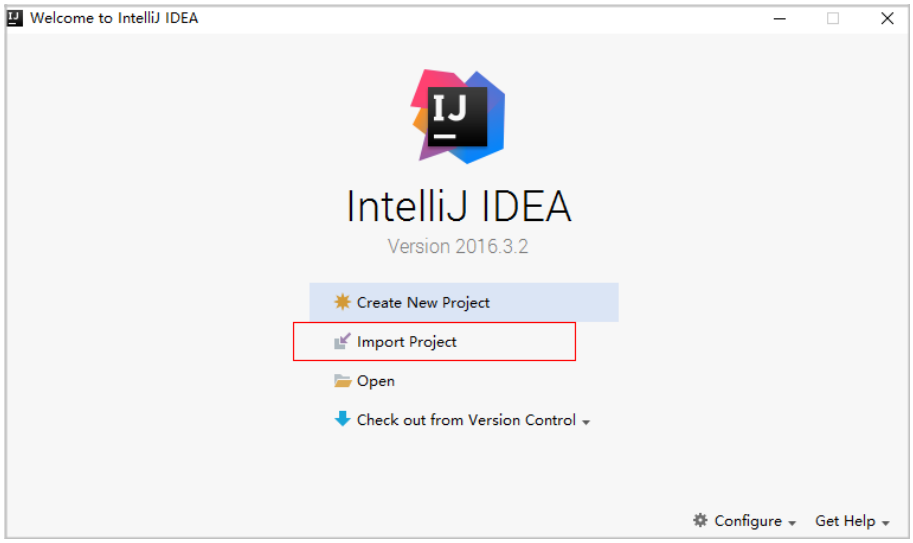


----End

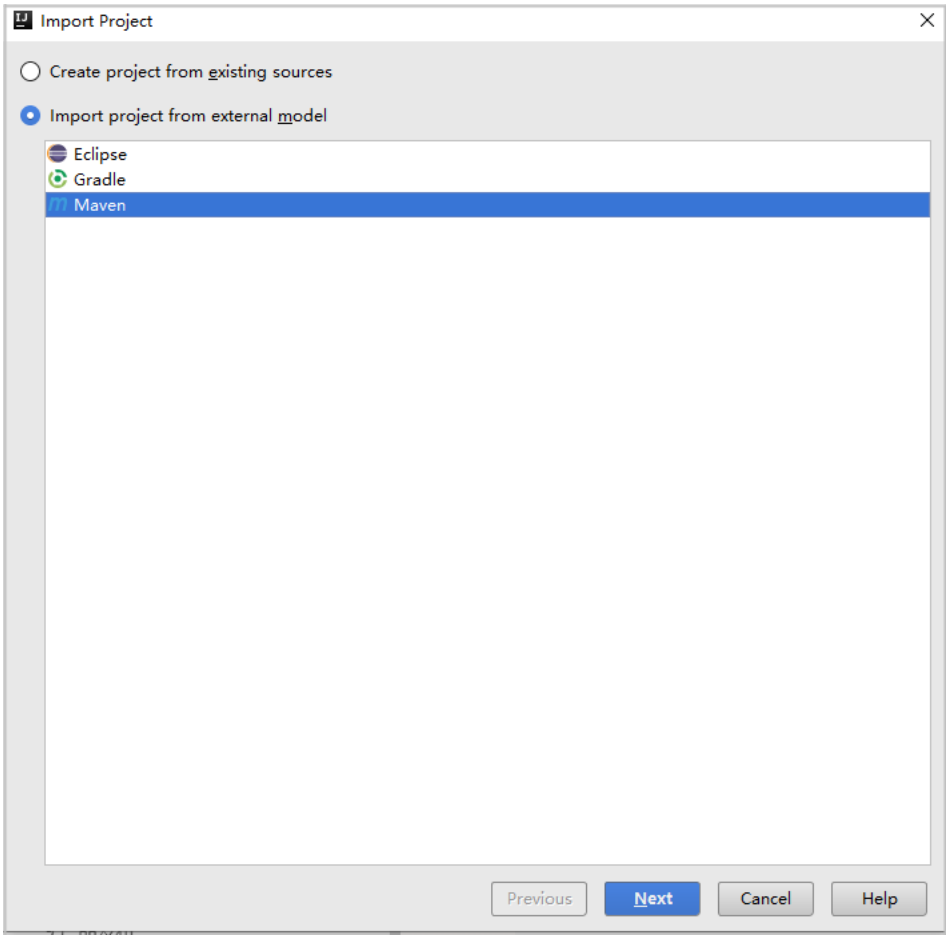
Importing Sample Code

Step 1 Download the **Java demo**.

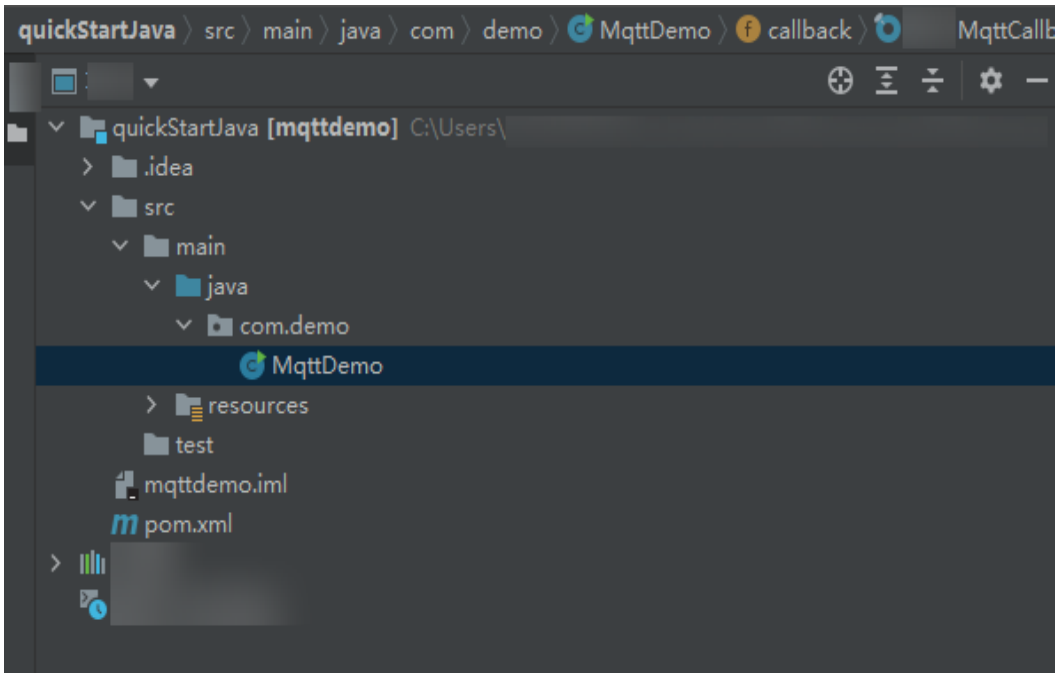
Step 2 Open the IDEA developer tool and click **Import Project**.



Step 3 Select the Java demo downloaded in **1** and click **Next**.



Step 4 Import the sample code.



----End

Establishing a Connection

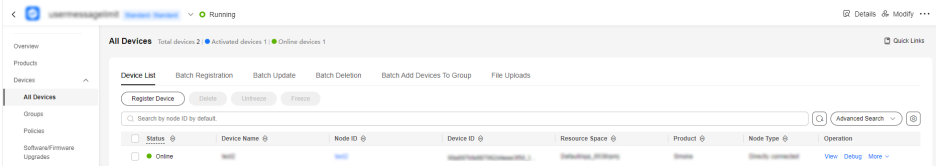
To connect a device or gateway to the platform, upload the device information to bind the device or gateway to the platform.

1. Before establishing a connection, modify the following parameters:

```
// MQTT connection address of IoTDA
static String serverIp = "iot-mqtts.cn-north-4.myhuaweicloud.com";
// Device ID and secret obtained during device registration (Replace them with the actual values.)
static String deviceId = "yourDeviceID"; // device_id obtained during device registration
static String secret = "yourSecret";     // secret obtained during device registration
```

   - **serverIp** indicates the address used by devices to access IoTDA using MQTT. For details about how to obtain the address, see **Obtaining Resources**.
   - **device_id** and **secret** indicate the device ID and secret, which can be obtained after **the device is registered**.
2. Run the program. The device is displayed as online on the platform.

Figure 1-40 Device list - Device online status



Reporting Properties

A device reports its properties to IoTDA. (The sample code implements scheduled reporting. You can view the data reported by the device in IoTDA by referring to **Viewing Reported Data**.)

```
// Report JSON data. service_id must be the same as that defined in the product model.
String jsonMsg = "{\"services\":[{\"service_id\":\"BasicData\",\"properties\":{\"luminance\":32},\"eventTime
\":null}]}";
```

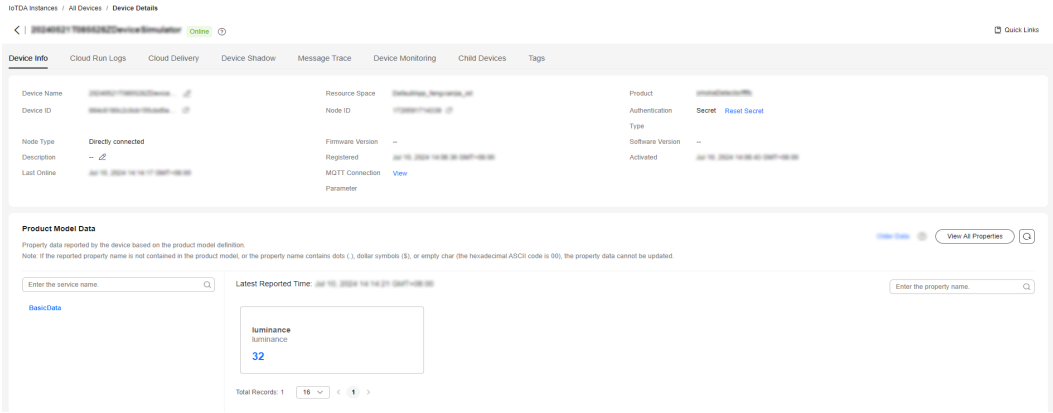
- The message body **jsonMsg** is assembled in JSON format, and **service_id** must be the same as that defined in the product model. **properties** indicates a device property.
- **luminance** indicates the street light brightness.
- **eventTime** indicates the UTC time when the device reports data. If this parameter is not specified, the system time is used by default.

After a device or gateway is connected to the platform, you can call **publish(String topic,MqttMessage message)** of **MqttAsyncClient** to report device properties to the platform.

Viewing Reported Data

After the **main** method is called, you can view the reported device property data on the device details page. For details about the API, see **Device Reporting Properties**.

Figure 1-41 Viewing reported data - luminance



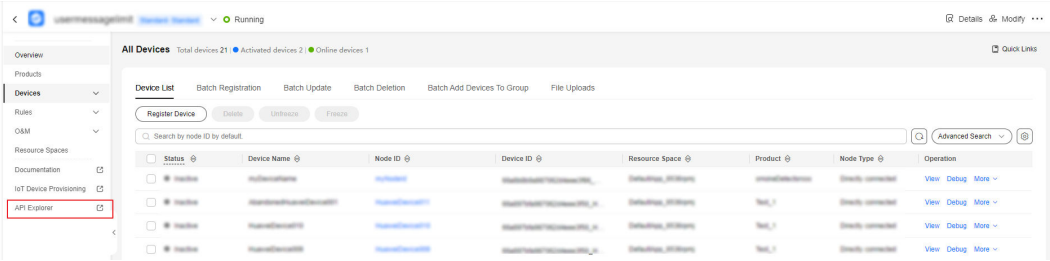
NOTE

If no latest reported data is displayed on the device details page, modify the services and properties in the product model to ensure that the services and properties reported by the device are consistent with those in the product model. If they are inconsistent, the data reported by the device is not available on the historical data page. Alternatively, delete all services on the **Basic Information** page.

Delivering a Command

Step 1 In the navigation pane, choose **API Explorer**.

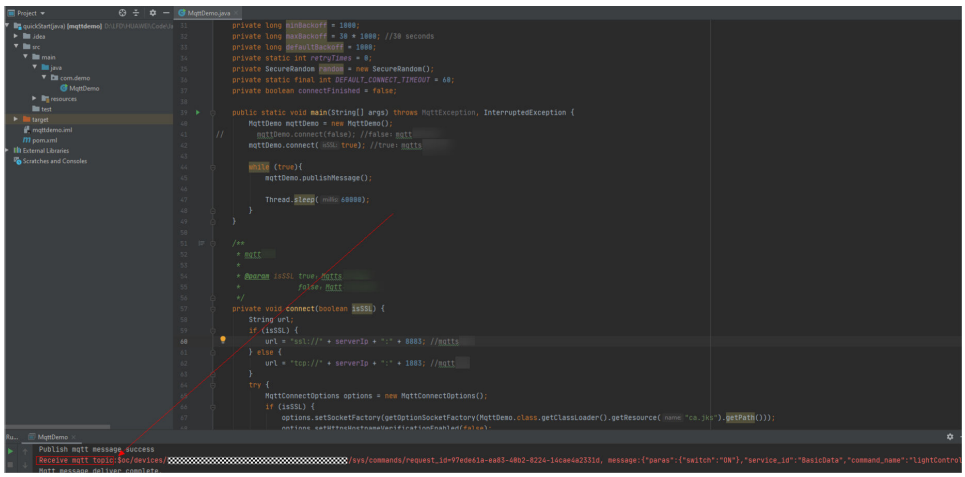
Figure 1-42 Navigation pane -API retrieval and debugging



Step 2 Locate the row that contains the device command. For details about the delivered parameters, see the figure (consistent with those in the product model). Then, click **Debug** to send the command.

- **service_id** indicates the service ID, for example, **BasicData**.
- **command_name** indicates the command name, for example, **lightControl**.
- **paras** indicates a delivered parameter, for example, **{"switch":"ON"}**.

You can view the received commands on the device. (The sample code has implemented the subscription to the command receiving topic.)



----End

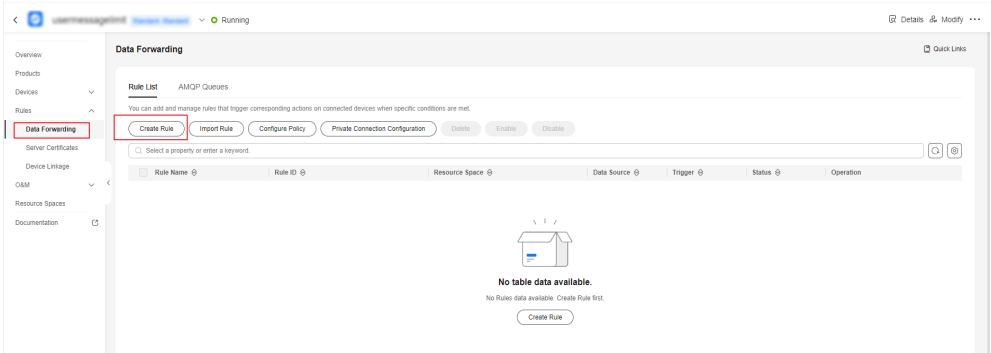
Obtaining Data Reported by a Device from the Cloud

The following uses AMQP as an example to describe how to obtain data reported by a device to the cloud.

Step 1 Obtain the Java AMQP access **demo**.

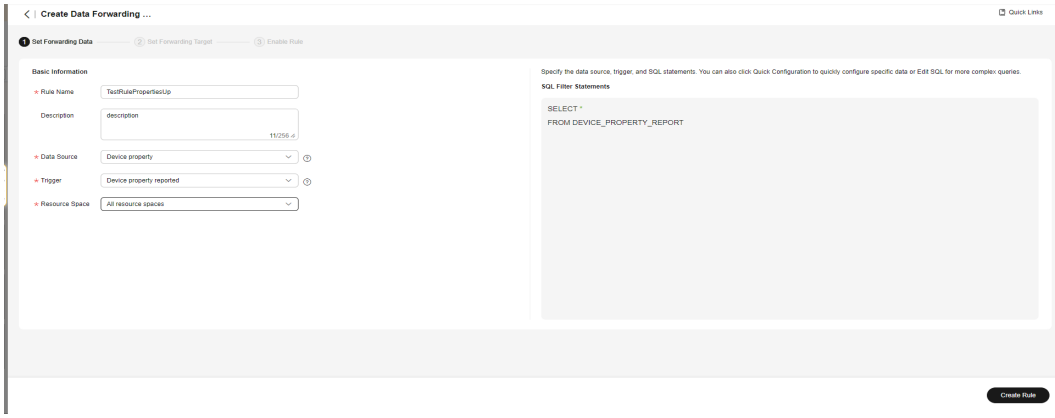
Step 2 Log in to the **console**, choose **Rules** > **Data Forwarding**, and click **Create Rule** to create a data forwarding rule.

Figure 1-43 Data forwarding - Creating a rule



Step 3 On the **Set Forwarding Data** page, configure parameters, and click **Create Rule**.

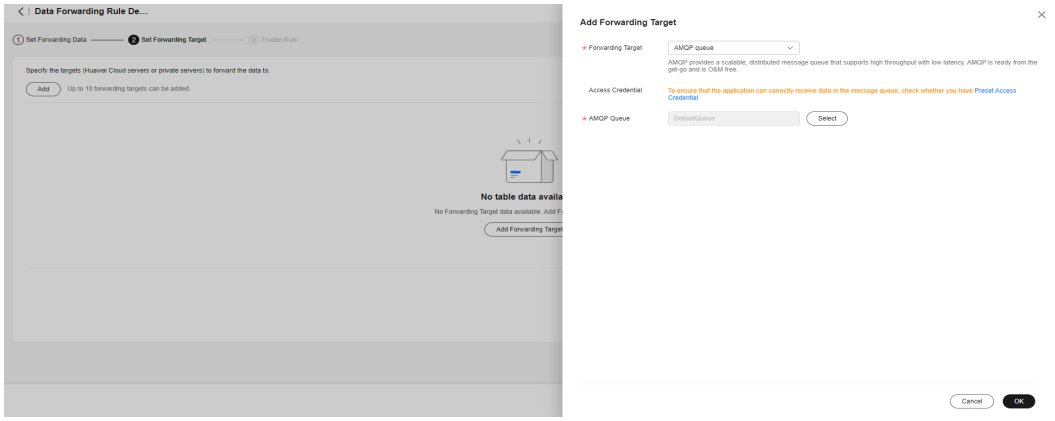
Figure 1-44 Data forwarding - Creating a property reporting rule



Parameter	Description
Rule Name	Customize a rule name.
Description	Describe the rule.
Data Source	Select **Device property**.
Trigger	Select **Device property reported**.
Resource Space	Select **All resource spaces**.

Step 4 Set the forwarding target. Note that you need to click **Preset Access Credential** to download the file.

Figure 1-45 Creating a forwarding target - to an AMQP push message queue

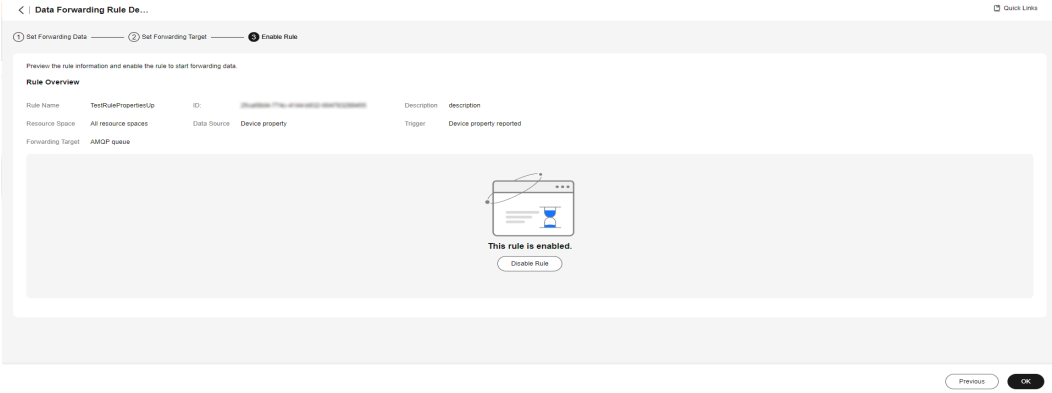


Parameter	Description
Forwarding Target	Select **AMQP message queue**.

Parameter	Description
Access Credential	Click **Preset Access Credential** and save the downloaded file, which includes **access_key** and **access_code**.
Message Queue	**DefaultQueue** is selected by default.

Step 5 Click **Enable Rule**.

Figure 1-46 Enabling a rule - Forwarding data to AMQP

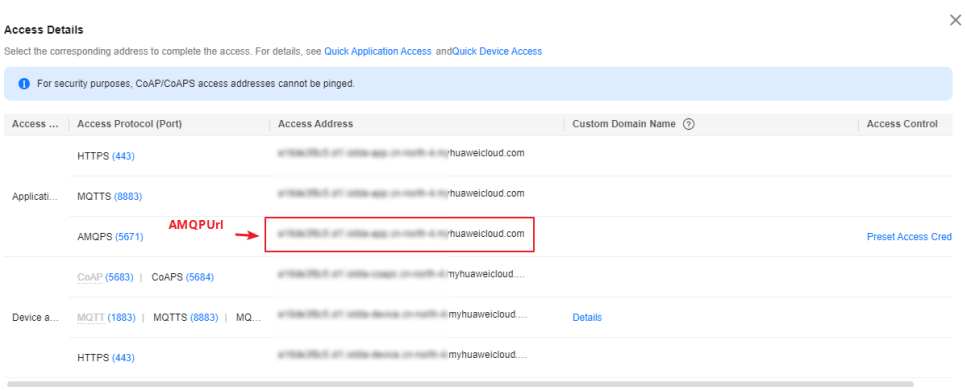


Step 6 Modify the parameters in the AMQP sample code obtained in **Step 1**.

```
        Runtime.getRuntime().availableProcessors(),  maximumPoolSize: Runtime.getRuntime().availableProcessors
        keepAliveTime: 60, TimeUnit.SECONDS, new LinkedBlockingQueue<>( capacity: 5000));

public static void main(String[] args) throws Exception {

    String accessKey = "yourAccessKey";
    long timeStamp = System.currentTimeMillis();

    String userName = "accessKey=" + accessKey + "|timestamp=" + timeStamp;

    String password = "yourAccessCode";

    String baseUrl = "yourAMQPUrl";
    String connectionUrl = "amqps://" + baseUrl + ":5671?amqp.vhost=default&amqp.idleTimeout=8000&amqp.saslM
    Hashtable<String, String> hashtable = new Hashtable<>();
    hashtable.put("connectionfactory.HwConnectionURL", connectionUrl);

    String queueName = "yourQueue";
    hashtable.put("queue.HwQueueName", queueName);
    hashtable
            .put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");
    Context context = new InitialContext(hashtable);
```

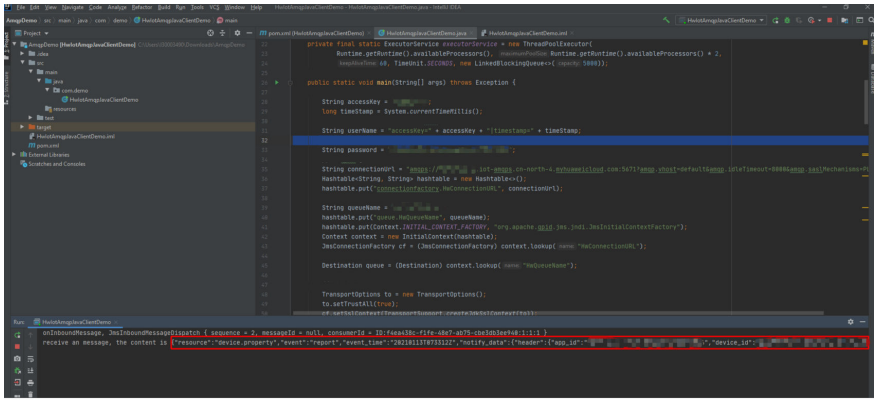
- **yourAccessKey**: access key of the access credential. For details about how to obtain it, see **Step 4**.
- **yourAccessCode**: access code of the access credential. For details about how to obtain it, see **Step 4**.
- **yourAMQPUrl**: AMQP domain name. You can log in to the **console**, choose **Overview**, and click **Access Addresses** to obtain the domain name, as shown in the following figure.

Figure 1-47 Access information - AMQP access address



- **yourQueue**: queue name. Use the default queue **DefaultQueue**.

Step 7 AMQP data is received successfully.



----End

Additional Information

For more development guides, see **Using IoT Device SDKs for Access** and **Using MQTT Demos for Access**.

1.6 Using a Virtual Smart Street Light to Communicate with the Platform (C SDK)

Overview

This section describes how to connect a device to Huawei Cloud IoTDA through MQTTS/MQTT using C code, implement southbound data reporting and command delivery using platform APIs, and receive messages subscribed by the northbound server using the application-side sample code. Taking a smart street light as an example, the device reports information such as luminance to IoTDA, and an application receives device data pushed by IoTDA.

Prerequisites

You have installed Linux and GCC (4.8 or later).

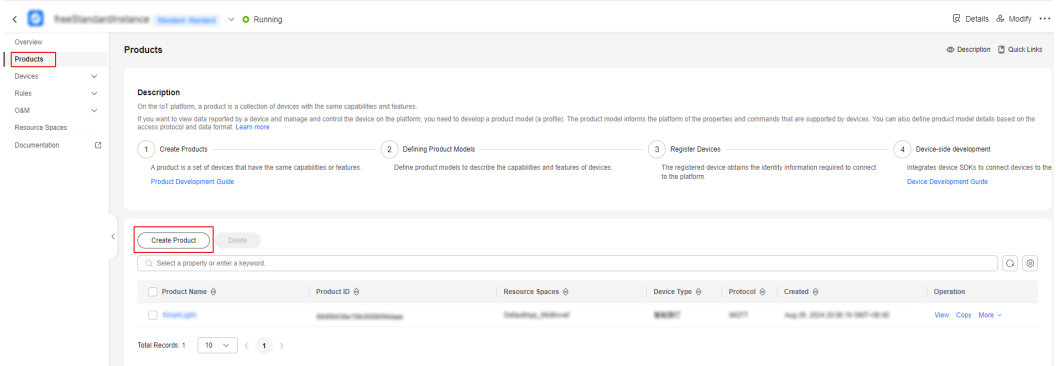
Uploading a Product Model

A product model is a JSON file that describes device capabilities. It defines basic device properties and message formats for data reporting and command delivery. Defining a product model is to construct an abstract model of a device in the platform to enable the platform to understand the device function.

Step 1 Access the **IoTDA** service page and click **Access Console**. Click the target instance card.

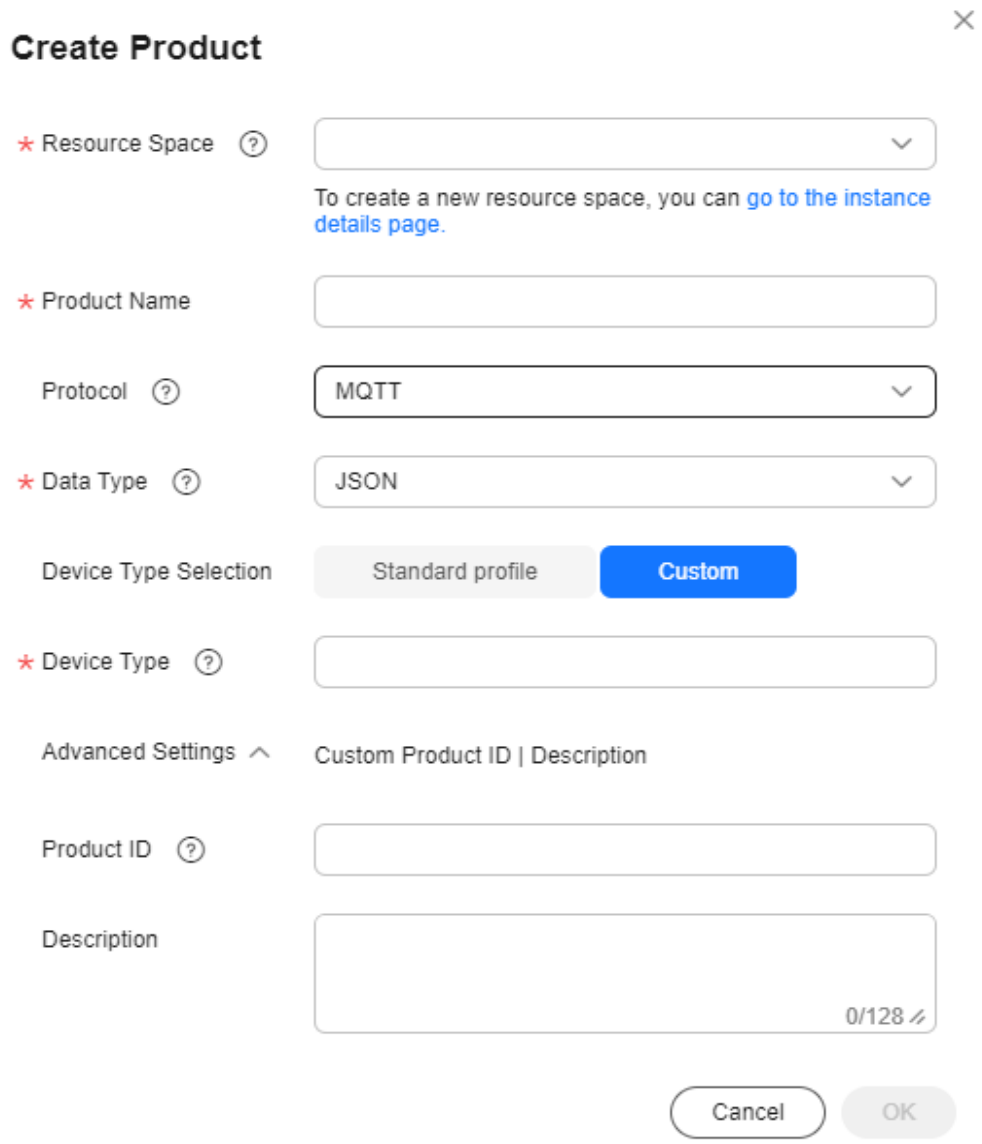
Step 2 Choose **Products** in the navigation pane and click **Create Product**.

Figure 1-48 Creating a product



Step 3 In the displayed dialog box, set parameters based on your requirements.

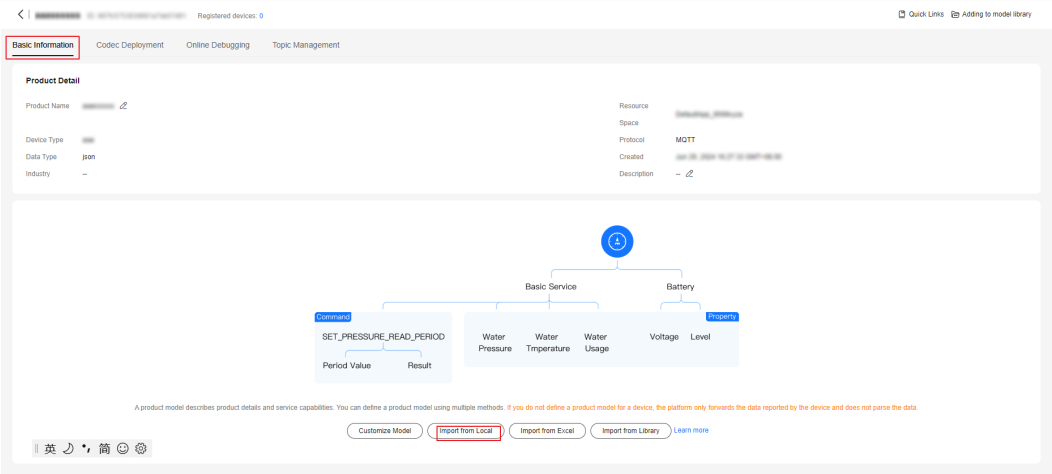
Figure 1-49 Creating a product - MQTT



Step 4 Download the **model file**. For details about the development process, see **Developing a Product Model Online**.

Step 5 After the product is created, click the product, and then click **Import from Local** to upload the downloaded model file. The model file does not need to be decompressed, and the package name cannot contain brackets.

Figure 1-50 Uploading a product model - MQTT

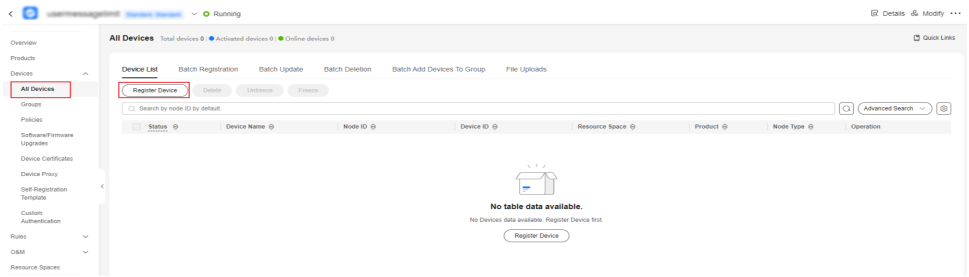


----End

Creating a Device

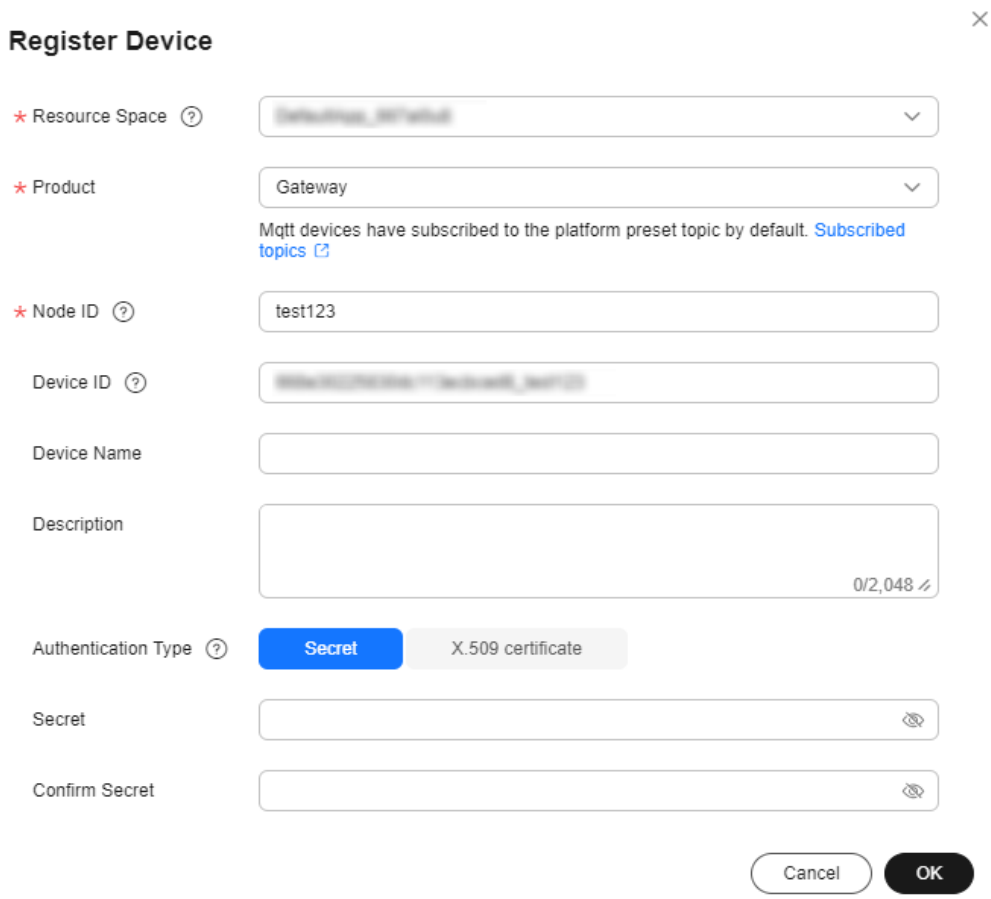
Step 1 In the navigation pane, choose **Devices** > **All Devices**, and click **Register Device**.

Figure 1-51 Registering a device



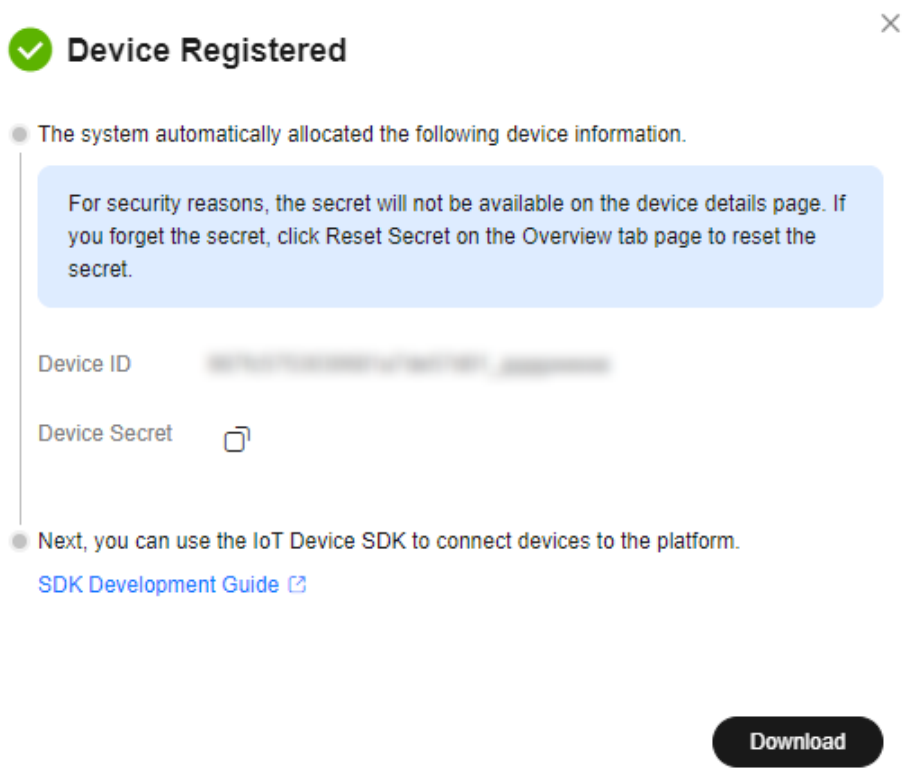
Step 2 In the displayed dialog box, configure the parameters by referring to the following figure (select the created product), and click **OK**. If you do not specify **Secret**, a secret will be automatically generated by the platform. In this example, the secret is automatically generated.

Figure 1-52 Registering a device - test



Step 3 After the device is created, save the device ID and secret, which will be used for device connection.

Figure 1-53 Device registered

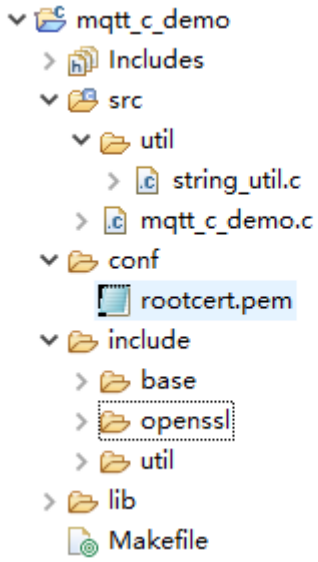


----End

Importing Sample Code

Step 1 Download the sample code quickStart(C).

Step 2 Copy the code to the Linux runtime environment. The following figure shows the code file hierarchy.



Description of the directories:

- **src**: source code directory

  mqtt_c_demo: core source code of the demo

  util/string_util.c: utility resource file
- **conf**: certificate directory

  rootcert.pem: certificate used by the device to verify the platform identity. It is used for login authentication when the device connects to the platform.
- **include**: header files

  base: dependent Paho header files

  openssl: dependent OpenSSL header files

  util: header files of the dependent tool resources
- **lib**: dependent library file

  libcrypto.so*/libssl.so*: OpenSSL library file

  libpaho-mqtt3as.so*: Paho library file
- **Makefile**: Makefile

----End

Compiling Library Files

- Compiling the OpenSSL library

  a. Download **OpenSSL**, upload it to any directory on the Linux compiler, and run the following command to decompress it:
  ```
  tar -zxvf openssl-1.1.1d.tar.gz
  ```

  b. Generate a **makefile**.

  Run the following command to access the OpenSSL source code directory:
  ```
  cd openssl-1.1.1d
  ```

  Create a directory (for example, **/home/test**) for OpenSSL compilation.
  ```
  mkdir /home/test
  ```

  Create a directory for OpenSSL compilation.
  ```
  mkdir /home/test/openssl
  ```

  Create a configuration file directory.
  ```
  mkdir /home/test/openssl/ssl
  ```

  Run the following configuration command:
  ```
  ./config shared --prefix=/home/test/openssl --openssldir=/home/test/openssl/ssl
  ```

  In this command, **prefix** is the installation directory, **openssldir** is the configuration file directory, and **shared** is used to generate a dynamic-link library (**.so** library).

  If an exception occurs during the compilation, add **no-asm** to the configuration command (indicating that the assembly code is not used).

  ./config no-asm shared --prefix=/home/test/openssl --openssldir=/home/test/openssl/ssl

  ```
  [root@server-1908071538 test]# cd openssl-1.1.1d
  [root@server-1908071538 openssl-1.1.1d]# ./config shared --prefix=/home/test/openssl --openssldir=/home/test/openssl/ssl
  ```

  c. Generate library files.

  Run the following command in the OpenSSL source code directory:

```
make depend
```

Run the following command for compilation:

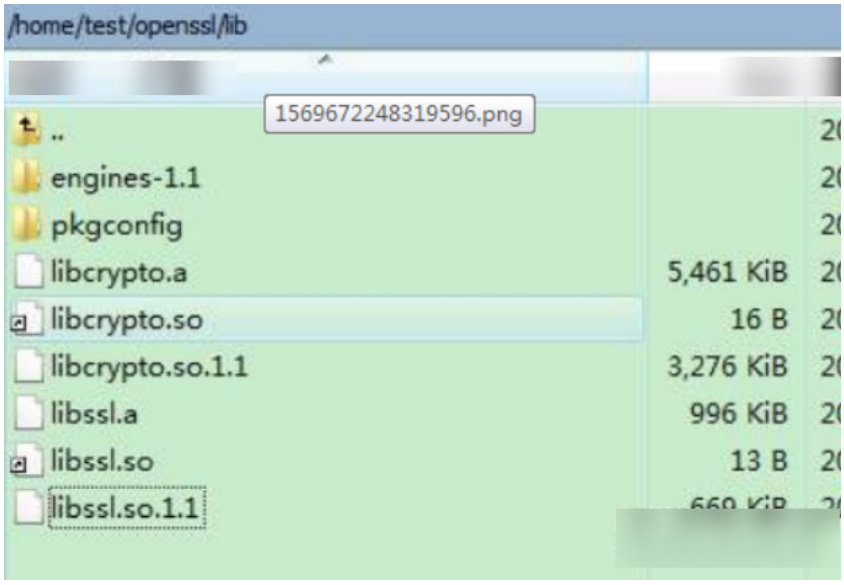
```
make
```

Install OpenSSL.

```
make install
```

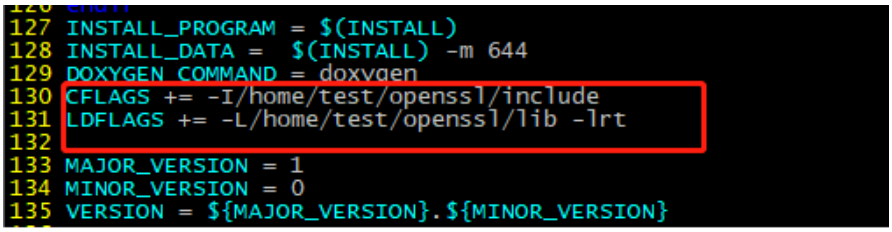
Find the **lib** directory in **home/test/openssl** under the OpenSSL installation directory.

The library files **libcrypto.so.1.1**, **libssl.so.1.1**, **libcrypto.so**, and **libssl.so** are generated. Copy these files to the **lib** folder of **quickStart(C)** and copy the content in **/home/test/openssl/include/openssl** to **include/openssl** of **quickStart(C)**.



Note: Some compilation tools are 32-bit. If these tools are used on a 64-bit Linux computer, delete **-m64** from the **makefile** before the compilation.

- Compiling the Eclipse Paho library file
  a. Download the **paho.mqtt.c source code**.
  b. Decompress the package and upload it to the Linux compiler.
  c. Modify the **makefile**.
     i. Run the following command to edit the **makefile**:
     ```
     vim Makefile
     ```
     ii. Run the following command to display the number of lines:
     ```
     :set nu
     ```
     iii. Add the following two lines (customized OpenSSL header files and library files) after line 129:
     ```
     CFLAGS += -I/home/test/openssl/include
     LDFLAGS += -L/home/test/openssl/lib -lrt
     ```



iv. Change the addresses in lines 195, 197, 199, and 201 to the corresponding addresses.

```
194
195 CCFLAGS_SO += -Wno-deprecated-declarations -DOSX -I /home/test/openssl/include
196 LDFLAGS_C += -Wl,-install_name,lib$(MQTTLIB_C).so.${MAJOR_VERSION}
197 LDFLAGS_CS += -Wl,-install_name,lib$(MQTTLIB_CS).so.${MAJOR_VERSION} -L /home/test/openssl/lib
198 LDFLAGS_A += -Wl,-install_name,lib${MQTTLIB_A}.so.${MAJOR_VERSION}
199 LDFLAGS_AS += -Wl,-install_name,lib${MQTTLIB_AS}.so.${MAJOR_VERSION} -L /home/test/openssl/lib
200 FLAGS_EXE += -DOSX
201 FLAGS_EXES += -L /home/test/openssl/lib
202
203 LDCONFIG = echo
204
205 endif
```

d. Start the compilation.

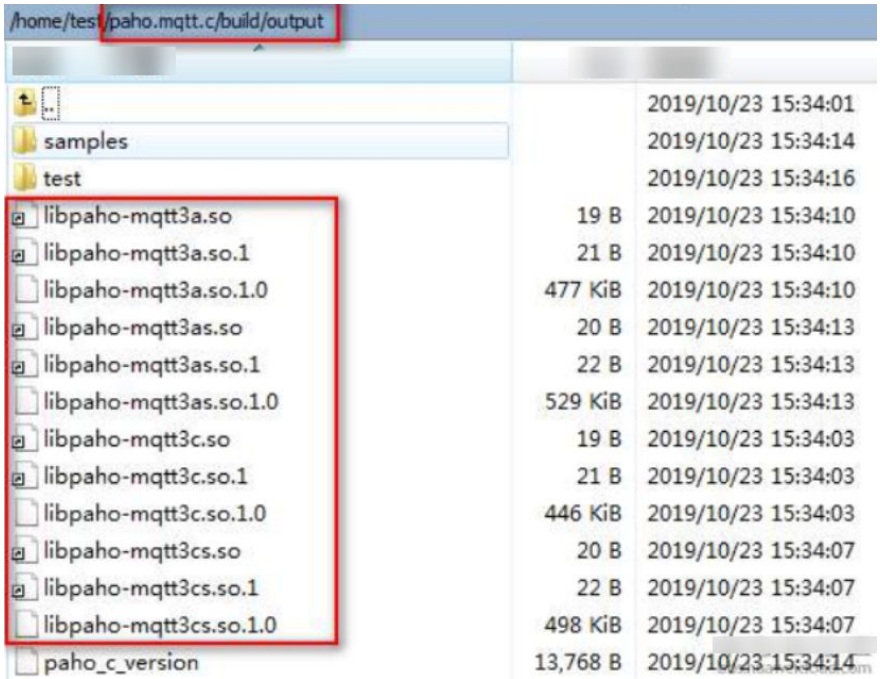
i. Run the following command:

```
make clean
```

ii. Run the following command:

```
make
```

e. After the compilation is complete, you can view the libraries that are compiled in the **build/output** directory.



f. Copy the Paho library file.

Currently, only **libpaho-mqtt3as** is used in the SDK. Copy the **libpaho-mqtt3as.so** and **libpaho-mqtt3as.so.1** files to the **lib** folder of **quickStart(C)**. Go back to the Paho source code directory, and copy **MQTTAsync.h**, **MQTTClient.h**, **MQTTClientPersistence.h**, **MQTTProperties.h**, **MQTTReasonCodes.h**, and **MQTTSubscribeOpts.h** in the **src** directory to the **include/base** directory of **quickStart(C)**.

Establishing a Connection

To connect a device or gateway to the platform, upload the device information to bind the device or gateway to the platform.

Step 1 Configure the parameters. Change the values of **username** and **password** only. For details, see **Obtaining Resources**.

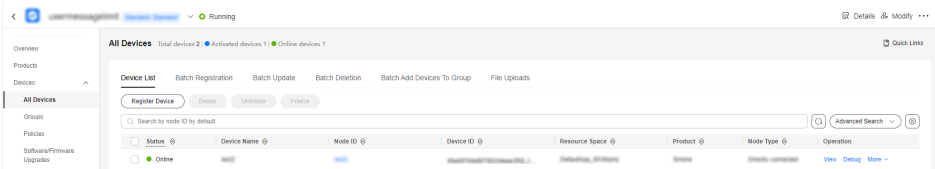
Step 2 Start the connection.

1. Run the **make** command for compilation. Delete **-m64** from the **makefile** in a 32-bit OS.
2. Run **export LD_LIBRARY_PATH=./lib/** to load the library file.
3. Run **./MQTT_Demo.o**.

Step 3 If the connection is successful, the message "connect success" is displayed. The device is also displayed as **Online** on the console.

```
begin to connect the server.
connect success.
```

Figure 1-54 Device list - Device online status



----End

Reporting Properties

A device reports its properties to IoTDA. The sample code implements scheduled reporting. You can view the data reported by the device in IoTDA. For details, see **Device Reporting Properties**.

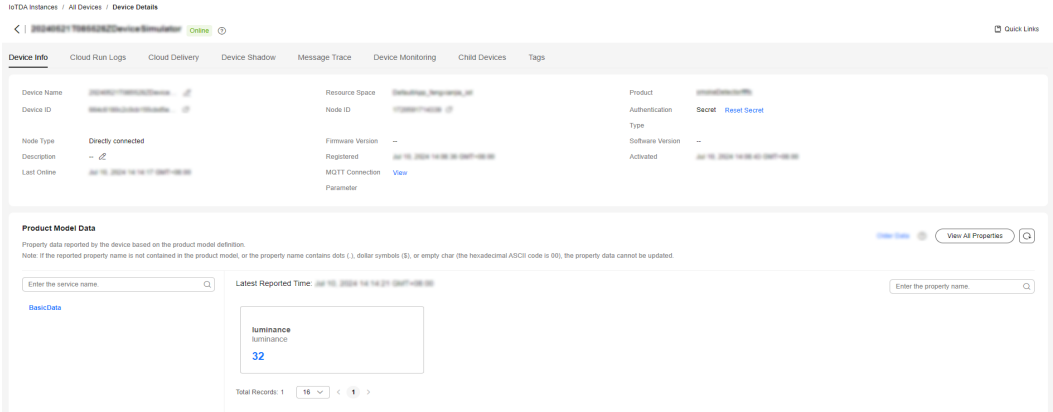
```
//publish data
char *payload = "{\"services\":[{\"service_id\":\"BasicData\",\"properties\":{\"luminance\":32},\"eventTime
\":NULL}]}";
```

- The message body **payload** is assembled in JSON format, and **service_id** must be the same as that defined in the product model. **properties** indicates a device property.
- **luminance** indicates the street light brightness.
- **eventTime** indicates the UTC time when the device reports data. If this parameter is not specified, the system time is used by default.

If the property reporting is successful, the message "publish success" is displayed in the demo.

The reported properties are displayed on the device details page.

Figure 1-55 Viewing reported data - luminance



Receiving a Command

After subscribing to a command topic, you can deliver a synchronous command on the console. For details, see **Synchronous Command Delivery to an Individual MQTT Device**.

If the command delivery is successful, the command received is displayed in the demo:

```
mqtt_message_arrive() success, the topic is $oc/devices/6[redacted]/sys/commands/request_id=8[redacted]
[redacted], the payload is {"paras":{"switch":"OFF"},"service_id":"BasicData","command_name":"lightControl"}
```

Obtaining Data Reported by a Device from the Cloud

After the platform receives, an application can receive push messages using AMQP. For details, see **Obtaining Data Reported by a Device from the Cloud**.

Additional Information

For more development guides, see **Using IoT Device SDKs for Access** and **Using MQTT Demos for Access**.

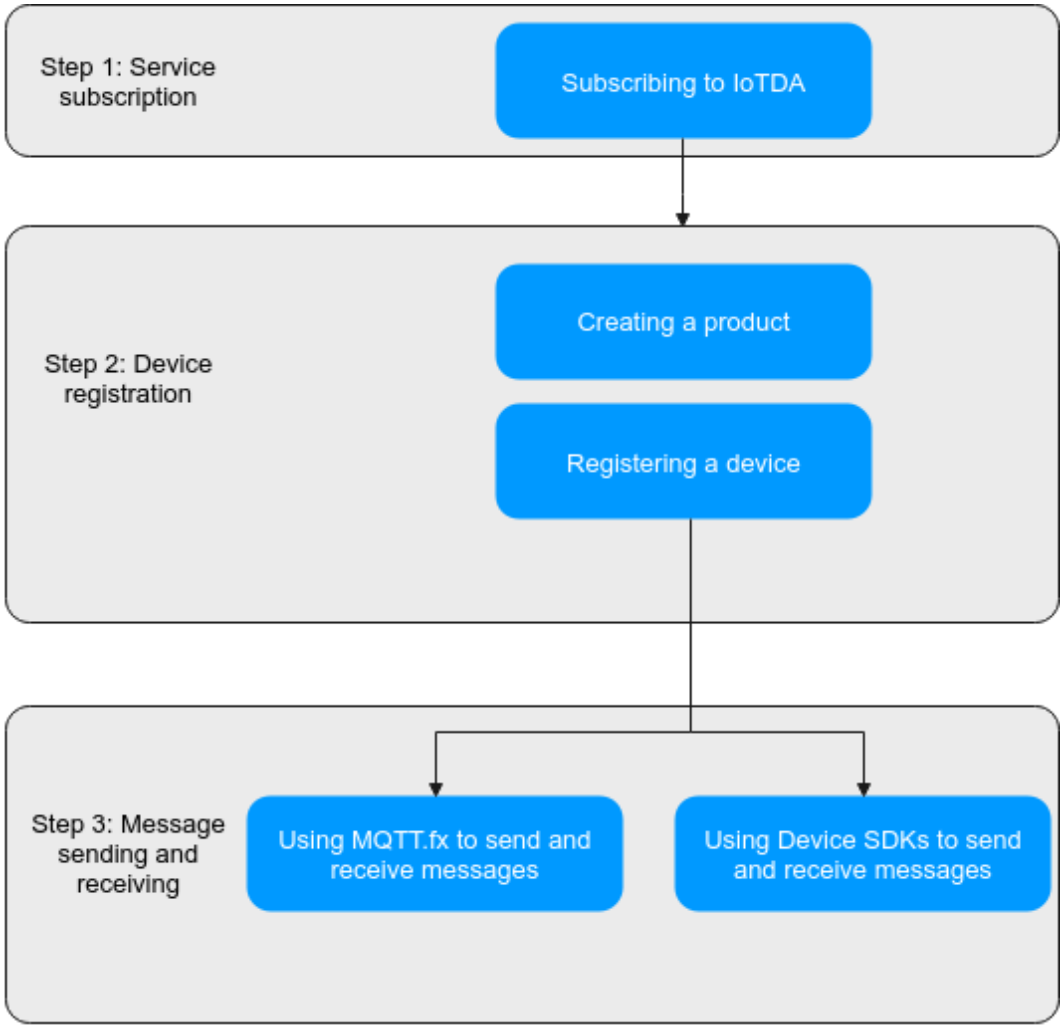
2 Quick Device Access - Message Sending and Receiving

2.1 Overview

IoTDA allows devices to access the IoT platform using MQTTS or MQTTT and send and receive messages. This section uses MQTT.fx and Java SDKs as examples to describe how a device sends and receives messages using MQTTS or MQTTT.

Procedure

Figure 2-1 Procedure



Perform the following steps.

1. Subscribe to IoTDA. You can select the instances specifications and number of units as required.
2. Register a device. Create a product and register a device on the **IoTDA** console.
   - A product is a collection of devices with the same capabilities or features.
   - Device: A device is a physical entity under a product and is used for identity authentication (necessary) when the device connects to the platform. You can use an application to call the API for **creating a device**, register a device on the console, or use the **self-registration** capability to automatically register the device when it is connected to the platform.
3. Send and receive messages.
   - Use MQTT.fx to send and receive messages.
   - Use device SDKs to send and receive messages.

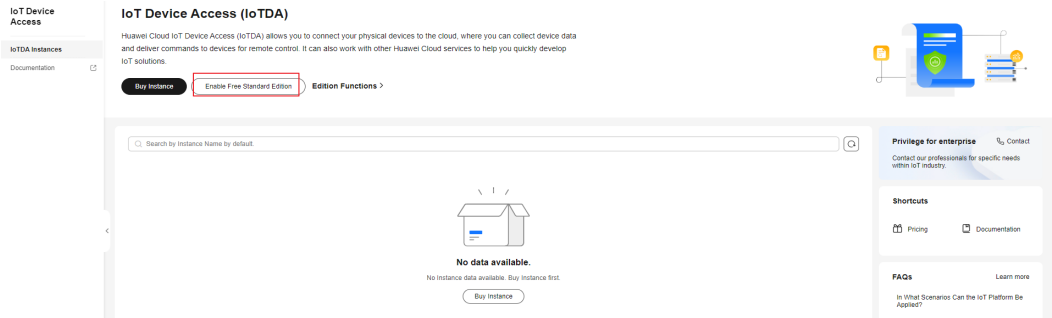
2.2 Subscribing to IoTDA

This section describes how to subscribe to a free IoTDA instance of the standard edition in the CN-Hong Kong region.

Step 1 Access the **IoTDA** service page and click **Access Console**.

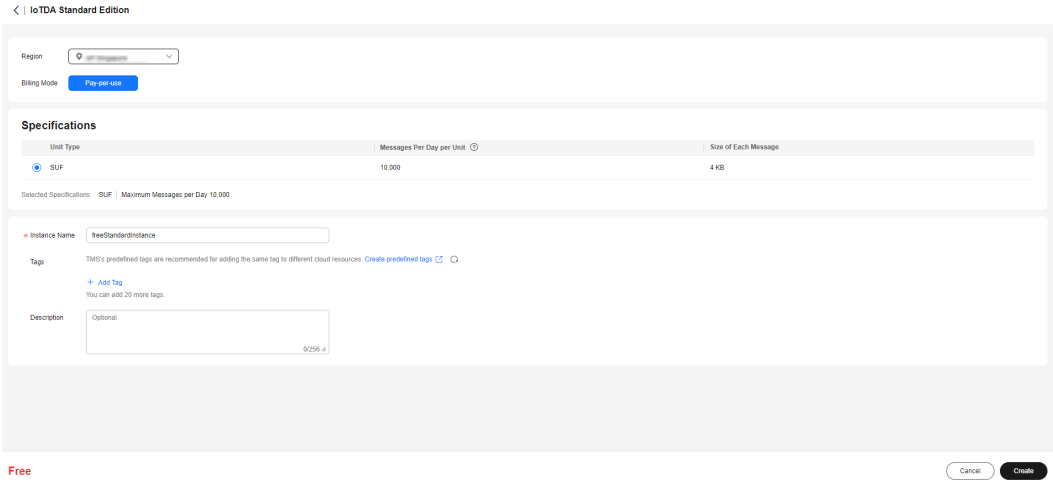
Step 2 In the navigation pane, choose **IoTDA Instances** and click the button for subscribing to a free instance.

Figure 2-2 Standard edition - Enabling free instances



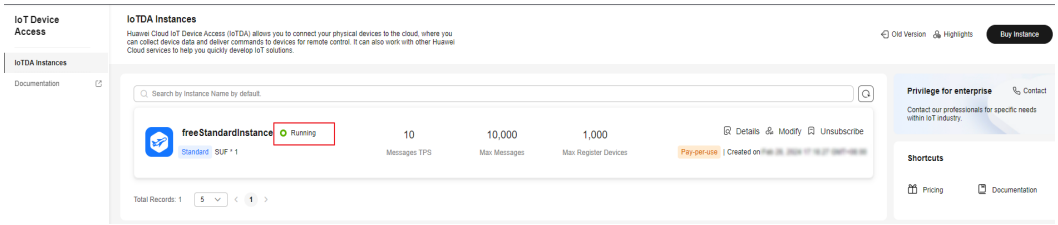
Step 3 Configure as required (recommended: default value).

Figure 2-3 Instance - Configuring a free instance



Step 4 Click **Create**, and the instance page is displayed. Refresh the page and wait until the instance status changes to **Running**, indicating that the free instance is successfully created.

Figure 2-4 Instance - Free instance created



----End

2.3 Registering a Device

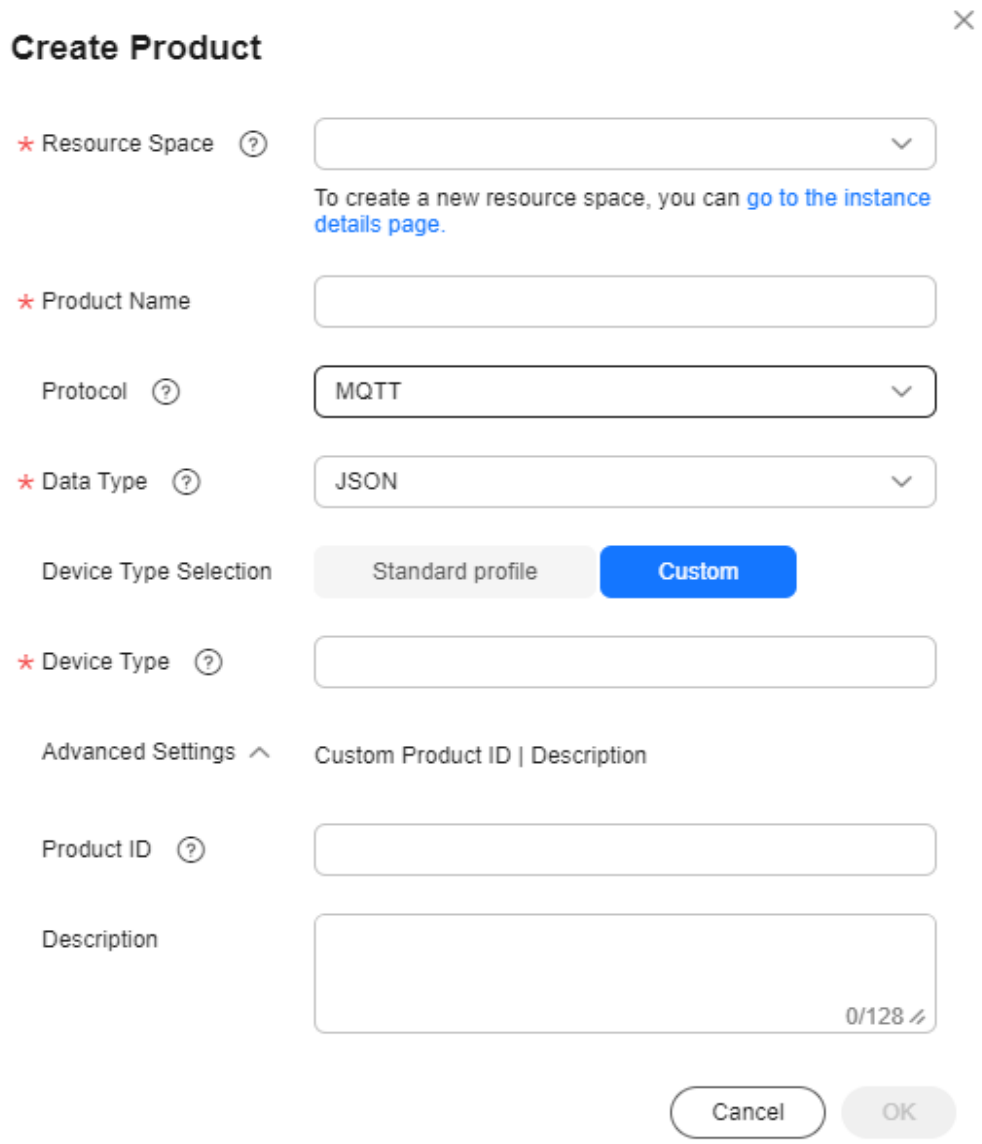
Before using IoTDA to send and receive messages, you need to create a product and device.

Step 1 Access the **IoTDA** service page and click **Access Console**. In the navigation pane, choose **IoTDA Instances** and click the target instance card.

Step 2 Create a product. Choose **Products** in the navigation pane and click **Create Product** on the left. Set the parameters as prompted and click **OK**.

Parameter	Description
Product Name	Set it to **MyProduct**.
Data Type	Select **JSON**.
Industry	Set this parameter as required.
Device Type	Set this parameter as required.

Figure 2-5 Creating a product - MQTT

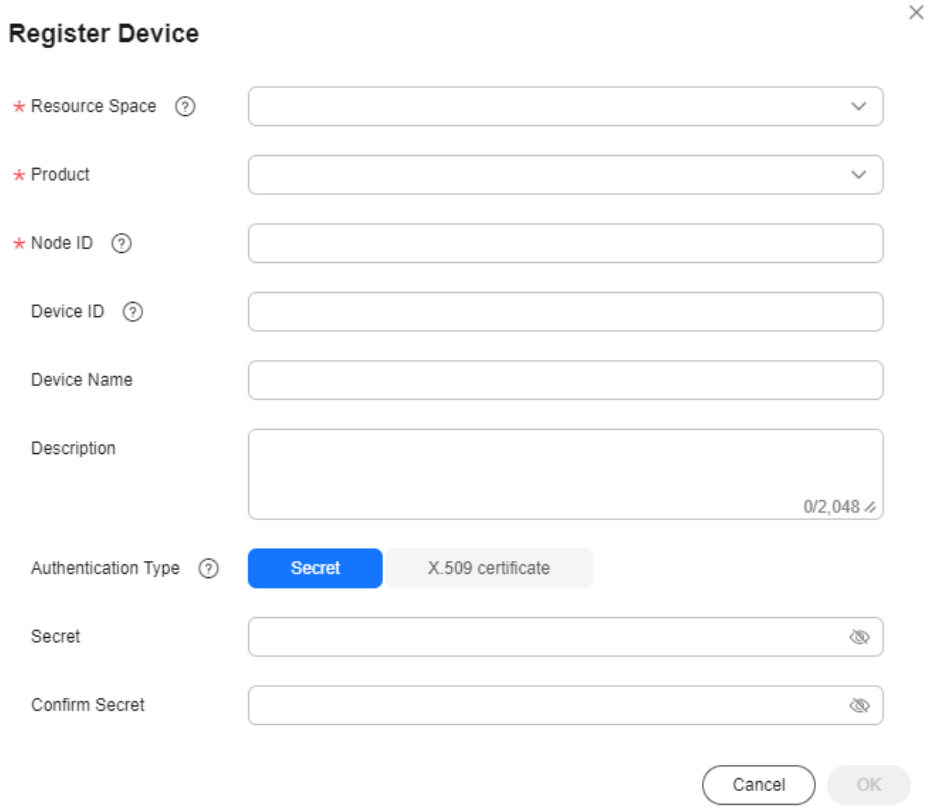


Step 3 Register a device. In the navigation pane, choose **Devices** > **All Devices**, click **Register Device**, set parameters based on the following table, and click **OK**. After the device is registered, keep the device ID and secret properly for authentication when the device accesses the IoT platform.

Parameter	Description
Product	Select the product to which the device belongs.
Node ID	Set it to **test001**.
Device Name	Set it to **test001**.
Authentication Type	Select **Secret**.

Parameter	Description
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates one.

Figure 2-6 Device - Registering a secret device



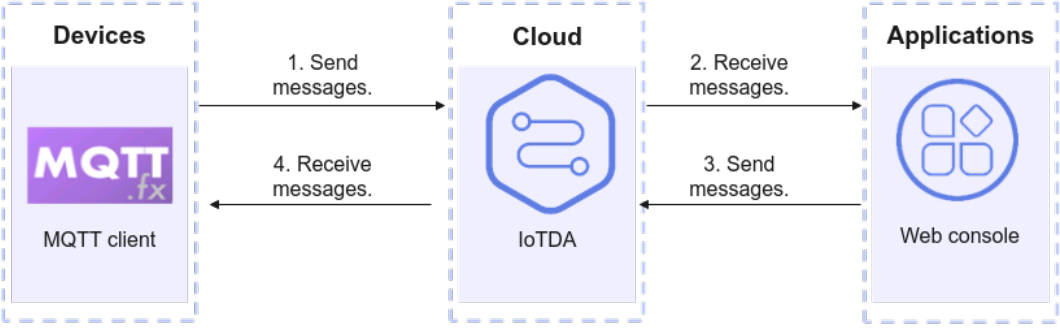
----End

2.4 Using MQTT.fx to Send and Receive Messages

Introduction

MQTT.fx is an MQTT client written in Java based on Eclipse Paho. It supports Windows, macOS, and Linux OSs. It can be used to simulate the connection of devices to Huawei Cloud IoTDA through MQTTS/MQTT, and the publishing and subscription of messages through topics. This section uses Windows as an example to describe how to use MQTT.fx to access Huawei Cloud IoTDA and send and receive messages.

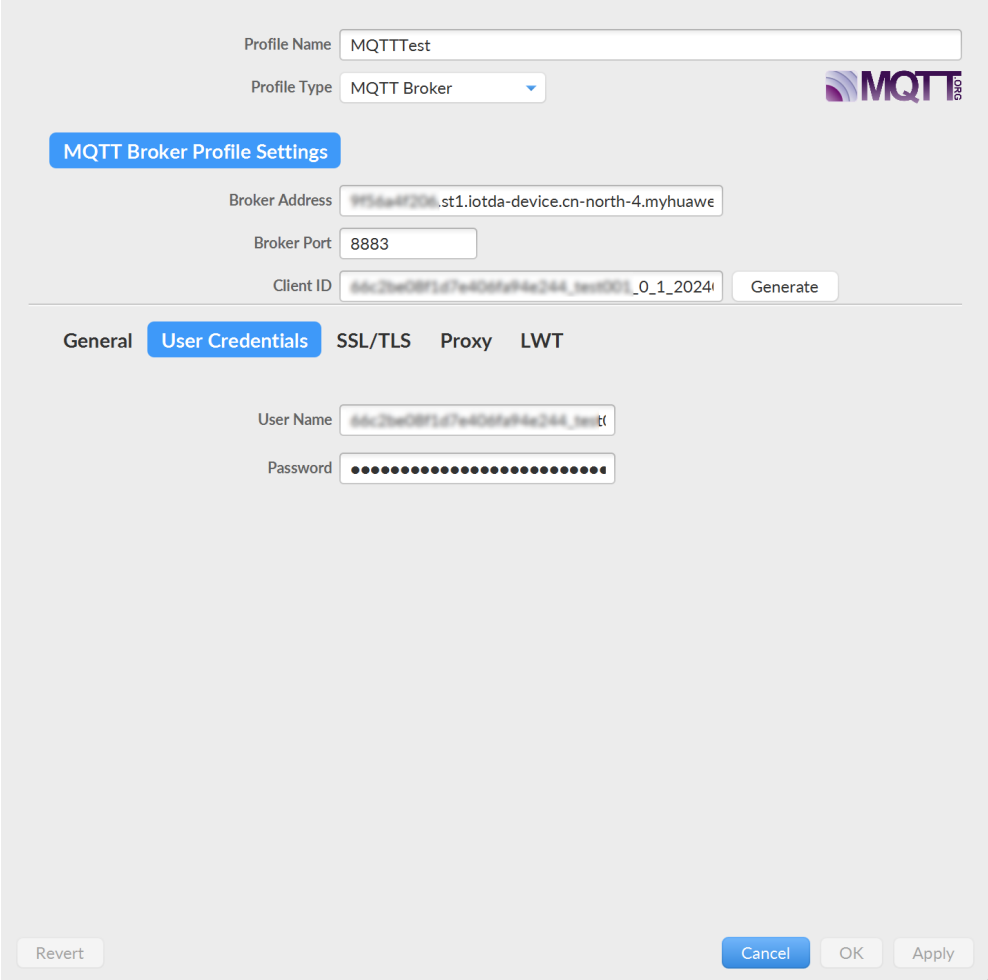
Figure 2-7 Message exchange process between MQTT.fx and IoTDA



Using MQTT.fx to Connect to IoTDA

1. Download **MQTT.fx** (64-bit OS) or **MQTT.fx** (32-bit OS) and install it.
2. Open the MQTT.fx client and choose **Extras** > **Edit Connection Profiles** from the menu bar.
3. On the **Edit Connection Profiles** page, set related parameters and click **OK**.

Figure 2-8 MQTT.fx connection parameters

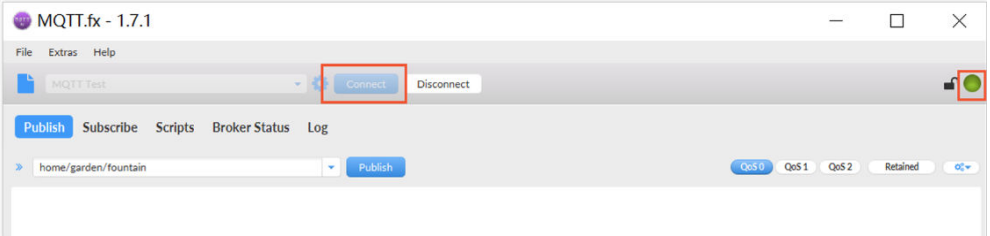


Parameter	Description	Example Value
Profile Name	Name of the configuration file.	Enter **MQTT Test**.
Profile Type	Type of the connection to be configured.	The value is fixed at **MQTT Broker**, indicating that the MQTT server is connected.
Broker Address	Access address of the MQTT server.	Access your instance, choose **Overview**, click **Access Details**, and obtain the MQTTS access address. For details, see **MQTTS access address**.
Broker Port	Access port of the MQTT server.	Enter **8883**.
Client ID	IoTDA can send and receive messages only after device access authentication is successful. For details about device authentication parameters, see **Device Connection Authentication**.	Go to the device details page, find **MQTT Connection Parameter**, and click **View** to check the clientId, username, and password.
User Name		
Password		
SSL/TLS		
Enable SSL/TLS	Whether to use the SSL or TLS encryption protocol.	Yes

Parameter	Description	Example Value
Protocol	Protocol version.	Select TLSv1.2.
CA certificate file	CA certificate file.	Obtain the CA certificate of the corresponding region from the **certificate resource** page.

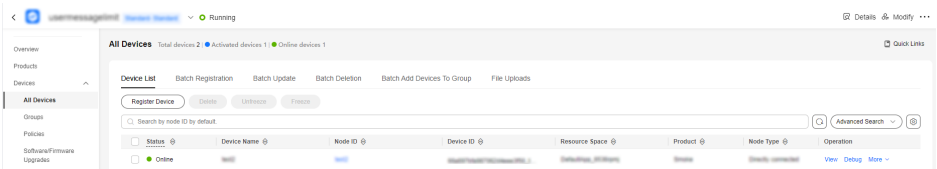
4. After setting the parameters, click **Connect**. If the icon in the upper right corner turns green, MQTT.fx has been connected to Huawei Cloud IoTDA. If the icon in the upper right corner turns red, the connection fails. Click the **Log** tab to check logs, modify the configuration based on the log information, and try again.

Figure 2-9 MQTT.fx connection



5. Access the **IoTDA** service page and click **Access Console**. In the navigation pane, choose **IoTDA Instances** and click the target instance card. In the navigation pane, choose **Devices** > **All Devices** to check the device status. The device status is expected to be online.

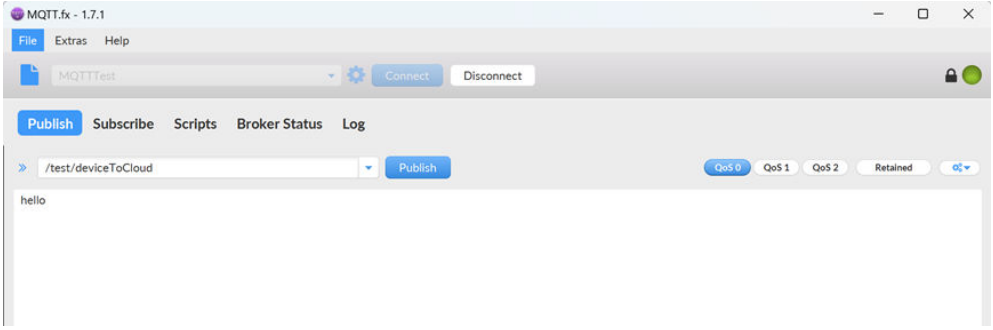
Figure 2-10 Device list - Device online status



Using MQTT.fx to Send Messages

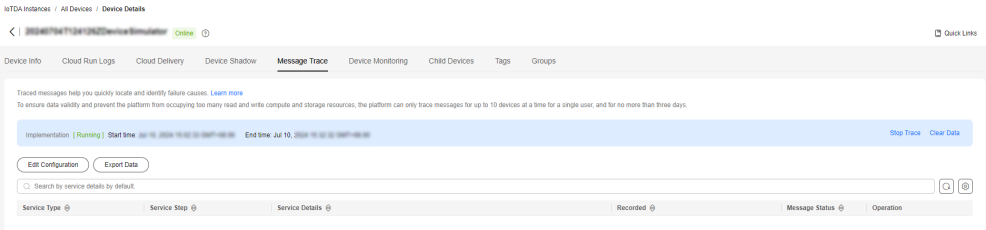
1. Click the **Publish** tab on the MQTT.fx client.
2. On the displayed tab page, enter the topic name in the **Topic** text box on the left, for example, **/test/deviceToCloud**. Enter the message content in the **Message** text box, for example, **hello**. Click **Publish** to send the message.

Figure 2-11 MQTT.fx message sending



3. Access the **IoTDA** service page and click **Access Console**. In the navigation pane, choose **IoTDA Instances** and click the target instance card. In the navigation pane, choose **Devices** > **All Devices**. On the displayed page, click **View**. On the **Message Trace** tab page, check the messages sent by MQTT.fx.

Figure 2-12 Message tracing - Viewing results

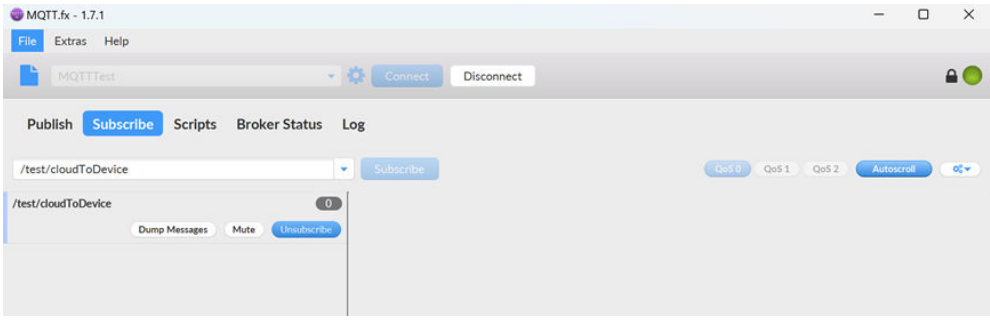


4. After MQTT.fx sends messages to the platform, configure **data forwarding rules** to forward the messages to message middleware, storage, data analysis, or service applications.

Using MQTT.fx to Receive Messages

1. Click the **Subscribe** tab on the MQTT.fx client.
2. On the displayed tab page, enter the topic name in the **Topic** text box on the left and click **Subscribe**. **/test/cloudToDevice** is used as an example. After the topic is subscribed to, check the topic in the subscription list.

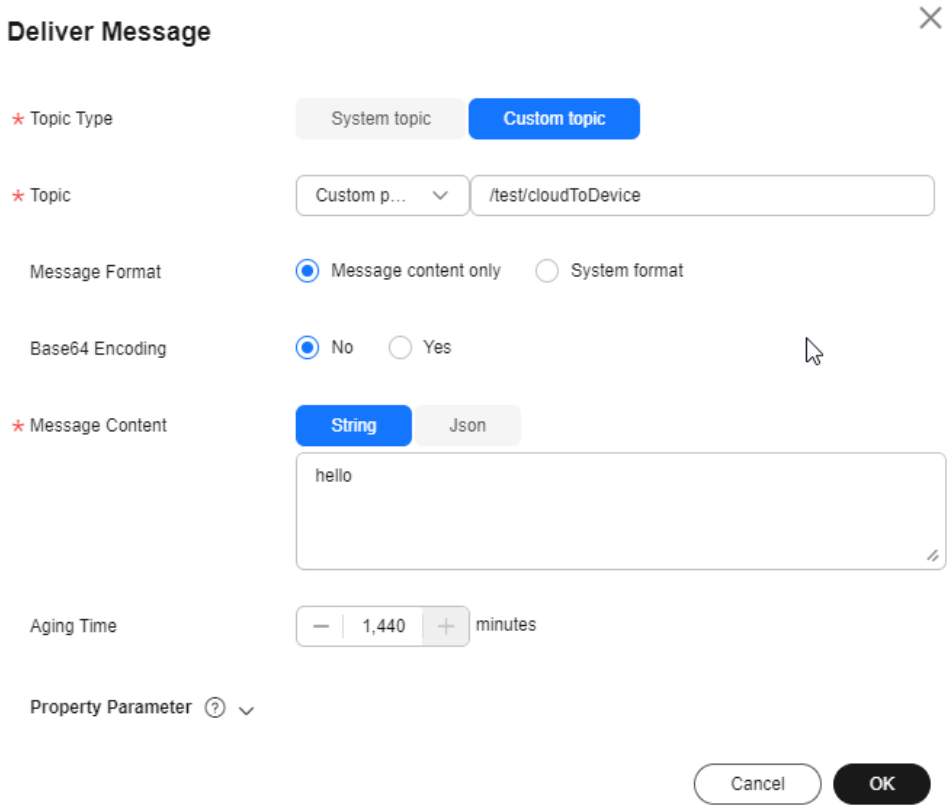
Figure 2-13 MQTT.fx subscription topic



3. Access the **IoTDA** service page and click **Access Console**. In the navigation pane, choose **IoTDA Instances** and click the target instance card. In the navigation pane, choose **Devices** > **All Devices**. On the device list, click a device to access its details page.

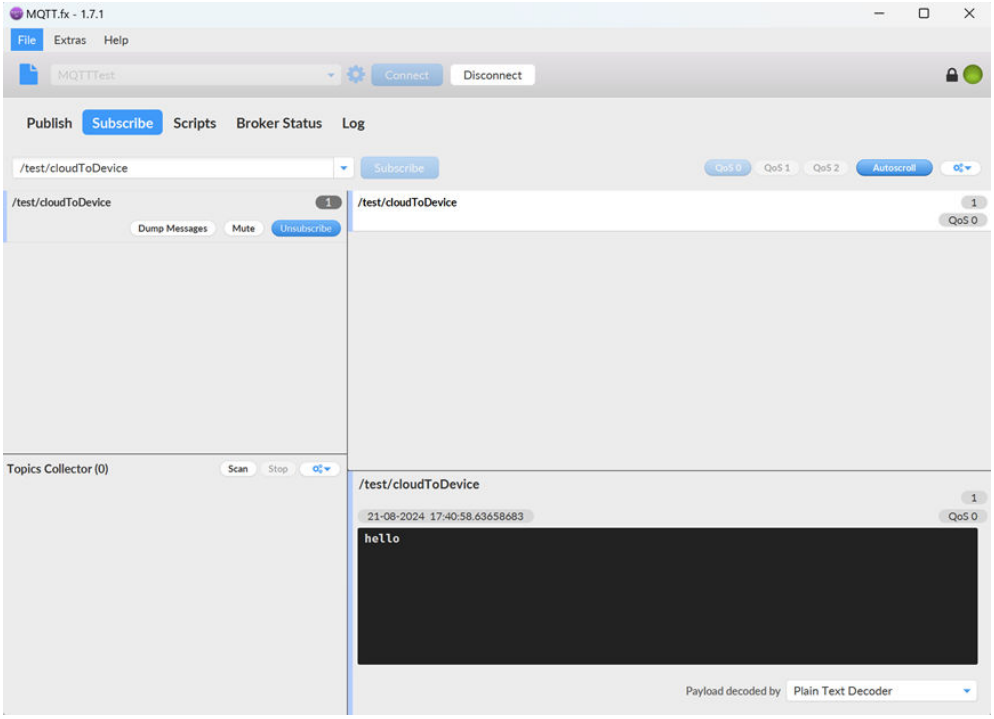
4. Click the **Cloud Delivery** tab. On the **Message Delivery** tab page, click **Deliver Message**. In the displayed dialog box, configure the content and the parameters for the message to deliver.

Figure 2-14 Delivering a message - Custom topic



5. On the MQTT.fx client, click the **Subscribe** tab. The message received from the subscribed topic is displayed.

Figure 2-15 Checking messages using MQTT.fx

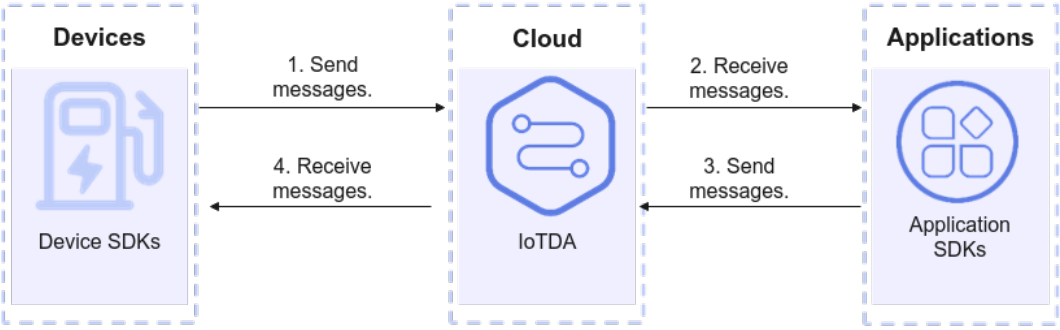


2.5 Using Device SDKs to Send and Receive Messages

Introduction

Devices can use **SDKs in multiple languages** to quickly connect to IoTDA for upstream and downstream message communications. This section uses Java sample code to demonstrate how a device accesses IoTDA using MQTTS/MQTT and publishes and subscribes to messages using topics.

Figure 2-16 Message exchange between the SDK and IoTDA



Device-side SDK Sending Messages

1. Configure the Maven dependency of the SDK on the device.

```
<dependency>
    <groupId>com.huaweicloud</groupId>
    <artifactId>iot-device-sdk-java</artifactId>
    <version>[1.2.1,)</version>
</dependency>
```


2. Connect the device to Huawei Cloud IoTDA.

   a. Modify the device connection parameters before establishing a connection by referring to the **sample code**.

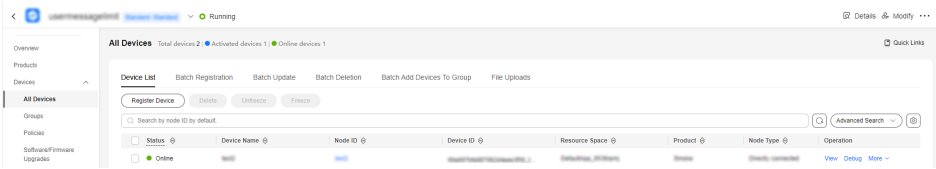
```
// Replace xxx.st1.iotda-device.cn-north-4.myhuaweicloud.com with the actual access address.
// To obtain the domain name, log in to the Huawei Cloud IoTDA console. In the navigation pane, choose Overview and click Access Details in the Instance Information area. Select the access address corresponding to port 8883.
IoTDevice device = new IoTDevice("ssl://xxx.st1.iotda-device.cn-north-4.myhuaweicloud.com:8883", "your device id",
        "your device secret", tmpCAFile);
device.getClient().setConnectListener(new MessageSample(device));

if (device.init() != 0) {
    return;
}
```

   b. Connect the device to the platform, and use it to subscribe to messages delivered by the platform.

```
    @Override
    public void connectComplete(boolean reconnect, String serverURI) {
        // Subscribe to downstream messages.
        device.getClient().subscribeTopic("/test/cloudToDevice", new ActionListener() {
            @Override
            public void onSuccess(Object context) {
                System.out.println("subscribeTopic success");
            }

            @Override
            public void onFailure(Object context, Throwable var2) {
                System.out.println("subscribeTopic failure");
            }
        } , rawMessage -> {
            System.out.println(" on receive message topic : " + rawMessage.getTopic() + " , payload : " + new String(
                rawMessage.getPayload(), StandardCharsets.UTF_8));
        }, 1);
    }
```

   c. Run the program. The device is displayed as online on the platform.

Figure 2-17 Device list - Device online status



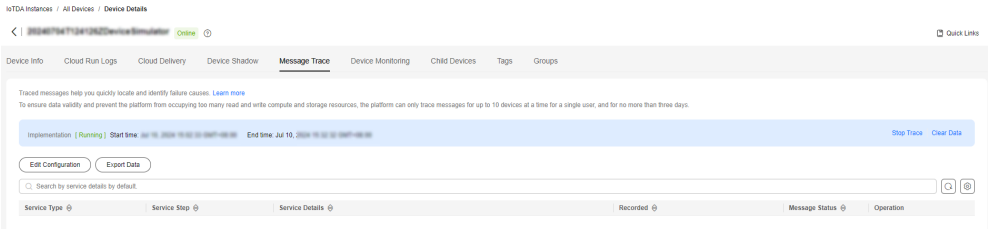
3. Specify a topic to report device messages.

```
device.getClient().publishRawMessage(new RawMessage("/test/deviceToCloud", "hello", 1), new ActionListener() {
    @Override
    public void onSuccess(Object context) {
        System.out.println("reportDeviceMessage success: ");
    }
    @Override
    public void onFailure(Object context, Throwable var2) {
        System.out.println("reportDeviceMessage fail: " + var2);
    }
});
```

4. Access the **IoTDA** service page and click **Access Console**. In the navigation pane, choose **IoTDA Instances** and click the target instance card. In the navigation pane, choose **Devices** > **All Devices** and click **View**. On the

Message Trace tab page, check whether the IoT platform receives the message.

Figure 2-18 Message tracing - Viewing results



Application-side SDK Receiving Messages

After a device sends messages to the platform through the SDK, you can configure **data forwarding rules** to forward the messages to the message middleware, storage, data analysis, or service applications. This section uses **Java SDK** as an example to describe how to receive messages reported by devices and process services.

Application-side SDK Delivering Messages

Configure the SDK on the application side to deliver messages.

1. Configure the Maven dependency. In this example, the development environment is JDK 1.8 or later. **Download an SDK**.

```
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-core</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
<dependency>
    <groupId>com.huaweicloud.sdk</groupId>
    <artifactId>huaweicloud-sdk-iotda</artifactId>
    <version>[3.0.40-rc, 3.2.0)</version>
</dependency>
```

2. The following is an example of a message sent by the application to a specific device:

```
public class MessageDistributionSolution {
    // REGION_ID: If CN East-Shanghai1 is used, enter cn-east-3. If CN North-Beijing4 is used, enter
cn-north-4. If CN South-Guangzhou is used, enter cn-south-4.
    private static final String REGION_ID = "<YOUR REGION ID>";
    // ENDPOINT: On the console, choose Overview and click Access Addresses to view the HTTPS
application access address.
    private static final String ENDPOINT = "<YOUR ENDPOINT>";
    // For the standard or enterprise edition, create a region object.
    public static final Region REGION_CN_NORTH_4 = new Region(REGION_ID, ENDPOINT);
    public static void main(String[] args) {
        String ak = "<YOUR AK>";
        String sk = "<YOUR SK>";
        String projectId = "<YOUR PROJECTID>";
        // Create a credential.
        ICredential auth = new
BasicCredentials().withDerivedPredicate(AbstractCredentials.DEFAULT_DERIVED_PREDICATE)
            .withAk(ak)
            .withSk(sk)
            .withProjectId(projectId);
        // Create and initialize an IoTDAClient instance.
        IoTDAClient client = IoTDAClient.newBuilder().withCredential(auth)
            // For the basic edition, select the region object in IoTDARegion.
```



```
            //.withRegion(IoTDARegion.CN_NORTH_4)
            // For the standard or enterprise edition, create a region object.
            .withRegion(REGION_CN_NORTH_4).build();
        // Instantiate a request object.
        CreateMessageRequest request = new CreateMessageRequest();
        request.withDeviceId("<YOUR DEVICE_ID>");
        DeviceMessageRequest body = new DeviceMessageRequest();
        body.withPayloadFormat("raw");
        body.withTopicFullName("/test/cloudToDevice");
        body.withMessage("hello");
        request.withBody(body);
        try {
            CreateMessageResponse response = client.createMessage(request);
            System.out.println(response.toString());
        } catch (ConnectionException e) {
            e.printStackTrace();
        } catch (RequestTimeoutException e) {
            e.printStackTrace();
        } catch (ServiceResponseException e) {
            e.printStackTrace();
            System.out.println(e.getHttpStatusCode());
            System.out.println(e.getRequestId());
            System.out.println(e.getErrorCode());
            System.out.println(e.getErrorMsg());
        }
    }
}
```

3. Check the device logs. The message sent by the application to the device is displayed as follows:

```
on receive message topic : /test/cloudToDevice , payload : hello
```


3 Quick Application Access

The IoT platform provides various APIs to reduce the application development difficulty and improve the application development efficiency. This topic uses local debugging (Postman) as examples to describe how to connect an application to IoTDA using HTTPS.

Local Debugging

This section uses Postman to describe how to use IoTDA by calling APIs on the application side.

The procedure is as follows:

Step 1 Enable IoTDA. Access the **IoTDA** service page, and click **Access Console** to subscribe to the service.

Step 2 Create an MQTT product.

Step 3 Configure the environment. Download and install Postman 7.17.0.

Step 4 Call the service. Use Postman to call the API and check the returned result, status code, and error code.

- **Step 1 Enable IoTDA.**

  Currently, IoTDA is available in AP-Bangkok, AP-Singapore, CN-Hong Kong, and AF-Johannesburg regions.

- **Step 2 Create a product.**

  Create a product on IoTDA before calling APIs.

  a. Access the **IoTDA** service page and click **Access Console**. Click the target instance card.

  b. Choose **Products** in the navigation pane and click **Create Product**.

  c. Set the parameters to create a product that uses MQTT, and click **OK**.

Basic Information

Resource Space	The platform automatically allocates the created product to the default resource space. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create one.
Product Name	Customize the product name. The value can contain up to 64 characters. Only letters, digits, and special characters (_?'#().,&%@!-) are allowed.
Protocol	MQTT is recommended.
Data Type	Select **JSON**.
Industry	Set this parameter as required.
Sub-industry	Set this parameter as required.
Device Type	Set this parameter as required.
Advanced Settings	
Product ID	Set a unique identifier for the product. If this parameter is specified, IoTDA uses the specified product ID. If this parameter is not specified, IoTDA allocates a product ID.
Description	Provide a description for the product. Set this parameter as required.

- **Step 3 Configure the environment.**

  Download and install Postman. For details, see **Installing and Configuring Postman**.

- **Step 4 Call the service.**

  After configuring Postman, debug the following APIs when the application simulator connects to IoTDA using HTTPS:

  - **Obtaining the Token for an IAM User**
  - **Listing Projects Accessible to an IAM User**
  - **Creating a Product**
  - **Querying a Product**
  - **Creating a Device**
  - **Querying a Device**

Advanced Experience

After using Postman to connect a simulated application to the platform, you may understand how the application interacts with the platform through open APIs.

To better experience the IoTDA service, develop real-world applications and devices and connect them to the platform. For details, see **Developer Guide**.

