

## IoT Device Access

# Best Practices

**Issue** 1.0  
**Date** 2024-11-04



**Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.**

No part of this document may be reproduced or transmitted in any form or by any means without prior written consent of 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

---

<b>1 Introduction.....</b>	<b>1</b>
<b>2 Device Access.....</b>	<b>6</b>
2.1 Developing an MQTT-based Simulated Smart Street Light Online.....	6
2.2 Developing a Smart Street Light Using NB-IoT BearPi.....	25
2.3 Developing a Smart Smoke Detector Using NB-IoT BearPi.....	41
2.4 Connecting and Debugging an NB-IoT Smart Street Light Using a Simulator.....	58
2.5 Developing a Protocol Conversion Gateway for Access of Generic-Protocol Devices.....	83
2.6 Connecting a Device That Uses the X.509 Certificate Based on MQTT.fx.....	86
2.7 Connecting to IoTDA Based on the BearPi-HM_Nano Development Board and OpenHarmony 3.0...	106
2.8 Testing MQTT Performance Using JMeter.....	115
<b>3 Device Management.....</b>	<b>122</b>
3.1 Automatically Adjusting Air Conditioner Temperature Through Device Shadow.....	122
3.2 Managing Indoor Air Conditioners Using Custom Topics.....	126
3.3 Performing OTA Firmware Upgrade for MQTT Devices.....	129
<b>4 Data Forwarding.....</b>	<b>141</b>
4.1 Pushing Metric Data to DMS for Kafka.....	141
<b>5 Device Linkage.....</b>	<b>144</b>
5.1 Triggering Alarms and Sending Email or SMS Notifications.....	144
5.2 Automatic Device Shutdown Upon High Temperature.....	152
5.3 Automatically Opening the Window upon High Gas Concentration.....	157
5.4 Monitoring Device Status Changes and Sending Notifications.....	164

# 1 Introduction

---

After you have a basic understanding of IoTDA, you may wonder how the platform can create value for you, in which business scenarios the platform can be used, and how you can access the platform. The following scenario examples are used to describe the service process and product model as well as the platform functions and benefits.

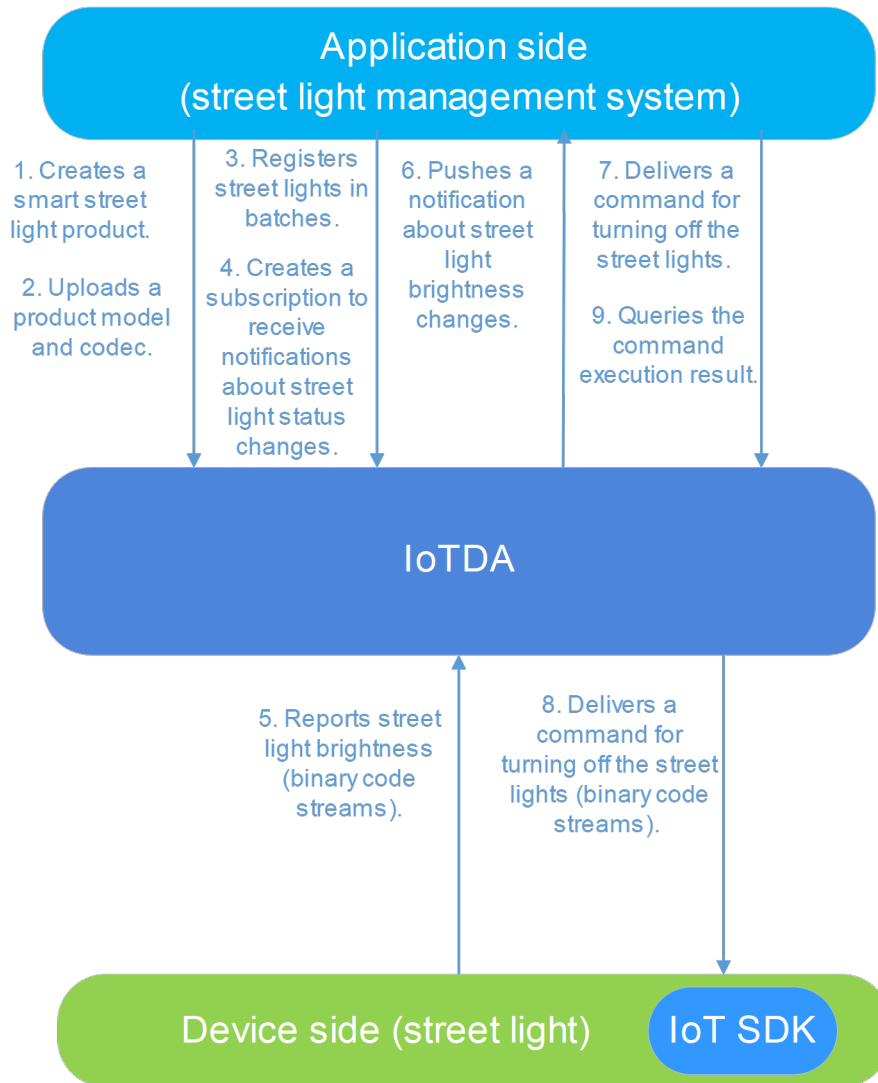
## Scenario Example: Smart Street Light

The street light management system connects to the platform to monitor street lights that are integrated with the NB-IoT module and turn on/off these street lights.

In this scenario, a device interacts with the platform using LwM2M. The application subscribes to device change notifications on the platform and delivers commands to the device.

**Key points:** product model, codec, subscription and push, property reporting, and command delivery

Reference: [Developing a Smart Street Light Using NB-IoT BearPi](#)



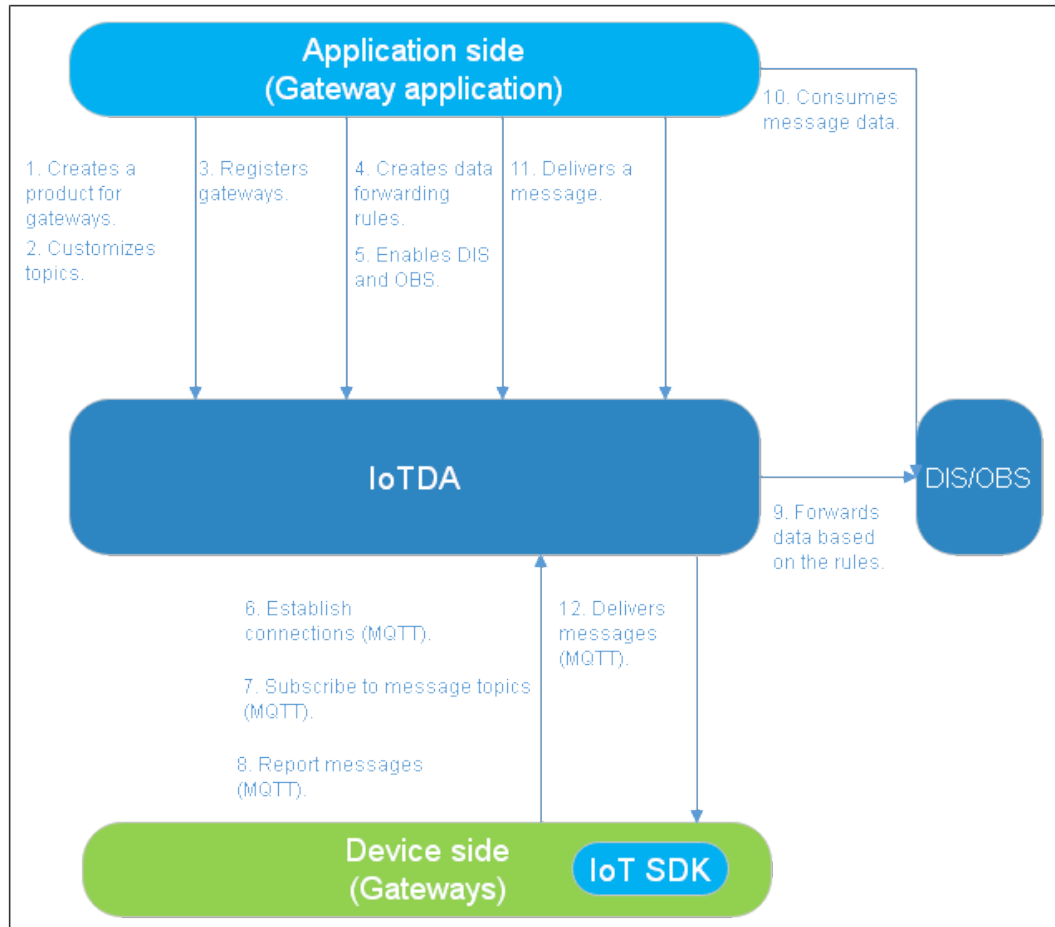
### Scenario Example: Smart Gateway

Using gateways, you can manage existing devices under the gateways without migration and add new devices to the gateways.

In this scenario, devices (gateways) interact with the platform using the MQTT protocol. You can create topics on the product details page of the console and create data forwarding rules using application APIs or the console to forward device messages to other Huawei Cloud services for consumption.

**Key points:** product model, message reporting, message delivery, MQTT, data forwarding rules, and topic customization

Reference: [Managing Indoor Air Conditioners Using Custom Topics](#)



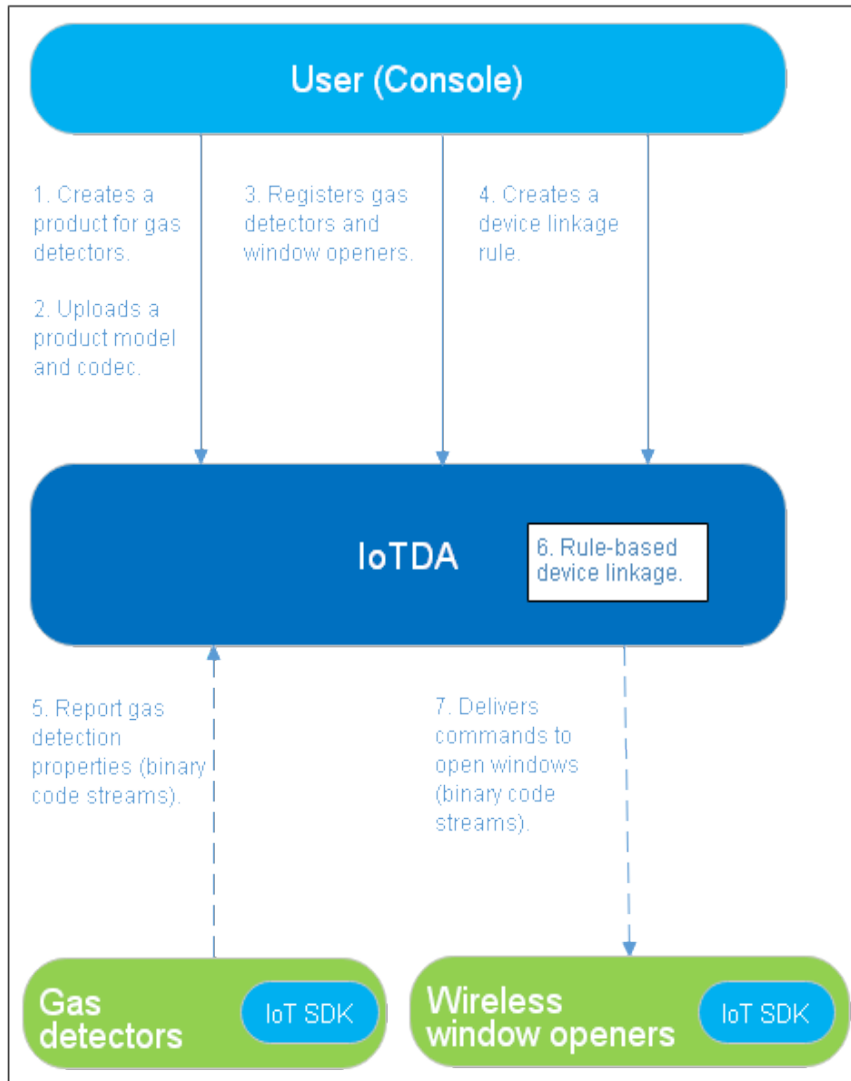
## Scenario Example: Smart Home Gas Detection

If a gas detector detects excessive gas, the wireless window opener associated with the gas detector automatically opens the window for ventilation.

In this scenario, devices interact with the platform over MQTT to report properties. You can create device linkage rules on the console or by calling APIs to convert the reported properties into commands and deliver the commands to other specific devices.

**Key points:** product model, property reporting, command delivery, MQTT, and device linkage rules

For details about this scenario, see [Automatically Opening the Window upon High Gas Concentration](#).



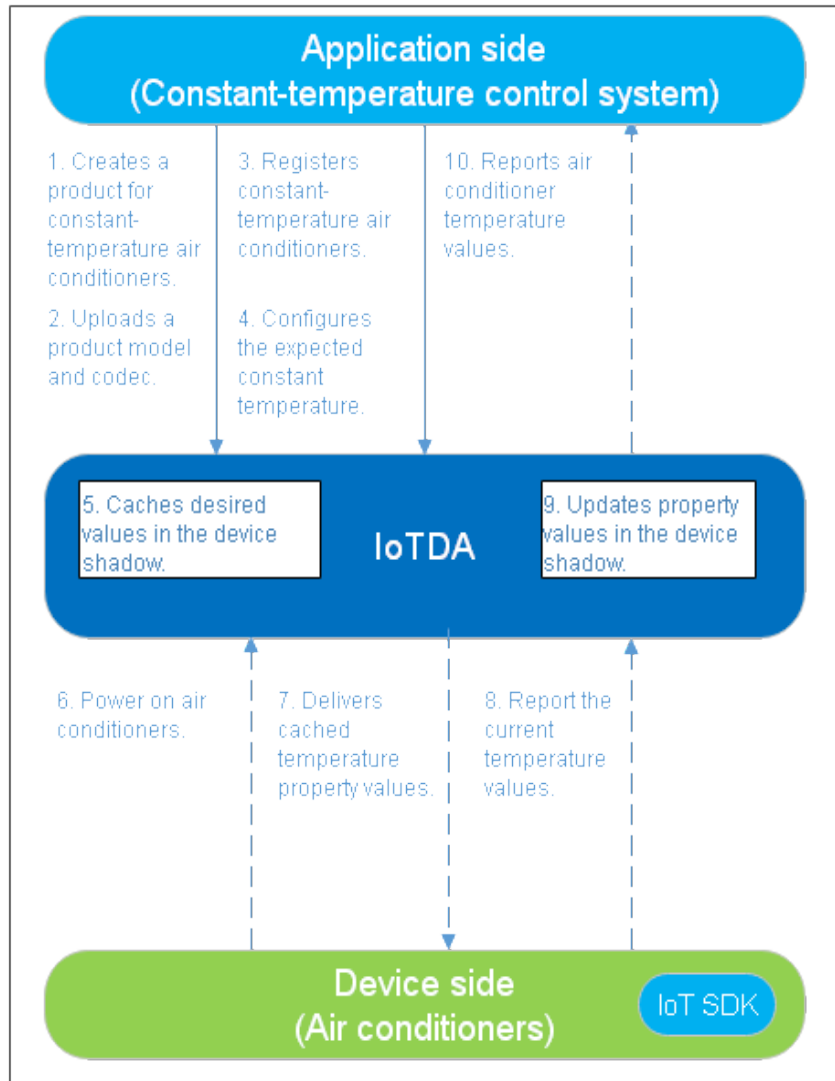
### Scenario Example: Constant-Temperature Air Conditioner

Using a constant-temperature control system, you can adjust the default temperature of air conditioners (regardless of whether they are powered on). After being powered on, the air conditioners automatically run at the default temperature.

In this scenario, the application or console delivers property pending commands to offline devices. If devices go online and report different properties, the console automatically delivers commands to modify device properties until they are the same as the desired values.

**Key points:** product model, codec, device shadow, property reporting, and property modification

Reference: [Automatically Adjusting Air Conditioner Temperature Through Device Shadow](#)





# 2 Device Access

---

## 2.1 Developing an MQTT-based Simulated Smart Street Light Online

### Scenarios

This section uses a smart street light as an example to describe how to use device simulators to experience data reporting and command delivery. You can use MQTT.fx (recommended) or MQTT\_Simulator.

A street light reports the light intensity (luminance) in JSON format. The command (switch) can be used to remotely control the street light status.

### Prerequisites

- You have [registered a Huawei Cloud account](#).
- You have subscribed to IoTDA. If you have not subscribed to the service, go to the [IoTDA](#) service page, and click **Access Console** to subscribe to the service.

### Service Flow

The MQTT.fx simulator is used as an example to describe data reporting and command delivery.

**Step 1** [Create an MQTT product](#).

**Step 2** [Develop a product model](#). Define a product model to develop a street light that supports light intensity reporting and status control commands.

**Step 3** [Register an MQTT device](#) to experience data reporting.

**Step 4** [Perform connection authentication](#). Use MQTT.fx to activate the device registered on IoTDA.

**Step 5** [Report data](#). Use MQTT.fx to report data to IoTDA.

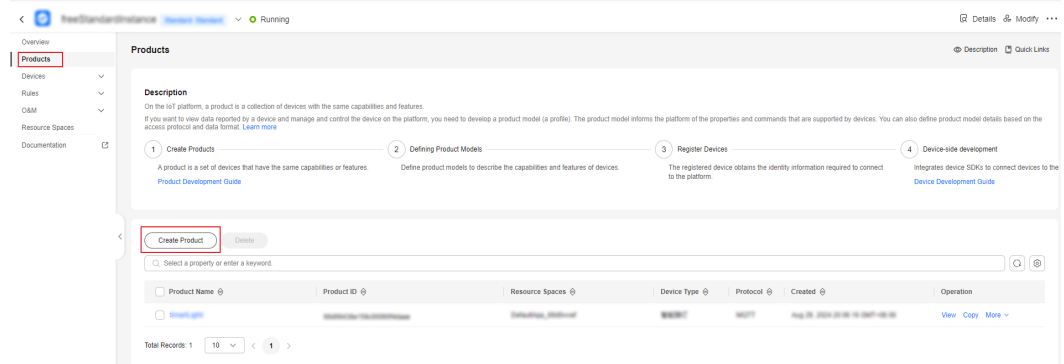
**Step 6** [Deliver a command](#) on the console to remotely control a device.

----End

## Creating a Product

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

**Figure 2-1** 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 2-2 Creating a product - MQTT

**Create Product** ×

\* Resource Space ?

To create a new resource space, you can [go to the instance details page](#).

\* Product Name

Protocol ?

\* Data Type ?

Device Type Selection

\* Device Type ?

Advanced Settings ^ Custom Product ID | Description

Product ID ?

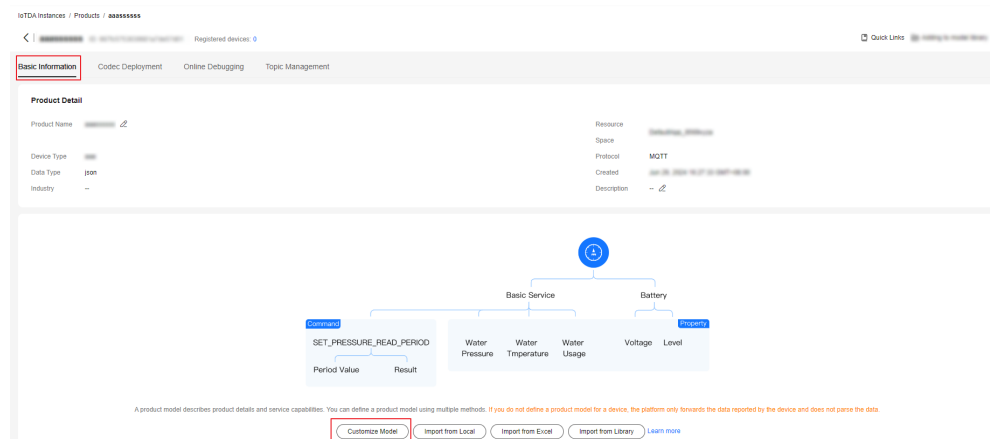
Description   
0/128 ↗

----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.

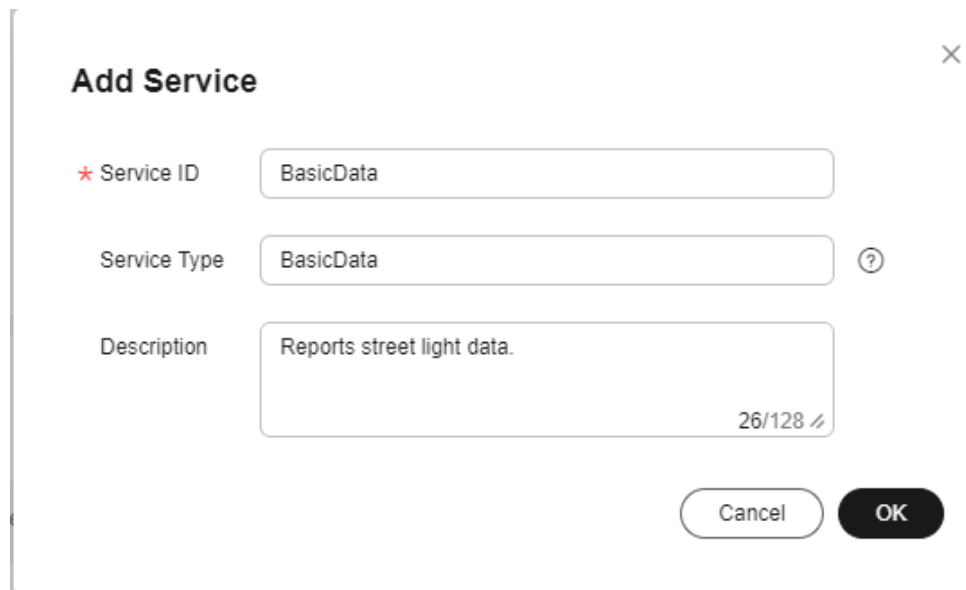
**Figure 2-3** Custom model - MQTT



**Step 3** Add the **BasicData** service.

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

**Figure 2-4** Adding a service - BasicData



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

Figure 2-5 Adding a property - luminance

**Add Property** ×

★ Property Name

Description  16/128 ↗

★ Data Type  ▼

★ Access Permissions

★ Value Range  –

Step

Unit

**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 2-6 Adding a command - Switch

**Add Command**

\* Command

Name

Command

Parameters

Parameter Name	Data Type	Description	Operation
<b>No table data available.</b> No Command Parameters data available. Add Command Parameter first.			

Total Records: 0  < 1 >

Response

Parameters

Parameter Name	Data Type	Description	Operation
<b>No table data available.</b> No Response Parameters data available. Add Response Parameter first.			

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

Figure 2-7 Adding a command parameter - -value

✕

### Add Parameter

★ Parameter Name

Description 0/128 ↗

★ Data Type

★ Length

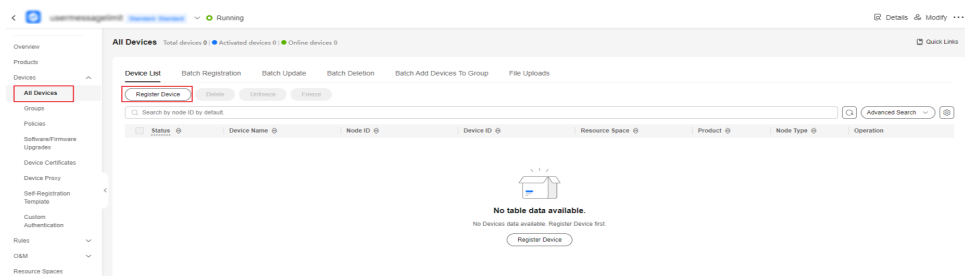
Enumerated Values 6/1,024 ↗

----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 2-8 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.

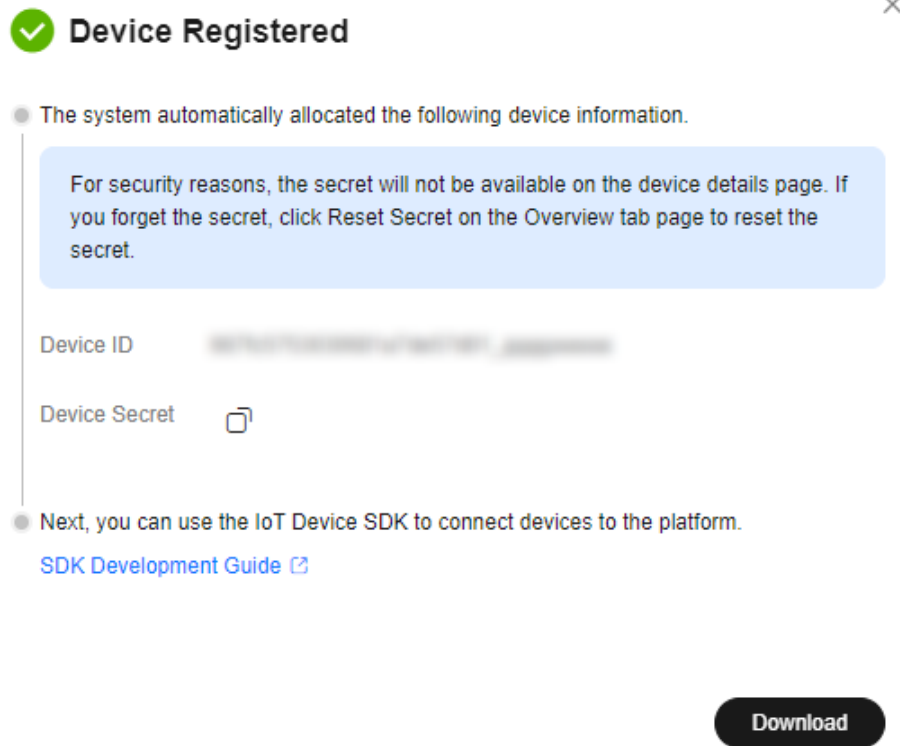
Parameter	Description
Node ID	Customize a unique physical identifier for the device. The value consists of letters and digits.
Device Name	Customize the device name.
Authentication Type	Select <b>Secret</b> .
Secret	If you do not set this parameter, IoTDA automatically generates a value.

**Figure 2-9** 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 2-10 Device registered

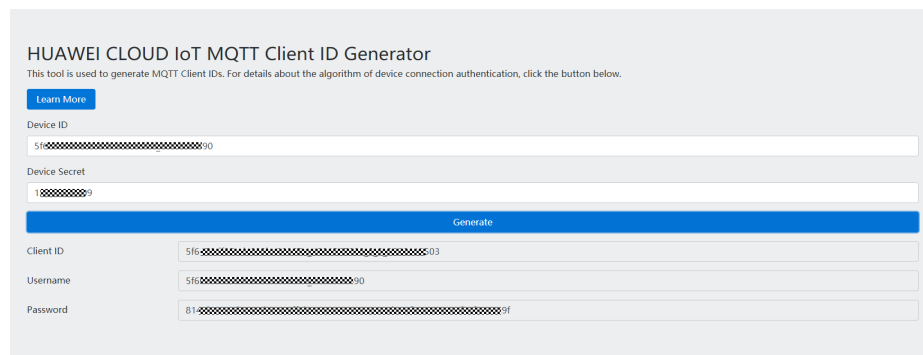


----End

## Performing Connection Authentication

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 [IoTDA client ID generator page](#), enter the device ID and secret generated after registering a device to generate connection information (including **ClientId**, **Username**, and **Password**).



Parameter	Mandatory	Type	Description
ClientId	Yes	String(256)	<p>The value of this parameter consists of a device ID, device type, password signature type, and timestamp. They are separated by underscores (_).</p> <ul style="list-style-type: none"> <li>• Device ID: A device ID uniquely identifies a device and is generated when the device is registered with IoTDA. The value usually consists of a device's product ID and node ID which are separated by an underscore (_).</li> <li>• Device type: The value is fixed at <b>0</b>, indicating a device ID.</li> <li>• Password signature type: The length is 1 byte, and the value can be <b>0</b> or <b>1</b>. <ul style="list-style-type: none"> <li>- <b>0</b>: The timestamp is not verified using the HMAC-SHA256 algorithm.</li> <li>- <b>1</b>: The timestamp is verified using the HMAC-SHA256 algorithm.</li> </ul> </li> <li>• Timestamp: The UTC time when the device was connected to IoTDA. The format is YYYYMMDDHH. For example, if the UTC time is 2018/7/24 17:56:20, the timestamp is <b>2018072417</b>.</li> </ul>
Username	Yes	String(256)	Device ID.
Password	Yes	String(256)	<p>Encrypted device secret. The value of this parameter is the device secret encrypted by using the HMAC-SHA256 algorithm with the timestamp as the key.</p> <p>The device secret is returned by IoTDA upon successful device registration.</p>

Each device performs authentication using the MQTT CONNECT message, which must contain all information of the client ID. After receiving a CONNECT message, IoTDA checks the authentication type and password digest algorithm of the device.

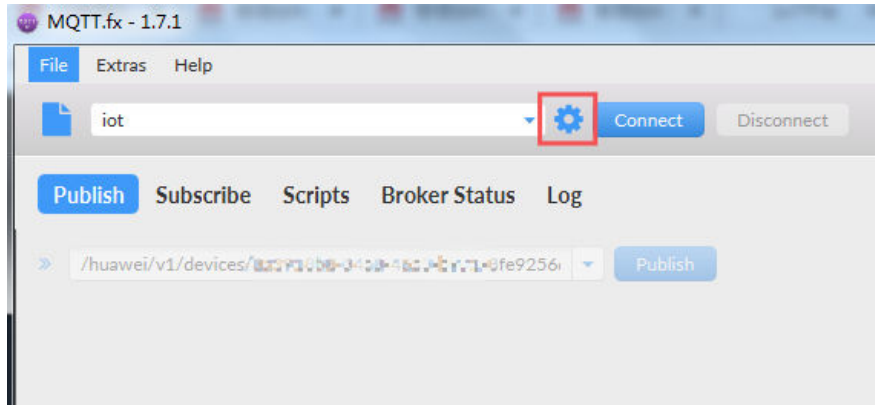
The generated client ID is in the format ***Device ID\_0\_0\_Timestamp***. By default, the timestamp is not verified.

- If the timestamp needs to be verified using the HMAC-SHA256 algorithm, the platform checks whether the message timestamp is consistent with the platform time and then checks whether the password is correct.
- If the timestamp does not need to be verified using the HMAC-SHA256 algorithm, the timestamp must also be contained in the CONNECT message,

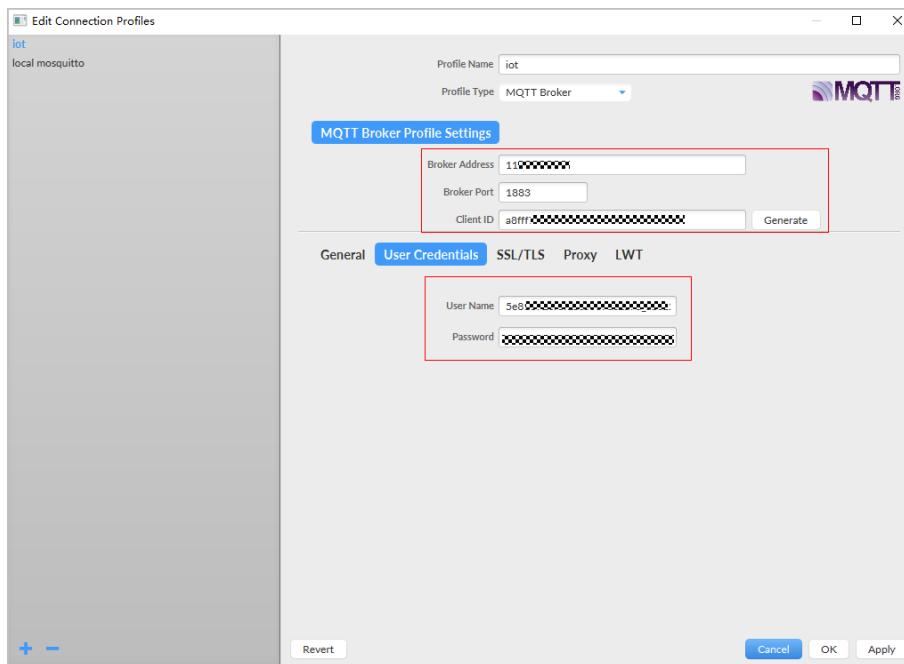
but the platform does not check whether the time is correct. In this case, only the password is checked.

If the authentication fails, the platform returns an error message and automatically disconnects the MQTT connections.

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



**Step 4** Configure authentication parameters and click **Apply**.

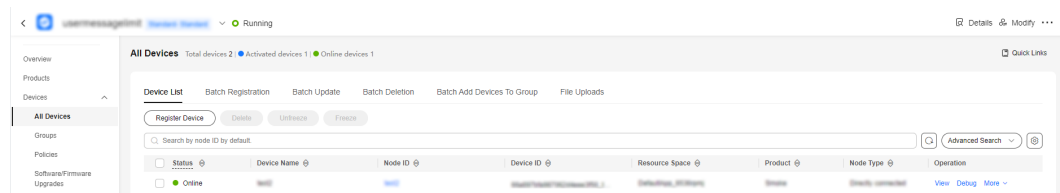


Parameter	Description
Broker Address	Enter the <b>device access address</b> (domain name) obtained from the IoTDA console. For devices that cannot be connected to the platform using a domain name, run the <b>ping Domain name</b> command in the CLI to obtain the IP address. The IP address is variable and needs to be set using a configuration item.
Broker Port	The default value is <b>1883</b> .

Parameter	Description
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 **Connect**. If the device authentication is successful, the device is displayed online on the platform.

**Figure 2-11** Device online status

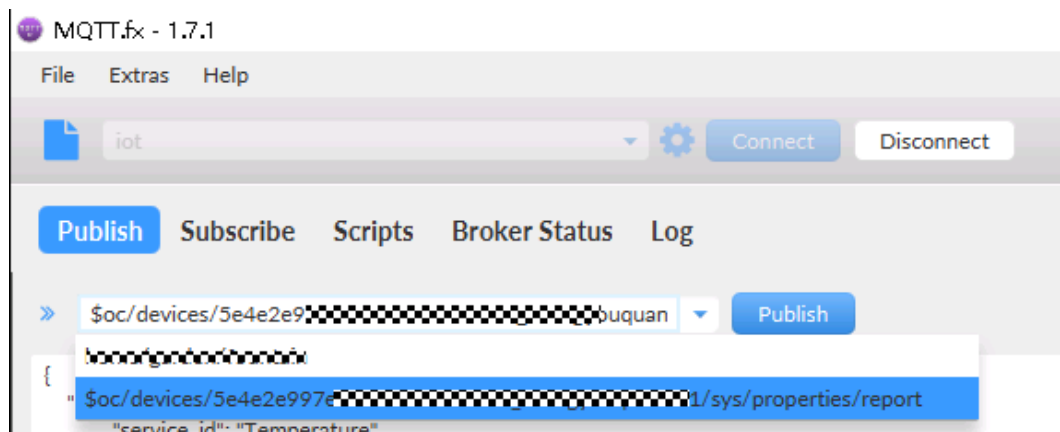


----End

## Reporting Data

Use MQTT.fx to report data to IoTDA. 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 address in the format of "`$oc/devices/{device_id}/sys/properties/report`", for example, `$oc/devices/5e4e2e92ac-164aefa8fouquan1/sys/properties/report`.



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

**Table 2-1** Service data list

Field Name	Mandatory	Data Type	Description
services	Yes	List<ServiceProperty>	Service data list. (For details, see the <b>ServiceProperty</b> structure below.)

**Table 2-2** ServiceProperty structure

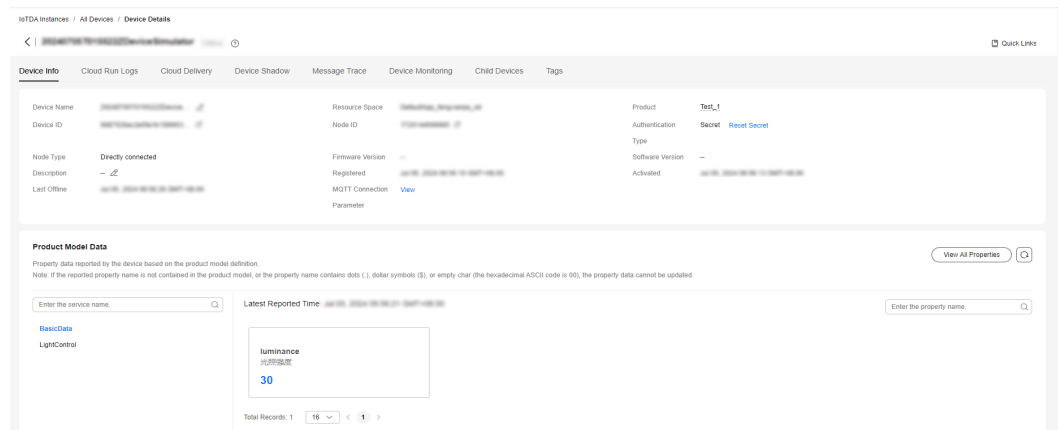
Field Name	Mandatory	Data Type	Description
service_id	Yes	String	Service ID.
properties	Yes	Object	Service properties, which are defined in the product model associated with the device.
eventTime	No	String	UTC time when the device reports data. The format is yyyyMMddTHH:mm:ssZ, for example, <b>20161219T11:49:20Z</b> . If this parameter is not carried in the reported data or is in incorrect format, the time when IoTDA receives the data is used.

**Example request**

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

**Step 3** Check whether the device successfully reports data on the device details page.

**Figure 2-12** Viewing reported data - MQTT



----End

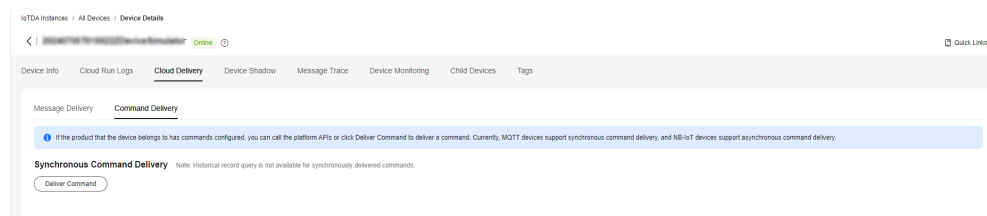
## Delivering a Command

Deliver a command on the console to remotely control a device.

**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 **Commands** tab and deliver a synchronization command.

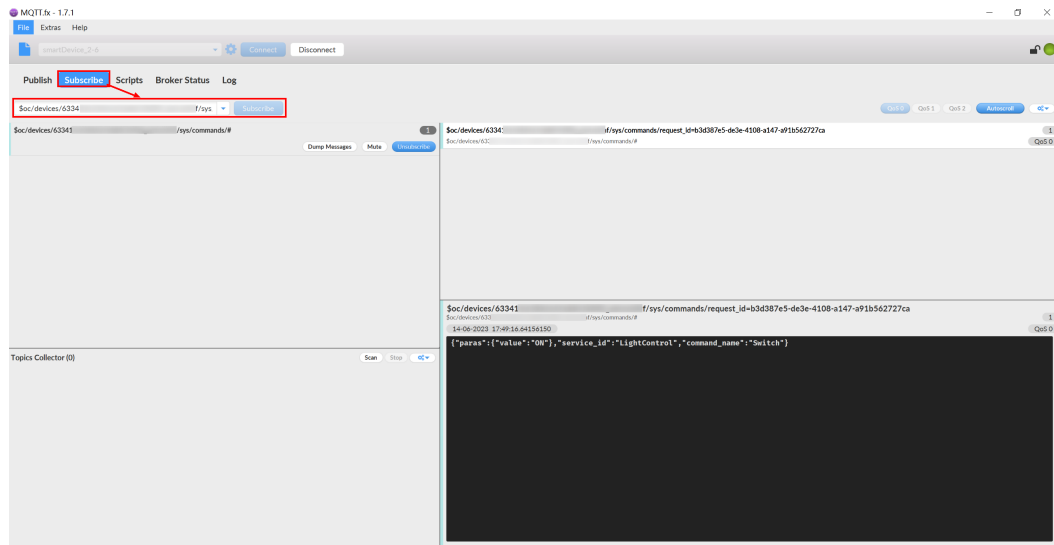
**Figure 2-13** Command delivery - MQTT



### NOTE

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

**Step 3** In the MQTT.fx simulator, select **Subscribe** and enter the command delivery topic. After the subscription, you can view the delivered command parameters.



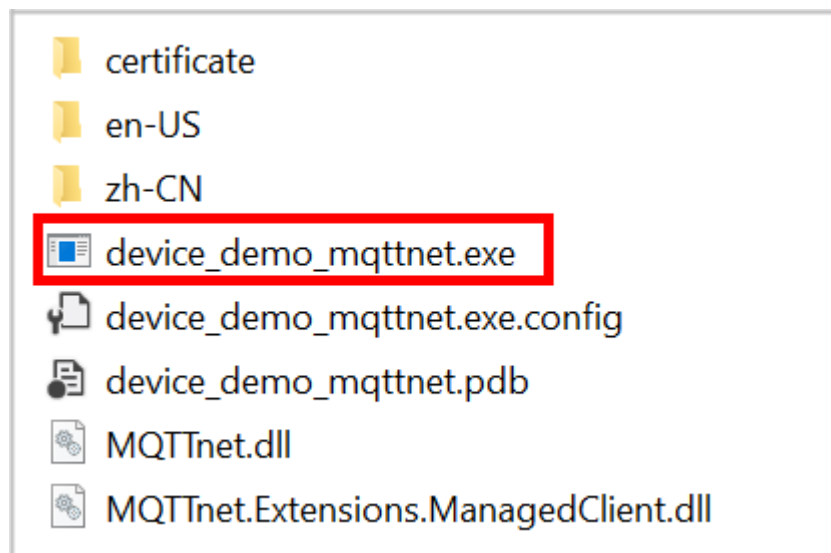
**NOTE**

- Use the MQTT.fx simulator to view the delivered command parameters. The command delivery topic is in the format of `$oc/devices/{device_id}/sys/commands/#`, where `{device_id}` indicates the value of device ID returned after the device is registered successfully.
- If the system displays a message indicating that the command delivery fails, the device needs to respond to the synchronization command in a timely manner. For details about the response content, see [Platform Delivering a Command](#).

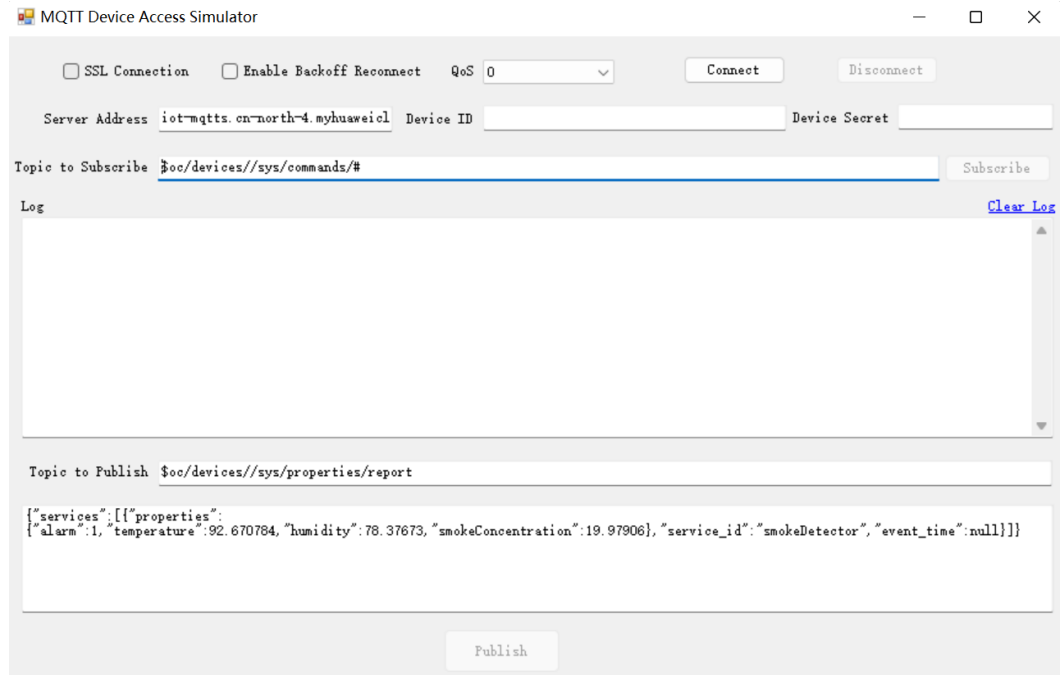
----End

## Using MQTT\_Simulator for Access

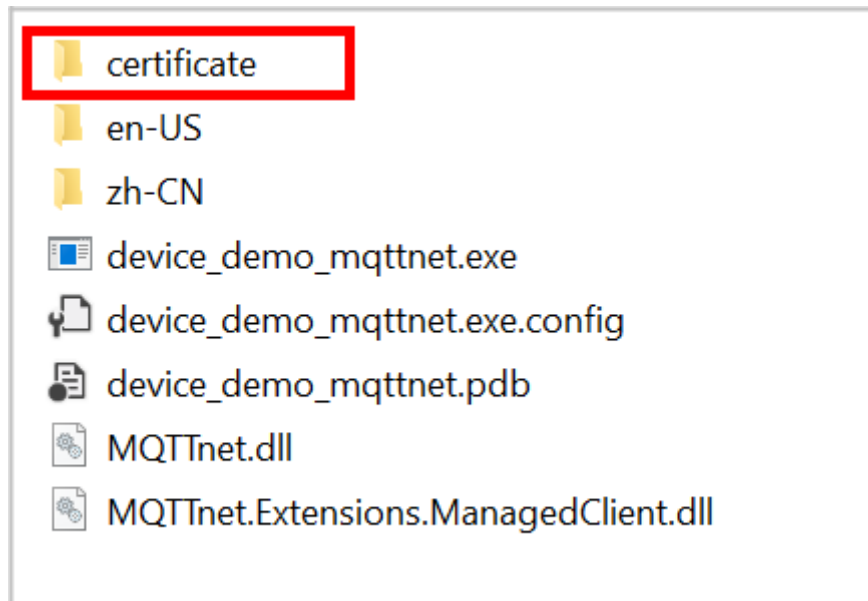
**Step 1** Download the [MQTT\\_Simulator simulator](#) (for 64-bit operating system by default) and start it.



**Step 2** Perform operations on the UI.



1. On the simulator UI, enter the server address, device ID, and device secret. Set the parameters based on the actual device information.
  - **Server Address:** domain name. Obtain it by referring to [Platform Connection Information](#).
  - **Device ID** and **Device Secret:** Obtain them from [here](#).
2. Use the corresponding [certificates](#) together with different server addresses during SSL-encrypted access. Obtain certificates by referring to [Obtaining Resources](#) and replace certificates in the **certificate** folder.



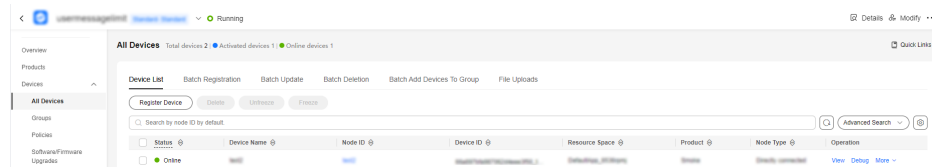
3. Select SSL encryption or no encryption when establishing a connection on the device side and set the QoS mode to **0** or **1**. Currently, QoS 2 is not supported. For details, see [Constraints](#).



**Step 3** Establish a connection.

To connect a device or gateway to the platform, upload the device information to bind the device or gateway to the platform. Click **Connect**. If the domain name, device ID, and secret are correct, a device connection success is displayed in the log. Check the device status on IoTDA, as shown in the following figure.

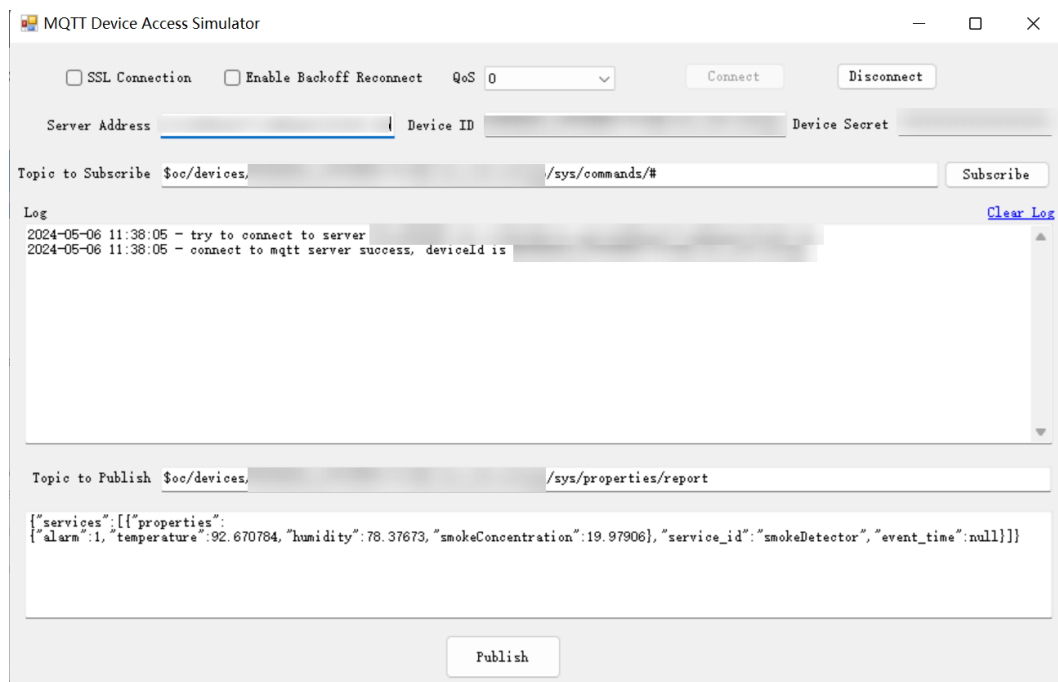
**Figure 2-14** Device list - Device online status



**Step 4** Subscribe to a topic.

Only devices that subscribe to a specific topic can receive messages about the topic published by the broker. For details on the preset topics, see [Topics](#).

After the connection is established and a topic is subscribed to, the following information is displayed in the log area on the home page of the demo.

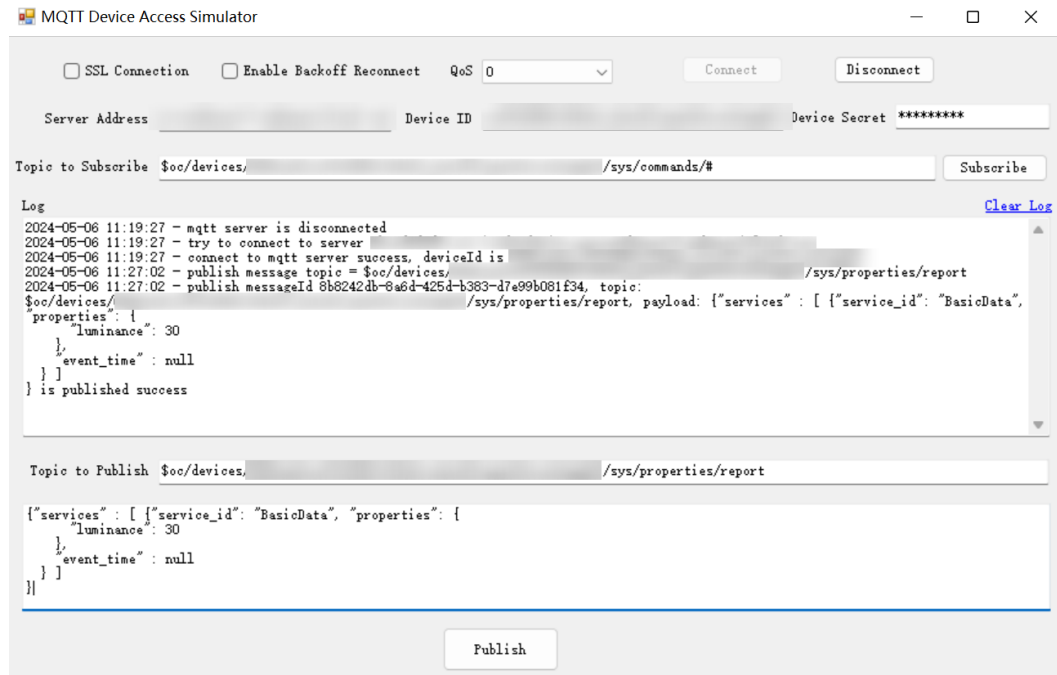


**Step 5** Publish a topic.

Publishing a topic means that a device proactively reports its properties or messages to the platform. For details, see the API [Device Reporting Properties](#).

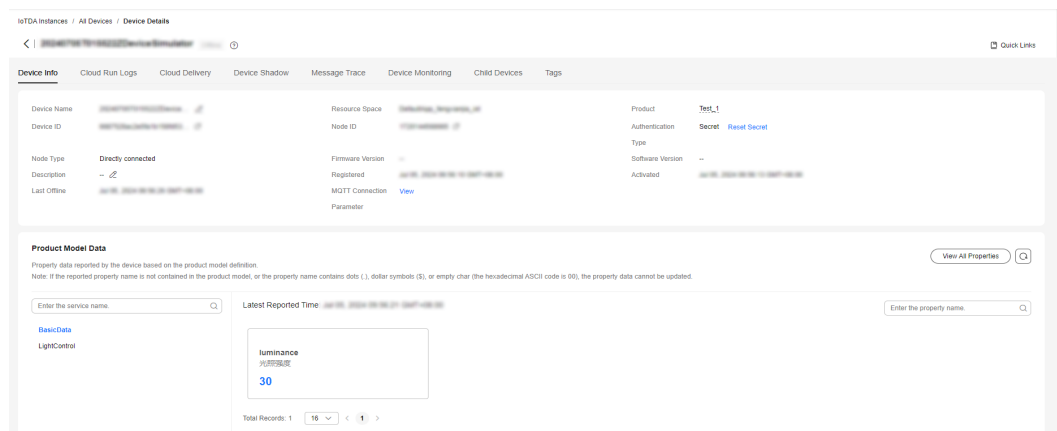
The simulator implements the property reporting topic and property reporting.

Enter the JSON message to be reported. The value of **luminance** is **30**. After a topic is published, the following information is displayed on the demo page.



If the reporting is successful, the reported device properties are displayed on the device details page.

Figure 2-15 Viewing reported data - MQTT

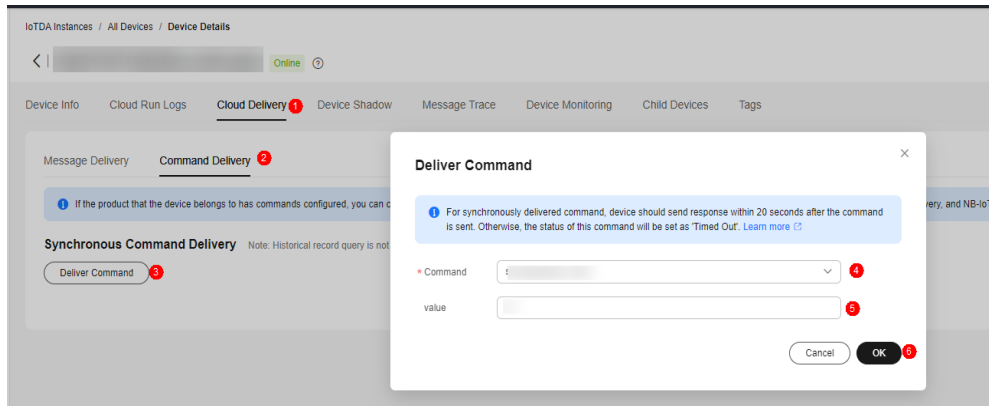


### Step 6 Receive a command.

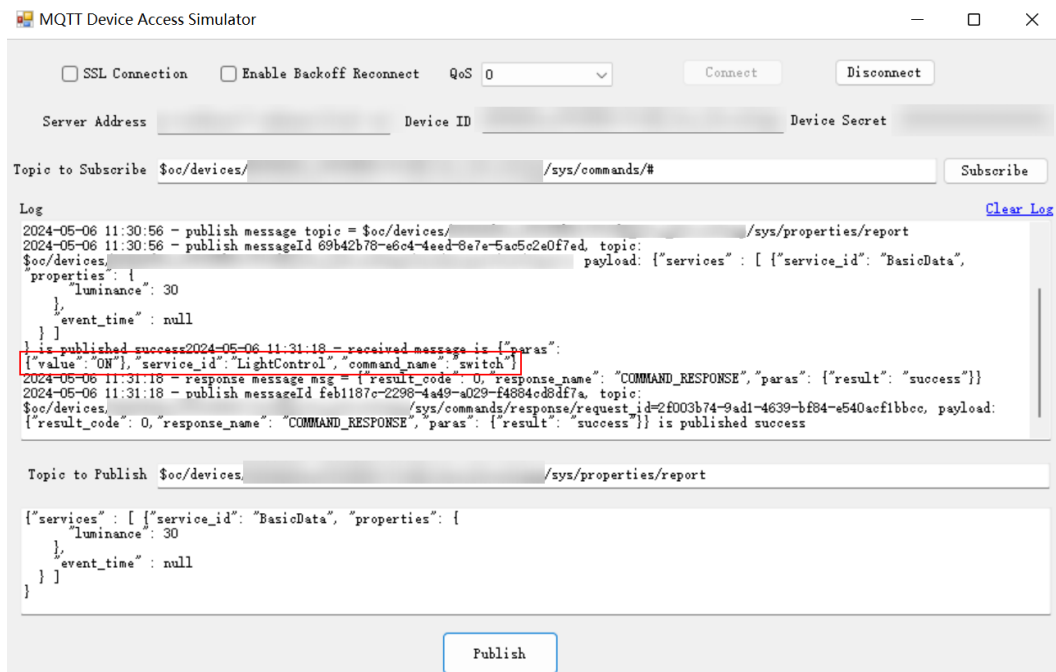
The simulator can receive commands delivered by the platform. After an MQTT connection is established and a topic is subscribed, you can deliver a command on the device details page of the IoTDA console. After the command is delivered, the MQTT callback receives the command delivered by the platform.

For example, if you want to remotely turn on the light, deliver the **LightControl: switch** command with the parameter value **ON**.

**Figure 2-16** Command delivery - Synchronous command delivery



After the synchronous command is delivered, the following information is displayed on the demo page.



----End

## Advanced Experience

After using the simulator to connect a simulated MQTT device to the platform, you may understand how the MQTT device interacts with the platform through open APIs.

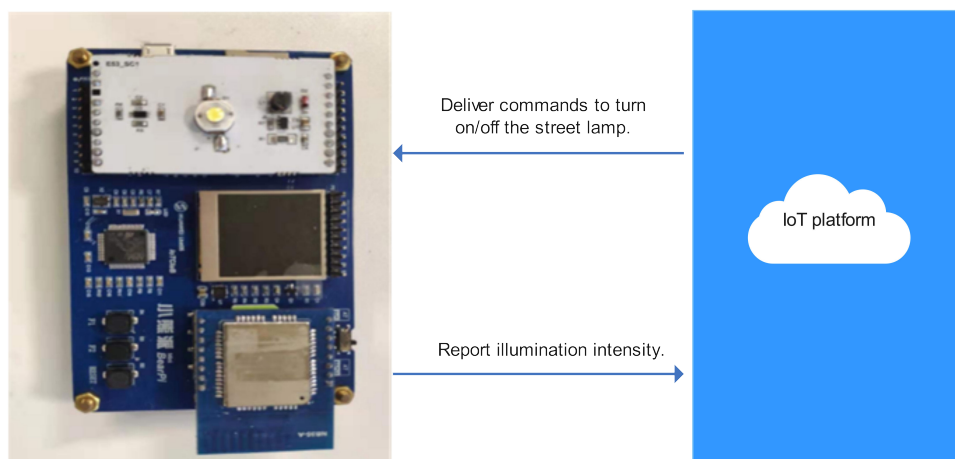
To better experience the IoTDA service, develop real-world applications and devices and connect them to the platform. For details, see [Developer Guide](#).

## 2.2 Developing a Smart Street Light Using NB-IoT BearPi

### Scenarios

Smart street lights play an important role in the intelligent transformation of city roads. They save energy in public lighting, reduce traffic accidents caused by poor lighting, and contribute to many other aspects in our community. As a common public facility, street lights can well exemplify how intelligence is transforming the world and implemented in our daily lives.

This section describes how to build a smart street light solution in just 10 minutes based on Huawei one-stop development tool platform (the IoT Link plug-in on Visual Studio Code), covering the device (BearPi development kit) and Huawei Cloud IoTDA. A smart street light detects and reports the illumination intensity to the IoTDA console. The LED light switch can be remotely controlled on the IoTDA console.

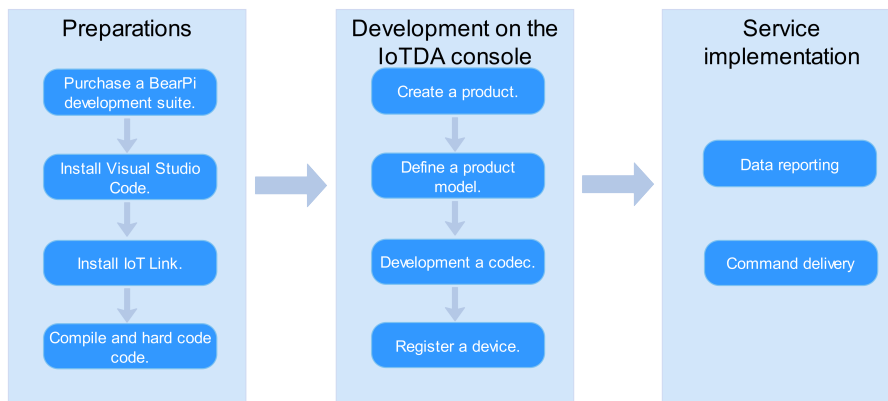


### Development Environment

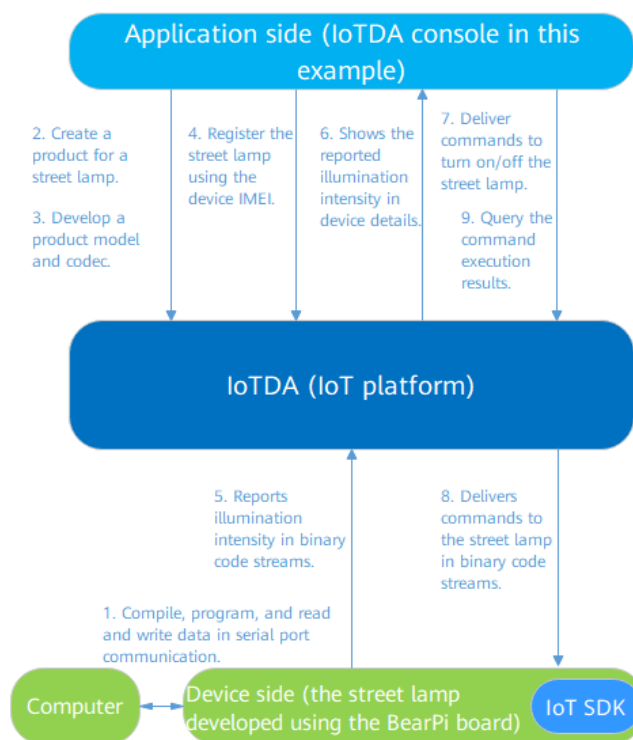
- Hardware: BearPi-IoT development suite (including NB-IoT cards, NB-IoT modules, smart street light function modules, and USB data cables)
- Software: Visual Studio Code, the IoT Link plug-in, Huawei Cloud **IoTDA** service, and 64-bit Windows 7 or later (64-bit Windows 10 is used in the following demonstration.)

### Development Process

The following figure shows the end-to-end process of developing a smart street light.



In this scenario, a device interacts with the platform using LwM2M (NB-IoT card). The application displays property changes of the device and delivers commands to the device.



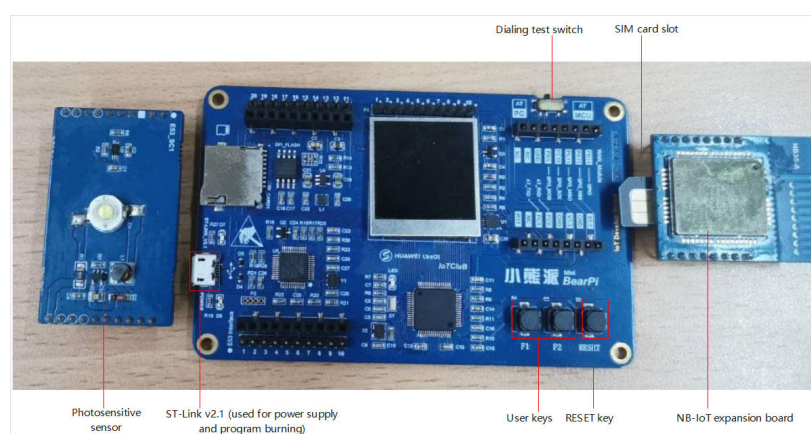
## Introduction to the BearPi Development Board

The development board is a sensing device in the IoT architecture. This type of device usually includes a sensor, communications module, chip, and operating system. To improve scalability of the development board, the BearPi development board does not use a conventional onboard design. Instead, it uses replaceable sensor and communications module expansion boards. The communications module is an entrance and exit of data transmission. Common communications modules include NB-IoT, Wi-Fi, and 4G ones. A chip controls a device. The development board has a built-in low-power STM32L431 chip as the main control chip (MCU). The operating system is Huawei LiteOS, which provides various device-cloud interworking components.

To facilitate development and debugging, the development board uses the onboard ST-Link of the 2.1 version, as shown in [Figure 1](#). It provides functions such as online debugging and programming, drag-and-drop download, and virtual serial port. An LCD screen with a resolution of 240 x 240 is installed at the center of the board to display sensor data and other debug logs. Below the LCD screen is the MCU.

There is a DIP switch in the upper right corner of the development board. When you set the switch to the AT-PC mode, use the serial port assistant on the computer to send AT commands to debug the communication module. When you set it to the AT-MCU mode, use the MCU to send AT commands to interact with the communication module and sends the collected sensor data to the cloud through the communication module.

**Figure 2-17** BearPi development board



## Hardware Connection

1. Insert the NB-IoT card into the SIM card slot of the NB-IoT expansion board. Ensure that the **notch-end faces outwards**, as shown in [Figure 2](#).
2. Insert the photosensitive sensor and NB-IoT expansion board into the development board. Ensure they are inserted in the correct direction. Use a USB data cable to connect the development board to the computer. If the screen displays information and the power indicator is on, the development board is powered on.

Figure 2-18 Hardware connection

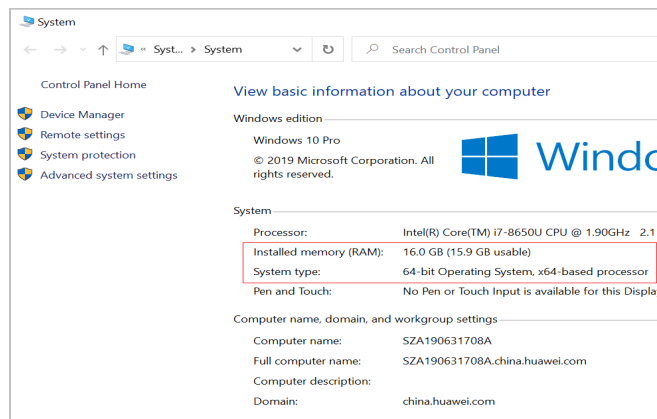


## Installing the IoT Link Studio Plug-in

IoT Link Studio is an integrated development environment (IDE) developed for IoT devices. It provides one-stop development capabilities, such as compilation, programming, and debugging, and supports multiple programming languages, such as C, C++, and assembly language.

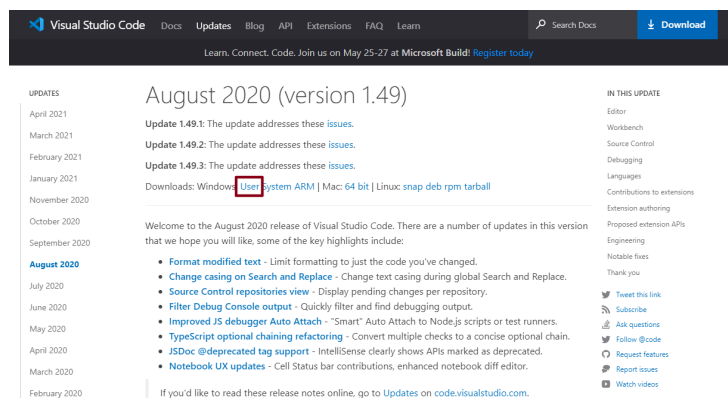
- Step 1** Obtain the operating system information. For example, on Windows 10, enter **pc** in the **Run** window, and click **Properties** to view the system information.

Figure 2-19 Obtaining the system configuration



- Step 2** Click [here](#) to download and install a Visual Studio Code version that suits your computer system. This section uses 64-bit Windows 10 as an example. Download version 1.49. Other versions do not support IoT Link.

Figure 2-20 Downloading Visual Studio Code




Note: Visual Studio Code does not support macOS.

**Step 3** After Visual Studio Code is installed, in its plug-in store, search for IoT Link and install it.

**Step 4** Perform the initial startup configuration.

When the IoT Link Studio is started for the first time, it automatically downloads the latest SDK package and GCC dependency environment. Ensure that the network is available. Do not close the window during the installation. After the installation is complete, restart the Visual Studio Code for the plug-in to take effect.

#### NOTE

If a proxy is required, click  in the lower left corner of the Visual Studio Code home page and choose **Settings > Application > Proxy**, and set **Use the proxy support for extensions** to **on**.

**Step 5** If the automatic downloading failed, manually download [SDK package](#), change the file name to **IoT\_LINK**, and save it to the **C:\Users\\${User name}\.iotlink\sdk** directory. Open the Visual Studio Code again. The following figure shows the directory format.

----End

## Configuring an IoT Link Studio Project

**Step 1** Click **Home** on the toolbar at the bottom of Visual Studio Code.

- **Home** is used to manage the IoT Link project.
- **Serial** is used to enter AT commands to check the status of the development board.
- **Build** is used to compile the sample code (displayed after Step 3).
- **Download** is used to hard code to the MCU (displayed after Step 3).

**Step 2** Configure the cross compilation toolchain. On the displayed page, click **IoT Link Settings** and select a toolchain. If the GCC tool directory or file does not exist, [download and install it](#).



**NOTE**

The version of the compilation toolchain downloaded by BearPi STM32431 is win32.zip.

**Step 3** On the displayed page, click **Create IoT Project**, enter the project name and project directory, and select the hardware platform and sample project template of the developer board.

- **Project Name:** Enter a project name, for example, QuickStart.
- **Project Path:** You can use the default installation path or select a path in a disk other than the system disk, for example, **D:\**.
- **Platform:** Currently, the demo applies only to the STM32L431\_BearPi hardware platform. Select **STM32L431\_BearPi**.
- **Create based on examples:** In this example, select **oc\_streetlight\_template**. Otherwise, the programmed demo does not match the product model defined on the console and data cannot be reported. If you need to adapt to other scenarios such as smart smoke sensors, select the **oc\_smoke\_template** demo.

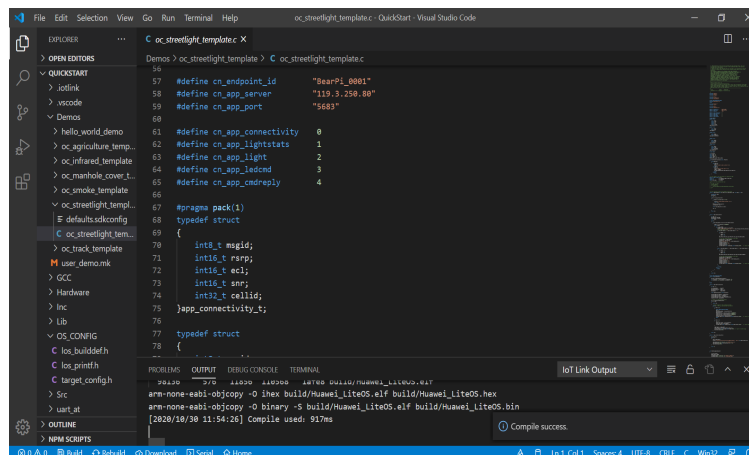
**Step 4** Click **OK**.

----End

## Compiling and Burning Code

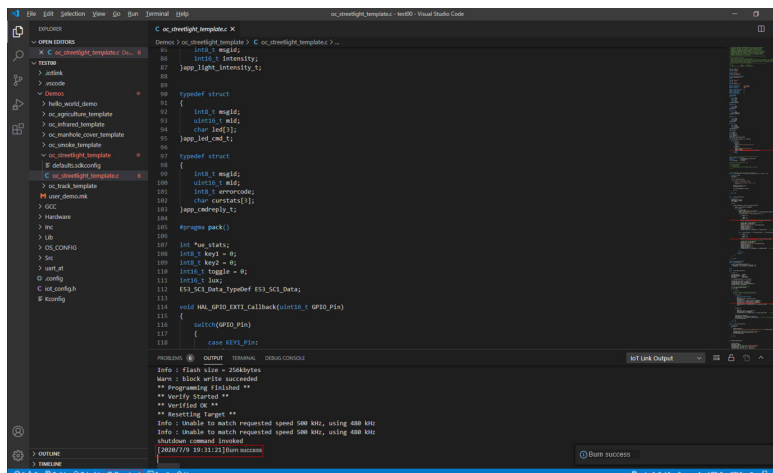
In the provided demo, the information for connecting to the Huawei Cloud IoTDA has been configured. You can directly compile code without modifying code and burn it to the development board MCU.

**Step 1** Click **Build** on the toolbar at the bottom of Visual Studio Code and wait until the compilation is complete. A message is then displayed, indicating that the compilation is successful.



**Step 2** Use a USB data cable to connect the BearPi development board to the computer. Set the dialing test switch in the upper right corner of the board to the **AT-MCU** mode on the right.

**Step 3** Click **Download** on the toolbar at the bottom of Visual Studio Code and wait until the burning is complete. A message is then displayed, indicating that the burning is successful.



**NOTE**

If the burning fails, the possible cause is that the development board does not have a driver and cannot communicate with the computer through the serial port. In this case, perform **2** to check whether the ST-Link driver is installed. If the driver is not installed, download and install the ST-Link driver by following **Step 4.1**.

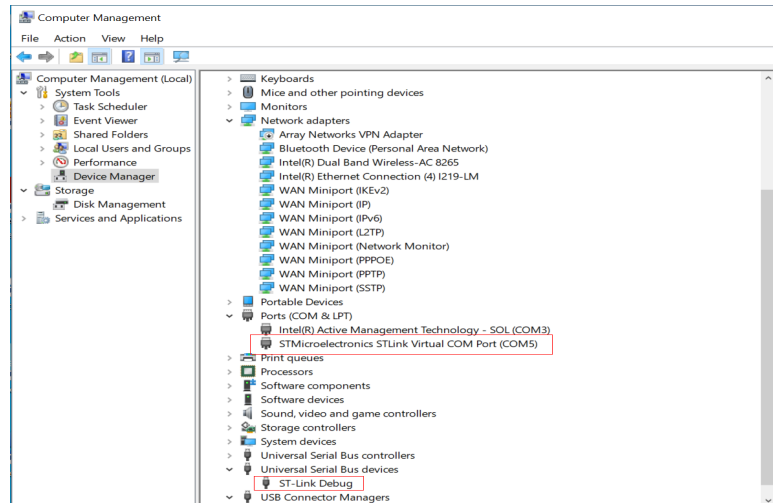
**Step 4** (Optional) Install the ST-Link driver.

1. Visit the **ST website**, download the ST-Link driver, and double-click the **stlink\_winusb\_install.bat** file to start automatic installation. This section uses Windows 10 64-bit ST-Link 2.0.1 as an example.

Name	Size	Packed	Type	Modified	CRC32
.			File folder		
amd64			File folder	2/8/2018 1:24 ...	
x86			File folder	2/8/2018 1:24 ...	
dpinst_amd64.exe	680,440	242,269	Application	2/8/2018 1:24 ...	E0BABB1A
dpinst_x86.exe	552,328	231,230	Application	2/8/2018 1:24 ...	99D60074
readme.txt	391	250	Text Document	9/11/2018 4:15...	ABBCDABC
stlink_bridge_winusb.inf	2,853	1,093	Setup Information	2/8/2018 1:25 ...	B797E7D3
stlink_dbg_winusb.inf	4,373	1,347	Setup Information	2/8/2018 1:26 ...	83C18B24
stlink_VCP.inf	2,467	871	Setup Information	2/8/2018 1:29 ...	16A7D847
stlink_winusb_install.bat	412	259	Windows Batch File	6/23/2017 10:1...	F9AE8CDC
stlinkbridgewinusb_x64.cat	11,004	5,890	Security Catalog	2/8/2018 1:33 ...	14FD19D4
stlinkbridgewinusb_x86.cat	11,004	5,892	Security Catalog	2/8/2018 1:33 ...	410257F4
stlinkdbgwinusb_x64.cat	10,997	5,891	Security Catalog	2/8/2018 1:33 ...	2B3C3C55
stlinkdbgwinusb_x86.cat	10,998	5,892	Security Catalog	2/8/2018 1:33 ...	9D9979E2
stlinkvcp_x64.cat	9,248	5,474	Security Catalog	2/8/2018 1:33 ...	E33A16BA
stlinkvcp_x86.cat	9,247	5,470	Security Catalog	2/8/2018 1:33 ...	829A23AB

Note: You can also use an EXE file that adapts to your system version to install the ST-Link driver.

2. Open the device manager on the computer to check whether the driver is installed. If the information shown in the following figure is displayed, the driver is installed.



----End

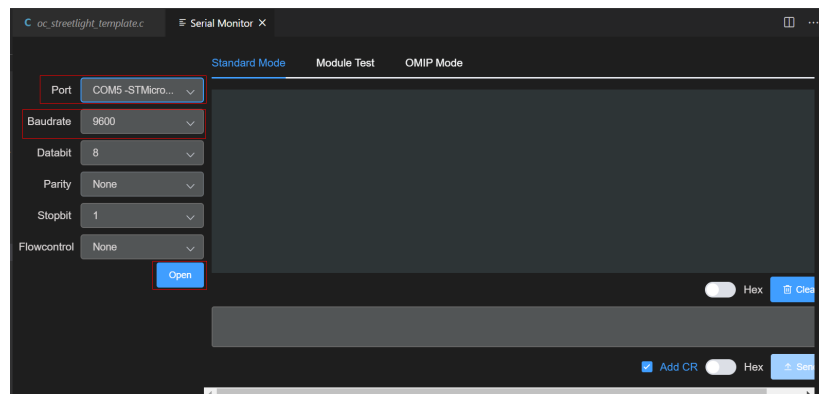
## Locating Module Communication Problems Using AT Commands

When IoT Link is connected to the platform, you can use AT commands to quickly locate the connectivity problem between the module and the cloud. This section describes how to use AT commands to detect common problems of the communications module, for example, the device fails to go online or data fails to be reported.

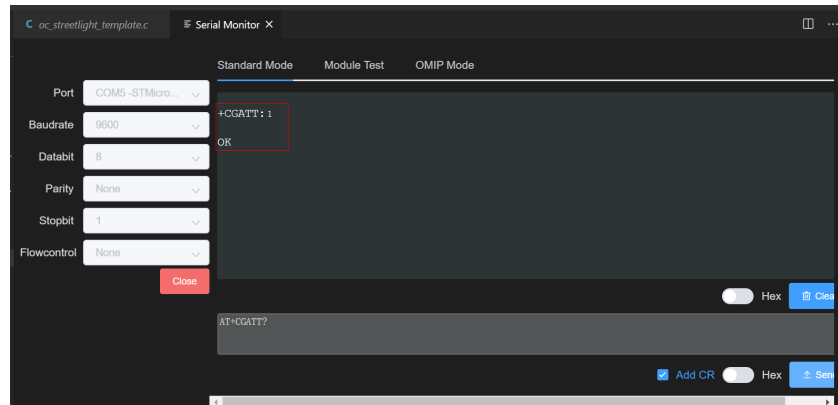
1. Connect the BearPi development board to the computer and ensure that the driver has been installed. Set the dialing test switch in the upper right corner of the board to the **AT-PC** mode.
2. Click **Serial** on the toolbar at the bottom of Visual Studio Code.



3. Select the port number obtained in 2, set **Baudrate** to **9600**, and click **Open**.



4. Enter **AT+CGATT?** and click **Send**. If **+CGATT:1** is returned, the network attach is successful, indicating that the NB-IoT network is normal. If **+CGATT:0** is returned, the network attach fails, indicating that the NB-IoT network is abnormal. In this case, check whether the SIM card is correctly inserted or contact the carrier to check the network status.



### NOTE

After using the AT commands to detect the module communication, set the dialing test switch to the **AT-MCU** mode so that the collected data can be sent to the platform through the communication module after the console configuration.

In the **AT-PC** mode, the development board communicates with the serial port of the computer, and AT commands are used to read and write data such as the status of the development board. In the **AT-MCU** mode, the development board connects to the network through the SIM card inserted into the module to implement NB-IoT communications.

5. The **AT+CSQ<CR>** command is used to check the network signal strength and SIM card status. Enter **AT+CSQ<CR>** and click **Send**. **+CSQ:\*\*,##** is returned. In the preceding output, **\*\*** ranges from 10 to 31. A larger value indicates better signal quality. **##** indicates the bit error rate, which ranges from 0 to 99. If the returned values are not within these ranges, check whether the antenna or SIM card is correctly installed.

Note: This section lists only two common AT commands for detecting the network status of the module. For more AT commands, see the instructions of the BearPi module.

## Operations on the Console

After connecting the physical device and compiling and programming code, go to the IoTDA console to create a product, define a product model, develop a codec, and register the device.

- Creating a product: Specify the protocol type, data format, manufacturer name, and device type of a product on the platform. In this example, create a smart street light product on the console based on the product features.
- Defining 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 properties. In this example, define a street light product model with light switch control, illumination intensity, and signal quality properties on the console.
- Developing a codec: A codec is called by the platform to convert data between the binary and JSON formats. The binary data reported by a device is decoded into the JSON format for the application to read, and the commands delivered by the application are encoded into the binary format for the device.

to understand and execute. Since the data format of smart street lights is binary, a codec is needed to enable the platform to understand the data reported by the smart street light and to enable the smart street light to understand the commands delivered by the platform.

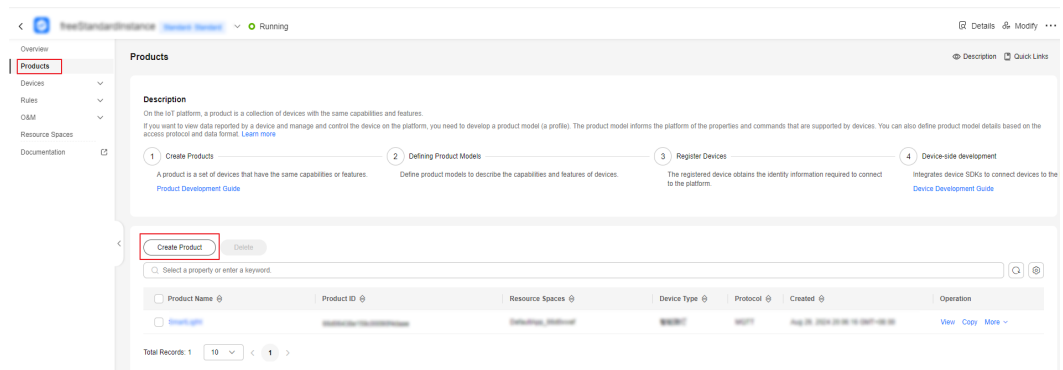
- Registering the device: Register the BearPi smart street light with the platform.

## Creating a Product

A product is a collection of devices with the same capabilities or features. In addition to physical devices, a product includes product information, product models, and codecs generated during IoT capability building. In this example, create a smart street light product on the IoTDA console.

- Step 1** Log in to the [console](#), choose **Products** in the navigation pane, and click **Create Product** on the left.

**Figure 2-21** Creating a product



- Step 2** Create a product whose protocol type is LwM2M or CoAP and device type is **StreetLamp**, set parameters as prompted, and click **OK**.

Figure 2-22 Creating a product - CoAP

**Create Product** ✕

\* Resource Space ?

To create a new resource space, you can [go to the instance details page](#).

\* Product Name

Protocol ?

\* Data Type ?

Device Type Selection  Standard profile  Custom

\* Device Type ?

Advanced Settings ^ Custom Product ID | Description

Product ID ?

Description

----End

## Uploading a Product Model

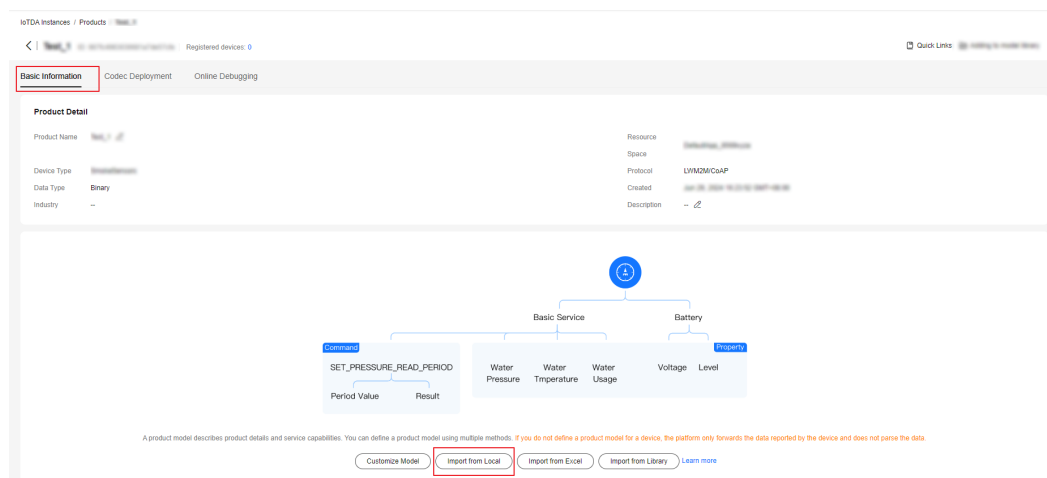
A product model is a JSON file that describes device capabilities. It defines basic device properties and message formats for data reporting and command delivery. Defining a product model is to construct an abstract model of a device in the platform to enable the platform to understand the device function. A developed product model is provided for you to quickly experience the cloud migration process. If you want to go through the process of developing a product model, go to [Developing a Product Model](#).

### Procedure

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

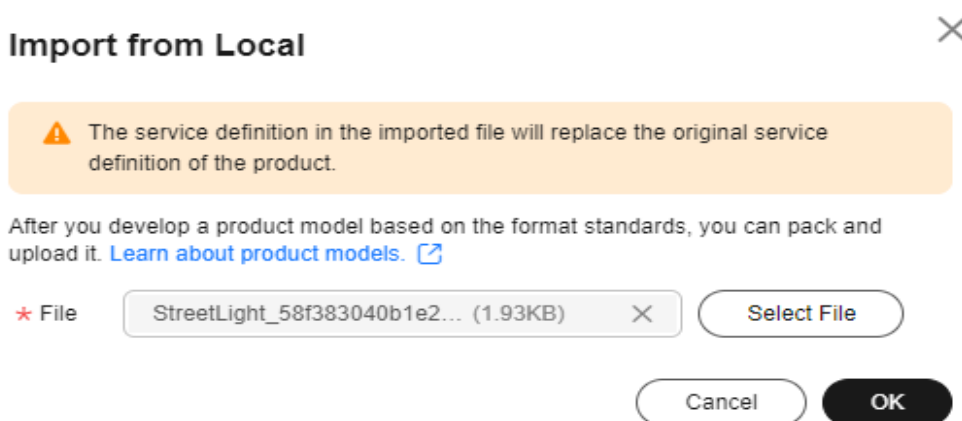
**Step 2** On the **Model Definition** tab page, click **Import from Local**.

**Figure 2-23** Uploading a product model - CoAP



**Step 3** On the dialog box displayed, upload the **product model provided** and click **OK**.

**Figure 2-24** Uploading a model file - CoAP



----End

## Registering a Device

This section describes how to register a device integrated with the NB-IoT module, the BearPi smart street light in this example, to the platform.

- Step 1** On the product details page, click the **Online Debugging** tab, and click **Add Test Device**. This section use a non-security NB-IoT device as an example.
- Step 2** In the dialog box displayed, set the parameters and click **OK**.

Figure 2-25 Adding a test device

**Add Test Device** ×

Device Type:  Physical device  Virtual device

Device Name:

\* Node ID:

Authentication Type:  Secret

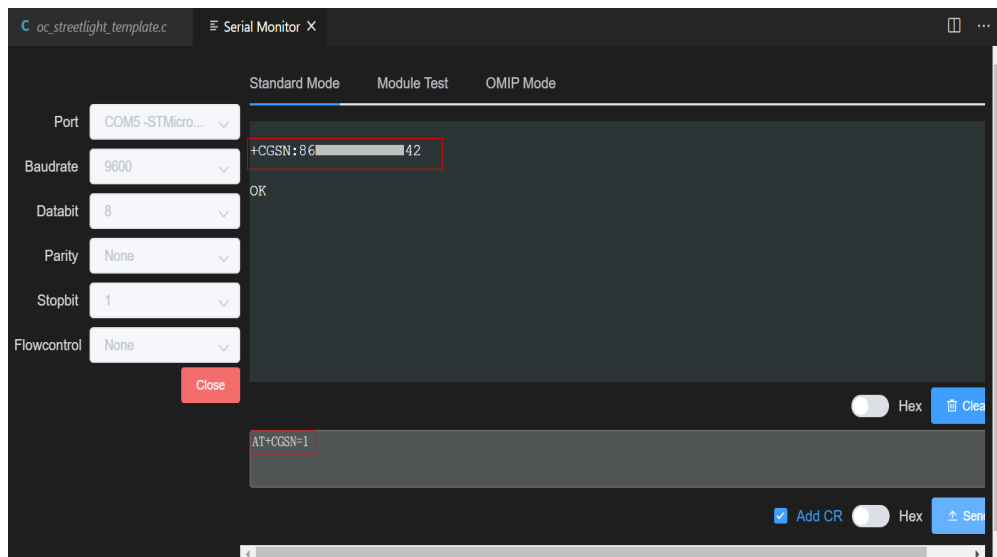
Secret:  👁

Confirm Secret:  👁

- **Device Name:** Customize a name.
- **Node ID:** Enter the IMEI of the device. The node ID will be carried by the device for device access authentication. You can view the node ID on the NB-IoT module. You can also set the dialing test switch to the **AT-PC** mode, select the STM port, set the baud rate to 9600, and run the **AT+CGSN=1** command to obtain the IMEI.

**Note:** After obtaining the IMEI and registering the device, set the dialing test switch of the development board to the **AT-MCU** mode because the development board connects to the network through the NB-IoT card only in MCU mode.





- **Registration Mode:** Select **Unencrypted**.

**Step 3** The device is created. You can view the created device on the console.

----End

## Data Reporting

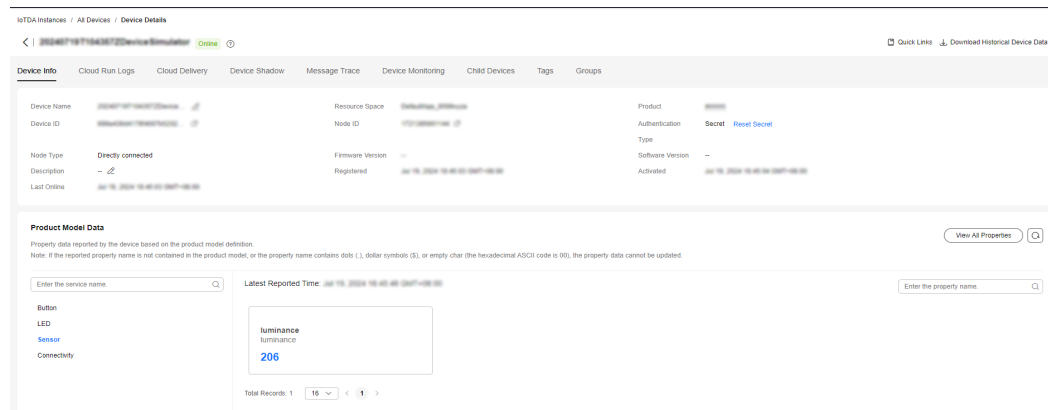
After the connection between the platform and the development board is set up, the BearPi smart street light reports the light sensor data every 2 seconds according to the code burnt to the development board. The reporting frequency can be customized in the demo based on service requirements. You can block the light with your hand to change the light intensity and view the real-time change of the light intensity data reported to the platform.

**Note:** Ensure that the dialing test switch of the development board is set to the AT-MCU mode.

**Step 1** Log in to the **IoTDA** console and click the target instance card. In the navigation pane, choose **Devices > All Devices**.

**Step 2** Select the target device and click **View** to view the data reported to the platform.

**Figure 2-26** NB-IoT device data reporting - Viewing data

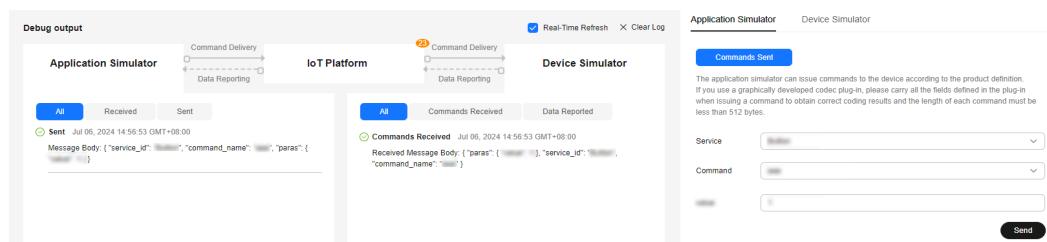


----End

## Delivering a Command

- Step 1** Log in to the [IoTDA console](#). Click the target product to go to its details page.
- Step 2** On the **Online Debugging** tab page, click the target device to access the debugging page.
- Step 3** After setting the command parameters, click **Send**.

**Figure 2-27** Command delivery debugging - SmokeDetector



- Step 4** The light on the BearPi board is on. Deliver the **OFF** command. The light is turned off.



----End

This is the end-to-end development of a smart street light by using the NB-IoT BearPi development board.

## Reference

- **Developing a Product Model**  
On the **Basic Information** tab page, click **Customize Model** to add services of the product.  
[Table 2-3](#) describes the service defined in the product model.

**Table 2-3** Device service list

Service ID	Description
Button	Real-time button detection
LED	LED control
Sensor	Real-time light intensity detection
Connectivity	Real-time signal quality detection

The following table lists the service capabilities.

**Table 2-4** Button

Capability Description	Property Name	Data Type	Data Range
Property	toggle	Integer	0 to 65535

**Table 2-5** LED command list

Capability Description	Command Name	Command Parameter	Parameter Name	Type	Data Length	Enumeration
Commands	Set_LED	Command	<b>LED</b>	String	3	ON,OFF
		Response	<b>Light_state</b>	String	3	ON,OFF

**Table 2-6** Sensor

Capability Description	Property Name	Data Type	Data Range
Property	luminance	Integer	0 to 65535

**Table 2-7** Connectivity

Capability Description	Property Name	Data Type	Data Range
Properties	SignalPower	Integer	-140 to -44

Capability Description	Property Name	Data Type	Data Range
	ECL	Integer	0 to 2
	SNR	Integer	-20 to 30
	CellID	Integer	0 to 65535

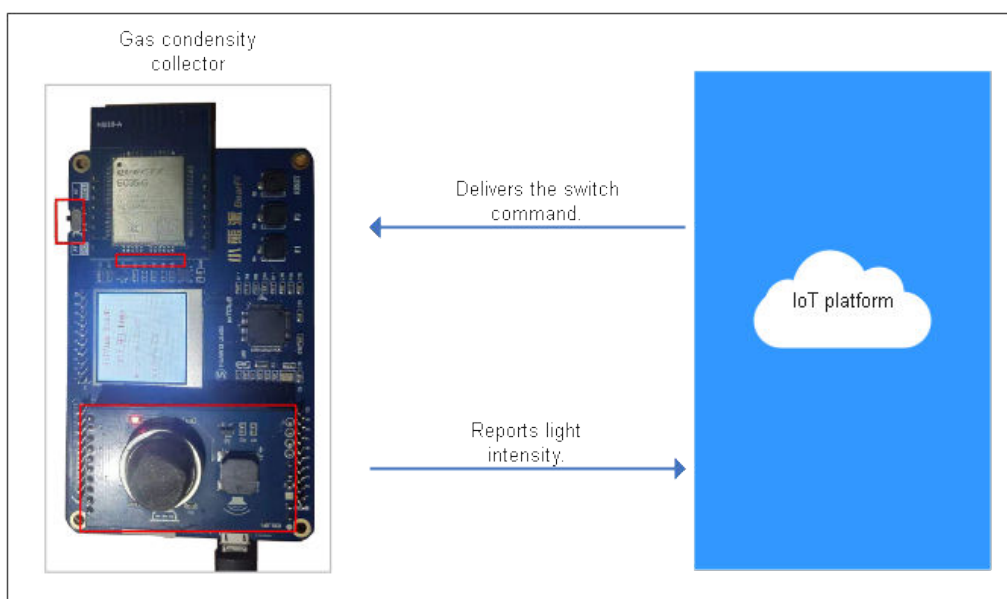
## 2.3 Developing a Smart Smoke Detector Using NB-IoT BearPi

### Scenarios

Fires have caused great loss of lives and properties each year. As more independent smoke detectors have been put to use, their limitations become obvious. For example, users cannot monitor the working status of smoke detectors in real time or receive alarm information when they are absent.

On the contrary, NB-IoT smart smoke detectors overcome their disadvantages like difficult cabling, short battery lifespan, high maintenance costs, and inability to interact with property owners and firefighting departments. These smart smoke detectors use wireless communication and feature plug-and-play, no cabling, and easy installation.

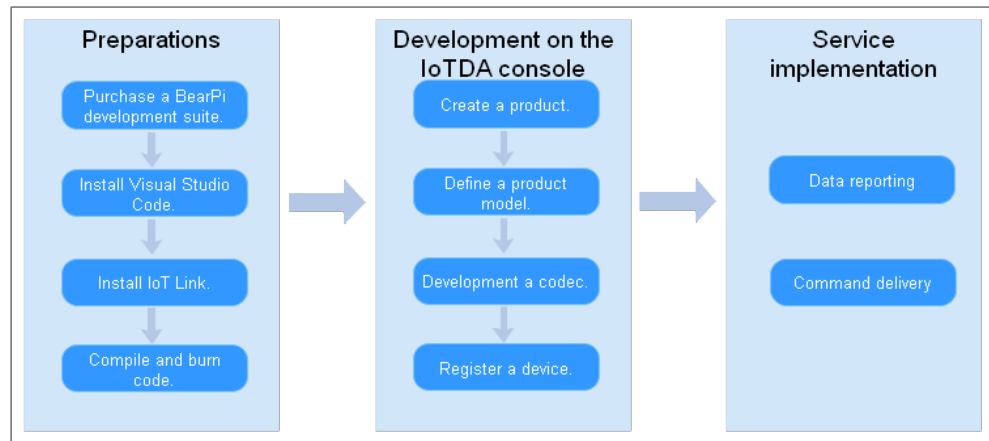
This topic describes how to build a smart smoke detector solution in just 10 minutes based on Huawei one-stop development tool platform (the IoT Link plug-in on Visual Studio Code), covering the device (BearPi development kit) and Huawei Cloud IoTDA. A smart smoke detector detects and reports smoke density to the IoTDA console. The beep switch can be remotely controlled on the IoTDA console.



## Development Environment

- Hardware: BearPi-IoT development suite (including NB-IoT cards, NB-IoT modules, smart street light function modules, and USB data cables)
- Software: Visual Studio Code, the IoT Link plug-in, Huawei Cloud **IoTDA** service, and 64-bit Windows 7 or later (64-bit Windows 10 is used in the following demonstration.)

## Development Process



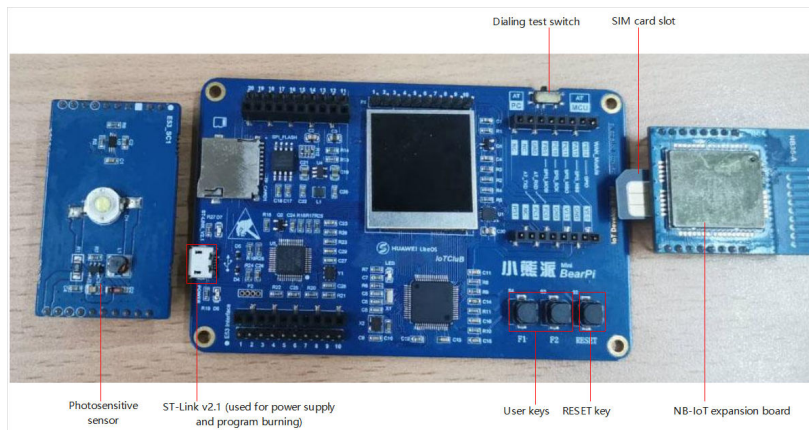
## Introduction to the BearPi Development Board

The development board is a sensing device in the IoT architecture. This type of device usually includes a sensor, communications module, chip, and operating system. To improve scalability of the development board, the BearPi development board does not use a conventional onboard design. Instead, it uses replaceable sensor and communications module expansion boards. The communications module is an entrance and exit of data transmission. Common communications modules include NB-IoT, Wi-Fi, and 4G ones. A chip controls a device. The development board has a built-in low-power STM32L431 chip as the main control chip (MCU). The operating system is Huawei LiteOS, which provides various device-cloud interworking components.

To facilitate development and debugging, the development board uses the onboard ST-Link of the 2.1 version, as shown in **Figure 1**. It provides functions such as online debugging and programming, drag-and-drop download, and virtual serial port. An LCD screen with a resolution of 240 x 240 is installed at the center of the board to display sensor data and other debug logs. Below the LCD screen is the MCU.

There is a DIP switch in the upper right corner of the development board. When you set the switch to the AT-PC mode, use the serial port assistant on the computer to send AT commands to debug the communication module. When you set it to the AT-MCU mode, use the MCU to send AT commands to interact with the communication module and sends the collected sensor data to the cloud through the communication module.

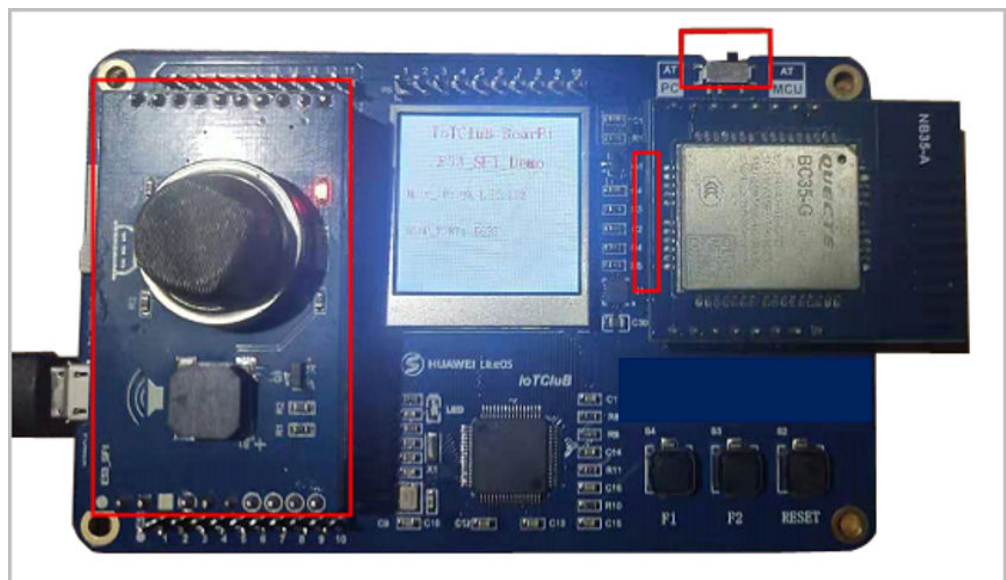
Figure 2-28 BearPi development board



## Hardware Connection

1. Insert the NB-IoT card into the SIM card slot of the NB-IoT expansion board. Ensure that the notch-end faces outwards, as shown in [Figure 2-29](#).
2. Insert the smoke density collection control board and NB-IoT expansion board into the development board. Ensure they are inserted in the correct direction. Use a USB data cable to connect the development board to the computer. If the screen displays information and the power indicator is on, the development board is powered on.

Figure 2-29 Hardware connection

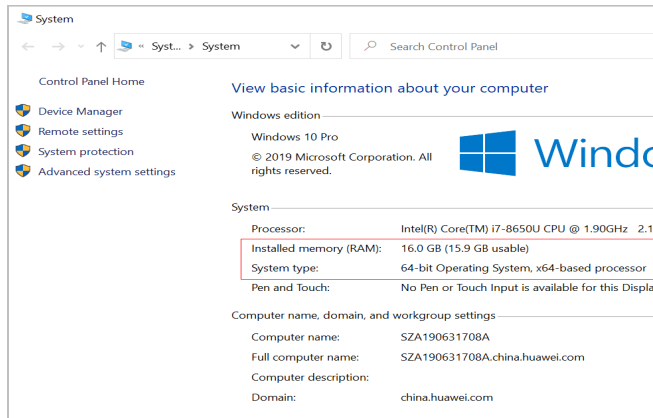


## Installing the IoT Link Studio Plug-in

IoT Link Studio is an integrated development environment (IDE) developed for IoT devices. It provides one-stop development capabilities, such as compilation, programming, and debugging, and supports multiple programming languages, such as C, C++, and assembly language.

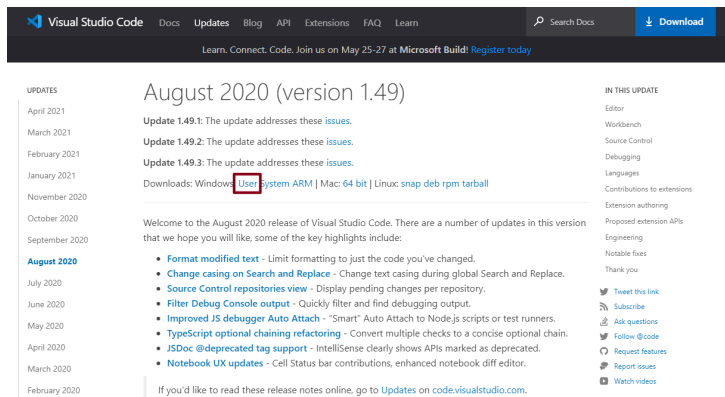
**Step 1** Obtain the operating system information. For example, on Windows 10, enter `pc` in the **Run** window, and click **Properties** to view the system information.

**Figure 2-30** Obtaining the system configuration



**Step 2** Click [here](#) to download and install a Visual Studio Code version that suits your computer system. This section uses 64-bit Windows 10 as an example. Download version 1.49. Other versions do not support IoT Link.

**Figure 2-31** Downloading Visual Studio Code




Note: Visual Studio Code does not support macOS.

**Step 3** After Visual Studio Code is installed, in its plug-in store, search for IoT Link and install it.

**Step 4** Perform the initial startup configuration.

When the IoT Link Studio is started for the first time, it automatically downloads the latest SDK package and GCC dependency environment. Ensure that the network is available. Do not close the window during the installation. After the installation is complete, restart the Visual Studio Code for the plug-in to take effect.

**NOTE**

If a proxy is required, click  in the lower left corner of the Visual Studio Code home page and choose **Settings > Application > Proxy**, and set **Use the proxy support for extensions** to **on**.

**Step 5** If the automatic downloading failed, manually download [SDK package](#), change the file name to **IoT\_LINK**, and save it to the **C:\Users\\${User name}\.iotlink\sdk** directory. Open the Visual Studio Code again. The following figure shows the directory format.

----End

## Configuring an IoT Link Studio Project

**Step 1** Click **Home** on the toolbar at the bottom of Visual Studio Code.

- **Home** is used to manage the IoT Link project.
- **Serial** is used to enter AT commands to check the status of the development board.
- **Build** is used to compile the sample code (displayed after Step 3).
- **Download** is used to hard code to the MCU (displayed after Step 3).

**Step 2** Configure the cross compilation toolchain. On the displayed page, click **IoT Link Settings** and select a toolchain. If the GCC tool directory or file does not exist, [download and install it](#).

### NOTE

The version of the compilation toolchain downloaded by BearPi STM32431 is win32.zip.

**Step 3** On the displayed page, click **Create IoT Project**, enter the project name and project directory, and select the hardware platform and sample project template of the developer board.

- **Project Name:** Set this parameter as required, for example, **Smoke**.
- **Project Path:** You can use the default installation path or select a path in a disk other than the system disk, for example, **D:\**.
- **Platform:** Currently, the demo applies only to the STM32L431\_BearPi hardware platform. Select **STM32L431\_BearPi**.
- **Create based on examples:** In this example, select **oc\_smoke\_template demo**. Otherwise, the demo does not match the product model defined on the console and data cannot be reported. Click **OK**.

**Step 4** Click **OK**.

----End

## Compiling and Burning Code

In the provided demo, the information for connecting to the Huawei Cloud IoTDA has been configured. You can directly compile code without modifying code and burn it to the development board MCU.

**Step 1** Click **Build** on the toolbar at the bottom of Visual Studio Code and wait until the compilation is complete. A message is then displayed, indicating that the compilation is successful.



- Step 2** Use a USB data cable to connect the BearPi development board to the computer. Set the dialing test switch in the upper right corner of the board to the **AT-MCU** mode on the right.
- Step 3** Click **Download** on the toolbar at the bottom of Visual Studio Code and wait until the burning is complete. A message is then displayed, indicating that the burning is successful.

#### NOTE

If the burning fails, the possible cause is that the development board does not have a driver and cannot communicate with the computer through the serial port. In this case, perform **2** to check whether the ST-Link driver is installed. If the driver is not installed, download and install the ST-Link driver by following **Step 4**.

- Step 4** (Optional) Install the ST-Link driver.

1. Visit the [ST website](#), download the ST-Link driver, and double-click the **stlink\_winusb\_install.bat** file to start automatic installation. This section uses Windows 10 64-bit ST-Link 2.0.1 as an example.

Note: You can also use an EXE file that adapts to your system version to install the ST-Link driver.

2. Open the device manager on the computer to check whether the driver is installed. If the information shown in the following figure is displayed, the driver is installed.

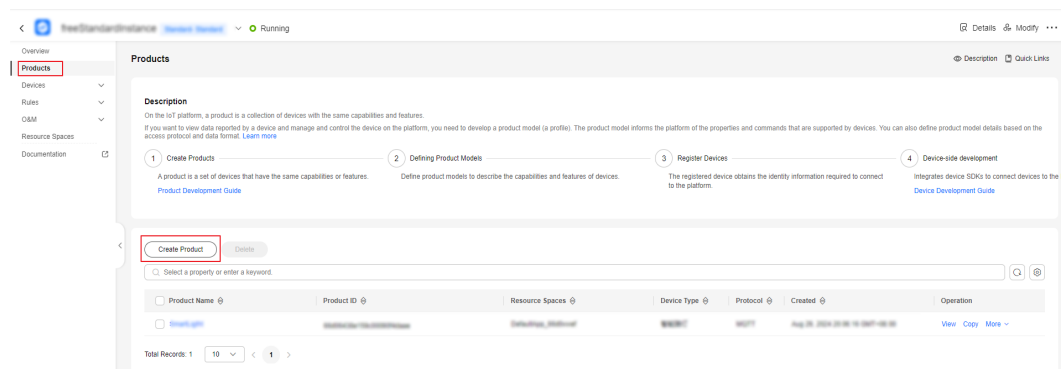
----End

## Creating a Product

A product is a collection of devices with the same capabilities or features. In addition to physical devices, a product includes product information, product models, and codecs generated during IoT capability building.

- Step 1** Log in to the [console](#), choose **Products** in the navigation pane, and click **Create Product** on the left.

**Figure 2-32** Creating a product



- Step 2** Create a product whose protocol type is LwM2M/CoAP and device type is **SmokeSensors**, set parameters as prompted, and click **OK**.

**Figure 2-33** Creating a product - CoAP

**Create Product** ✕

\* Resource Space ?

To create a new resource space, you can [go to the instance details page](#).

\* Product Name

Protocol ?

\* Data Type ?

Device Type Selection  Standard profile  Custom

\* Device Type ?

Advanced Settings  Custom Product ID | Description

Product ID ?

Description   
0/128 ↗

----End

## Defining a Product Model

On the **Basic Information** tab page, click **Customize Model** to add a service for the product.

**Table 2-8** describes the service defined in the product model.

**Table 2-8** Device service list

Service ID	Description
Enter <b>Smoke</b> .	Detects smoke density in real time.

**Table 2** and **Table 3** describe the service capabilities.

**Table 2-9** Smoke

Capability Description	Property Name	Data Type	Data Range
Properties	Smoke value	int	0 to 65535

**Table 2-10** Smoke commands

Capability Description	Command Name	Command Parameter	Parameter Name	Data Type	Data Length	Enumeration
Commands	Smoke control beep	Command	<b>beep</b>	string	3	ON,OFF
		Response	<b>beep_state</b>	int	/	/

### Adding the Smoke Service

- On the **Add Service** page, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.
  - Service ID:** Enter **Smoke**.
  - Service Type:** You are advised to set this parameter to the same value as **Service ID**.
  - Description:** Enter **Detects smoke density in real time**.

**Figure 2-34** Adding a service - Smoke

**Add Service**
×

\* Service ID

Service Type  ⓘ

Description  34/128 ↕

Cancel
OK

2. Choose **Smoke**, click **Add Property**, enter related information, and click **OK**.
  - **Property Name**: Enter **Smoke\_Value**.
  - **Data Type**: Select **Integer**.
  - **Access Permissions**: Select **Read** and **Write**.
  - **Value Range**: Set it to 0 to 65535.
  - **Step**: Enter **0**.
  - **Unit**: Leave it blank.

Figure 2-35 Adding a property - Smoke\_value

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

- Property Name**: A text input field containing "Smoke\_Value".
- Description**: A text area that is currently empty, with a character count "0/128" and a double-slash icon in the bottom right corner.
- Data Type**: A dropdown menu set to "Integer".
- Access Permissions**: Two buttons, "Read" and "Write", both of which are selected (indicated by blue checkmarks).
- Value Range**: Two text input fields, the first containing "0" and the second containing "65535", separated by a minus sign.
- Step**: A text input field that is empty.
- Unit**: A text input field that is empty.
- Buttons**: "Cancel" and "OK" buttons at the bottom right.

3. Choose **Smoke**, click **Add Command**, and enter the command name **Smoke\_Control\_Beep**.

**Figure 2-36** Adding a command - Smoke\_Control\_Beep

**Add Command** [Close]

\* Command Name:

Command:

Parameters

Parameter Name	Data Type	Description	Operation
<b>No table data available.</b> No Command Parameters data available. Add Command Parameter first.			

Total Records: 0 | 5 | < 1 >

Response:

Parameters

Parameter Name	Data Type	Description	Operation
----------------	-----------	-------------	-----------

4. Click **Add Command Parameter** and **Add Response Parameter** respectively, enter related information, and click **OK**.

Figure 2-37 Adding a command parameter - Beep

✕

### Add Parameter

★ Parameter Name

Description   
0/128 ↗

★ Data Type  ▾

★ Length

Enumerated Values   
9/1,024 ↗

**Figure 2-38** Adding a command parameter - Beep\_State

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 and values:

- Parameter Name:** Beep\_State
- Description:** (Empty text box with a character count of 0/128)
- Data Type:** Integer (dropdown menu)
- Value Range:** 0 - 1
- Step:** (Empty text box)
- Unit:** (Empty text box)

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

## Registering a Device

This section describes how to register a non-secure NB-IoT module.

**Step 1** On the product details page, click the **Online Debugging** tab, and click **Add Test Device**. This section use a non-security NB-IoT device as an example.

**Step 2** In the dialog box displayed, set the parameters and click **OK**.

Figure 2-39 Adding a test device

**Add Test Device** ×

Device Type:  Physical device  Virtual device

Device Name:

\* Node ID:

Authentication Type:  Secret

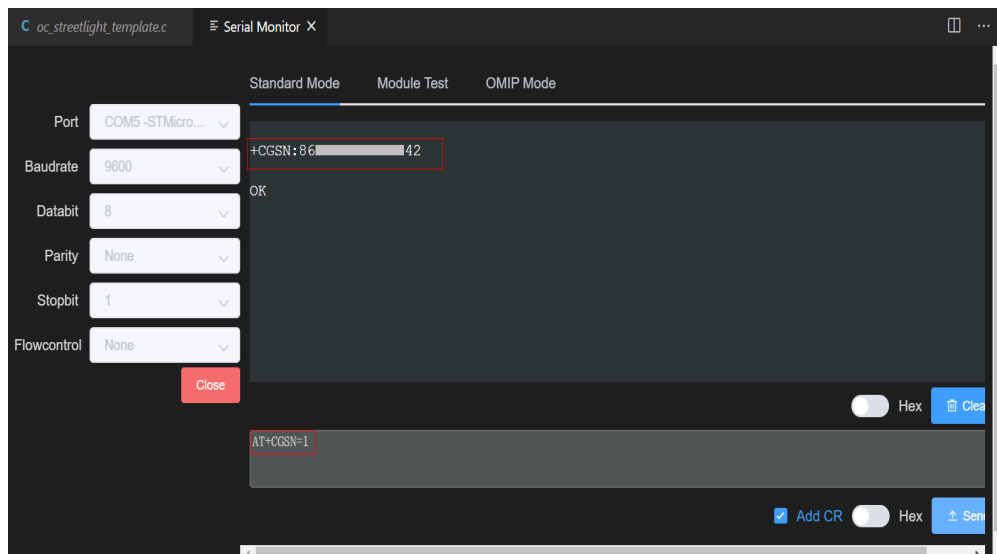
Secret:  👁

Confirm Secret:  👁

- **Device Name:** Customize a name.
- **Node ID:** Enter the IMEI of the device. The node ID will be carried by the device for device access authentication. You can view the node ID on the NB-IoT module. You can also set the dialing test switch to the **AT-PC** mode, select the STM port, set the baud rate to 9600, and run the **AT+CGSN=1** command to obtain the IMEI.

**Note:** After obtaining the IMEI and registering the device, set the dialing test switch of the development board to the **AT-MCU** mode because the development board connects to the network through the NB-IoT card only in MCU mode.





- **Registration Mode: Select Unencrypted.**

**Step 3** The device is created. You can view the created device on the console.

----End

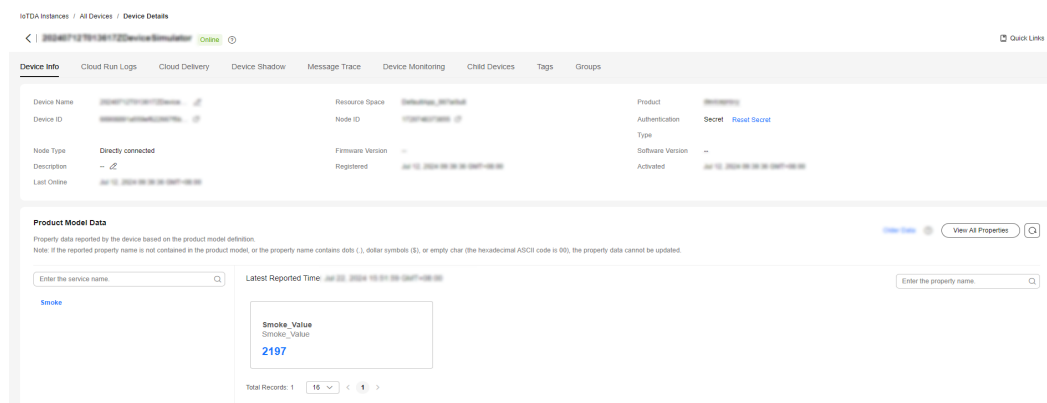
## Reporting Data

After the development board is connected to the platform, it reports the smoke density.

**Step 1** Log in to the **IoTDA** console and click the target instance card. Choose **Devices > All Devices**.

**Step 2** Select the target device and click **View** to check the data reported to the platform.

**Figure 2-40** NB-IoT device data reporting - Viewing data (Smoke)



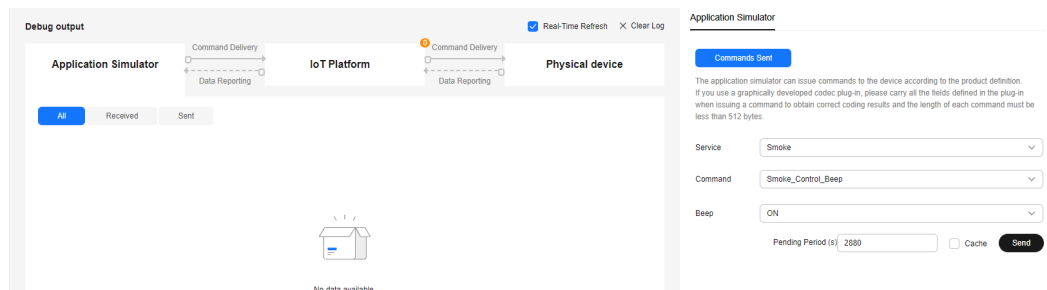
----End

## Delivering a Command

**Step 1** Log in to the **IoTDA console**. Click the target product to go to the product details page.

- Step 2** On the **Online Debugging** tab page, click the target device to access the debugging page.
- Step 3** After setting the command parameters, click **Send**.

**Figure 2-41** Command delivery debugging - Smoke

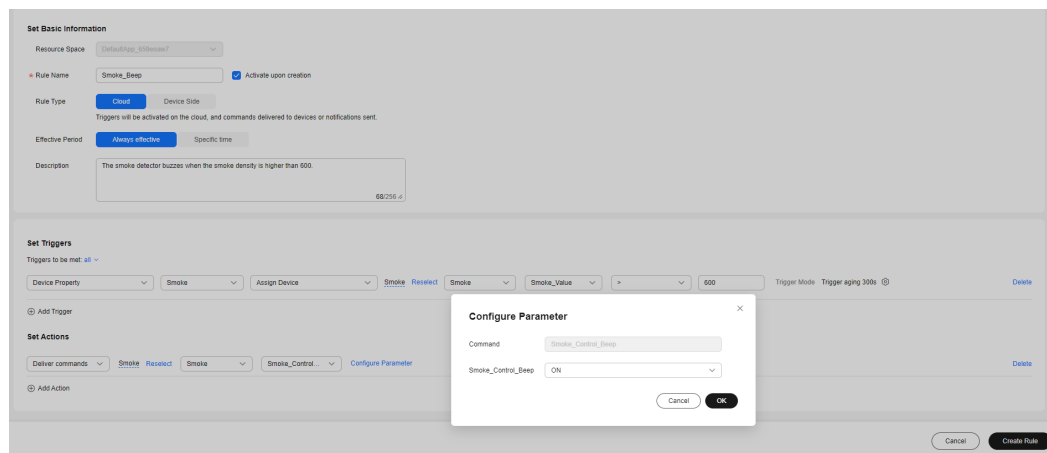


----End

### (Optional) Configuring a Device Linkage Rule

- Step 1** In the navigation pane, choose **Rules > Device Linkage**, and click **Create Rule** in the upper right corner.
- Step 2** Create a device linkage rule based on the table below.

**Figure 2-42** Creating a linkage rule - Smoke\_Beep



Parameter	Description
Rule Name	Specify the name of the rule to create, for example, <b>Smoke_Beep</b> .
Activate upon creation	Select <b>Activate upon creation</b> .
Effective Period	Select <b>Always effective</b> .

Parameter	Description
Description	Enter a description of the rule, for example, "The smoke detector buzzes when the smoke density is higher than 600."
Set Triggers	<ol style="list-style-type: none"><li>1. Click <b>Add Trigger</b>.</li><li>2. Select <b>Device Property</b>, and select the device added in <a href="#">Registering a Device</a>.</li><li>3. Select <b>smoke</b> for <b>Service Type</b>, <b>Smoke_Value</b> for <b>Property</b>, <b>&gt;</b> for <b>Operation</b>, and <b>600</b> for <b>Value</b>. Click <b>Trigger Mode</b>. In the dialog box displayed, set <b>Trigger Strategy</b> to <b>Repetition suppression</b> and <b>Data Validity Period (s)</b> to <b>300</b>, and click <b>OK</b>.</li></ol>
Set Actions	<ol style="list-style-type: none"><li>1. Click <b>Add Action</b>.</li><li>2. Select <b>Deliver Commands</b>, and select the device created in <a href="#">Registering a Device</a>.</li><li>3. Select <b>Smoke</b> for <b>Service Type</b>, and <b>Smoke_Control_Beep</b> for <b>Command</b>. Click <b>Configure Parameter</b>. In the dialog box displayed, set <b>Beap</b> to <b>ON</b>, and click <b>OK</b>.</li></ol>


----End

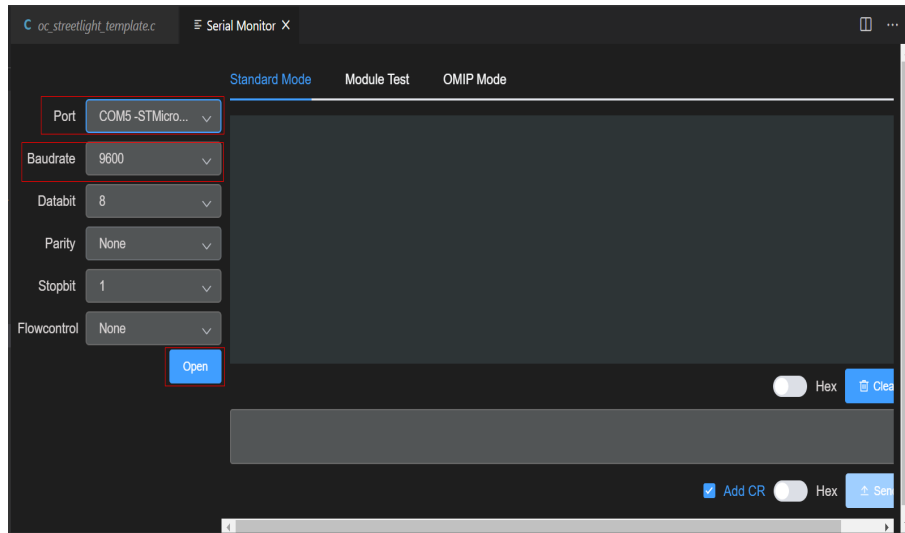
## (Optional) Verification

Spray cooling agents around the smoke detector to simulate the smoke density. When the smoke density is greater than 600, the smoke detector buzzes.

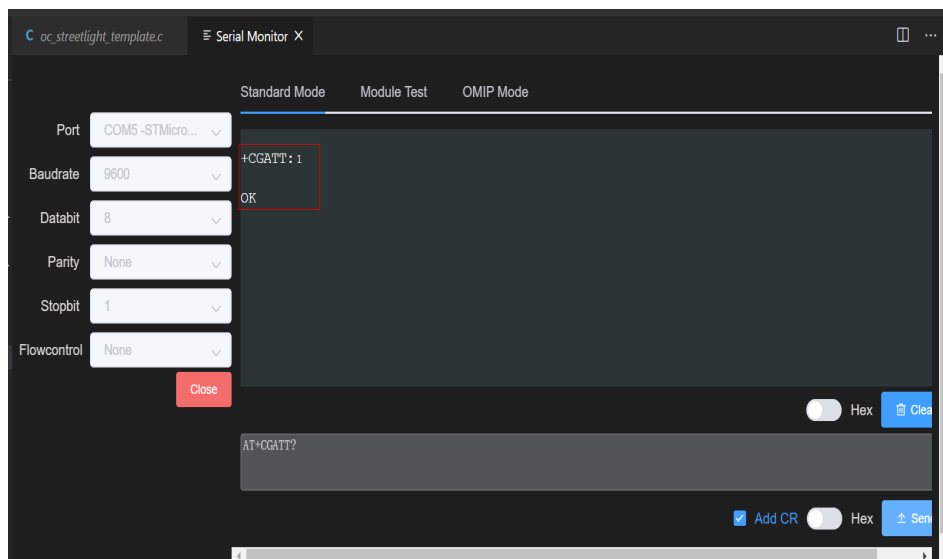
## Locating Module Communication Problems Using AT Commands

When IoT Link is connected to the platform, you can use AT commands to quickly locate the connectivity problem between the module and the cloud. This section describes how to use AT commands to detect common problems of the communications module, for example, the device fails to go online or data fails to be reported.

1. Connect the BearPi development board to the computer and ensure that the driver has been installed. Set the dialing test switch in the upper right corner of the board to the **AT-PC** mode.
2. Click **Serial** on the toolbar at the bottom of Visual Studio Code.  

3. Select the port number obtained in [Step 4.2](#), set **Baudrate** to **9600**, and click **Open**.



4. Enter **AT+CGATT?** and click **Send**. If **+CGATT:1** is returned, the network attach is successful, indicating that the NB-IoT network is normal. If **+CGATT:0** is returned, the network attach fails, indicating that the NB-IoT network is abnormal. In this case, check whether the SIM card is correctly inserted or contact the carrier to check the network status.



**NOTE**

After using the AT commands to detect the module communication, set the dialing test switch to the **AT-MCU** mode so that the collected data can be sent to the platform through the communication module after the console configuration.

In the **AT-PC** mode, the development board communicates with the serial port of the computer, and AT commands are used to read and write data such as the status of the development board. In the **AT-MCU** mode, the development board connects to the network through the SIM card inserted into the module to implement NB-IoT communications.

## 2.4 Connecting and Debugging an NB-IoT Smart Street Light Using a Simulator

### Scenarios

This section uses a smart street light as an example to describe how to use the device and application simulators provided by the IoTDA console to experience data reporting and command delivery.

Assume that:

A street light reports a data message carrying the light intensity (`light_intensity`) and status (`light_status`). The data message is in binary format. The command (`SWITCH_LIGHT`) can be used to remotely control the street light status.

### Prerequisites

- You have [registered a Huawei Cloud account](#).
- You have subscribed to IoTDA. If you have not subscribed to the service, go to the [IoTDA](#) service page, and click **Access Console** to subscribe to the service.

### (Optional) Creating a Resource Space

A resource space is the commissioning space provided by the platform for applications and devices. You can create different resource spaces for different scenarios.

The system provides a preset resource space, where you can develop product models and codecs for devices online. To create a resource space, perform the following steps:

- Step 1** Visit the [IoTDA](#) service page and click **Access Console**.
- Step 2** In the navigation pane, choose **IoTDA Instances**, and click the target instance card.
- Step 3** In the navigation pane, choose **Resource Spaces** and click **Create Resource Space**. In the displayed dialog box, specify **Space Name** and click **OK**.

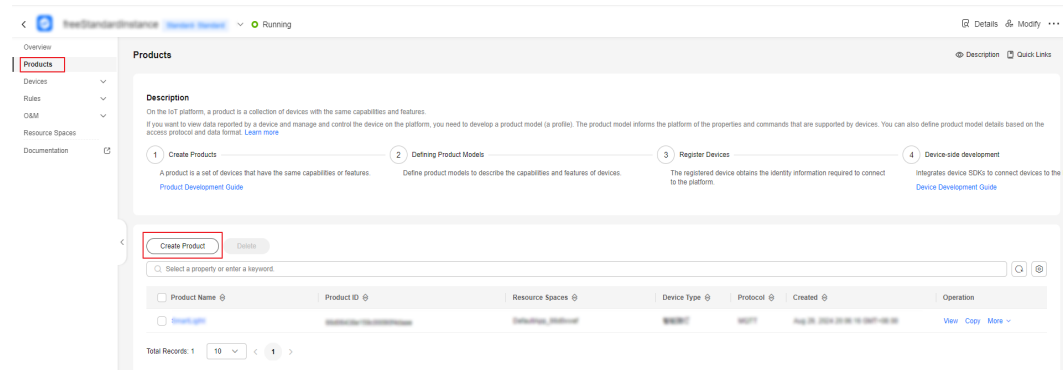
----End

### Creating a Product

You can develop product models and codecs online. The platform provides device and application simulators for you to debug the product models and codecs.

- Step 1** Visit the [IoTDA](#) service page and click **Access Console**. Click the target instance card.
- Step 2** In the navigation pane, choose **Products**.

Figure 2-43 Creating a product



**Step 3** Click **Create Product** to create a product using LwM2M over CoAP. Set the parameters and click **OK**.

**Figure 2-44** Creating a product - CoAP

Basic Information	
Resource Space	Select the resource space to which the product to create belongs.
Product Name	Customize a name, for example, <b>Test_1</b> .
Protocol	Select <b>LwM2M over CoAP</b> .
Data Type	Select <b>Binary</b> . <b>NOTE</b> If <b>Data Type</b> is <b>Binary</b> , codec development is required for the product. If <b>Data Type</b> is <b>JSON</b> , codec development is not required.

Industry	Select <b>Default</b> .
Device Type	streetlamp

----End

## Defining a Product Model

- Step 1** Click the name of the product created in **3** to go to the product details page.
- Step 2** On the **Model Definition** tab page, click **Customize Model** to add a service for the product, and click **OK**.
- **Service ID:** Enter **StreetLight**.
  - **Service Type:** You are advised to set this parameter to the same value as **Service ID**.
  - **Description:** Provide a description, for example, "Define the properties of light intensity and status."
- Step 3** Click the ID of the service added in **2**. On the displayed page, click **Add Property** to define a light intensity property collected by the street light.
- **Property Name:** Enter **light\_intensity**.
  - **Data Type:** Select **Integer**.
  - **Access Permissions:** Select **Read** and **Write**.
  - **Value Range:** Set it to 0 to 100 (light intensity range).



Figure 2-45 Adding a property - light\_intensity

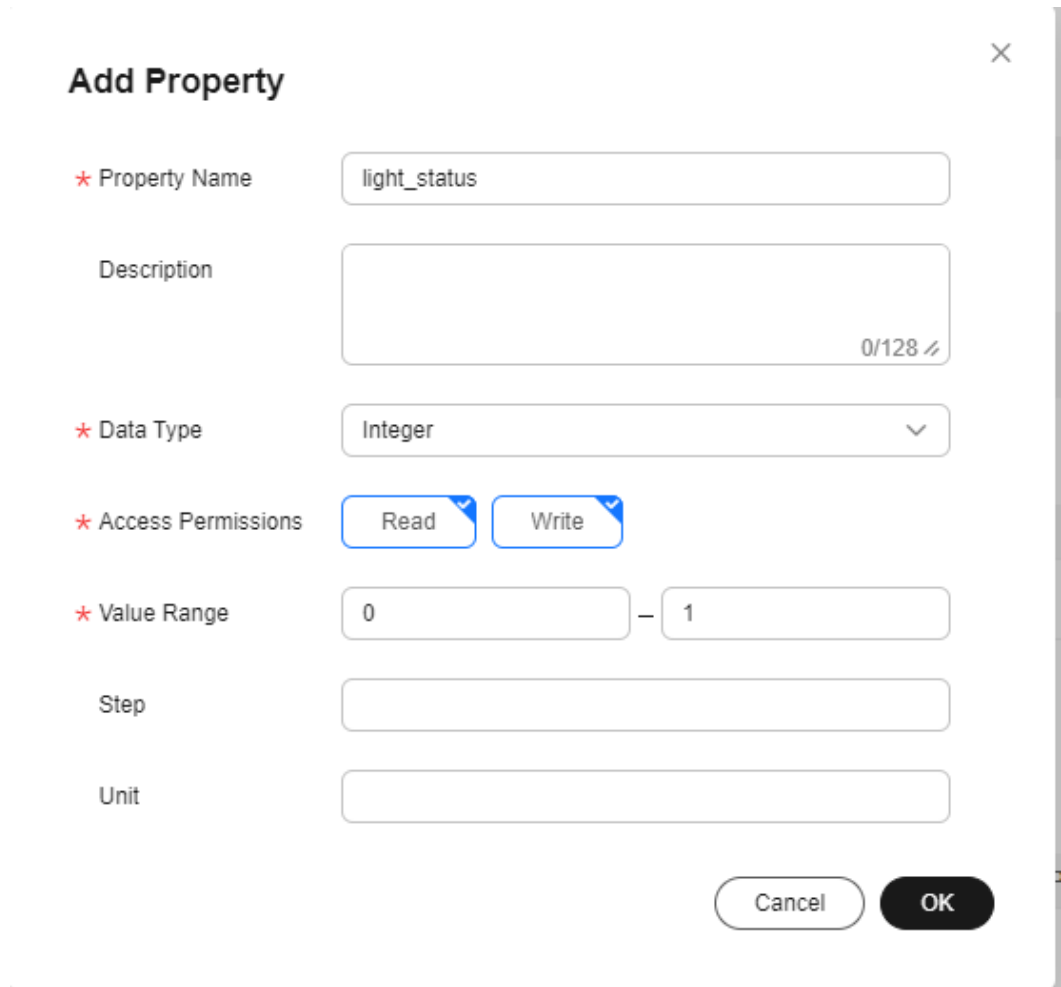
The screenshot shows a modal dialog titled "Add Property" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Property Name:** A text input field containing "light\_intensity".
- Description:** A text area with a character count "0/128" in the bottom right corner.
- Data Type:** A dropdown menu currently set to "Integer".
- Access Permissions:** Two toggle buttons labeled "Read" and "Write", both of which are currently selected (checked).
- Value Range:** Two input fields containing "0" and "100" separated by a minus sign.
- Step:** An empty input field.
- Unit:** An empty input field.
- Buttons:** "Cancel" and "OK" buttons at the bottom right.

**Step 4** Click **Add Property** to define a property as the status of the street light.

- **Property Name:** Enter **light\_status**.
- **Data Type:** Select **Integer**.
- **Access Permissions:** Select **Read** and **Write**.
- **Value Range:** Set it to 0 or 1. **0** indicates that the light is turned off. **1** indicates that the light is turned on.

Figure 2-46 Adding a property - light\_status



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

- Property Name:** A text input field containing "light\_status".
- Description:** A text area with a character count "0/128" and a slash icon in the bottom right corner.
- Data Type:** A dropdown menu set to "Integer".
- Access Permissions:** Two toggle buttons, "Read" and "Write", both of which are currently selected (indicated by blue checkmarks).
- Value Range:** Two input fields containing "0" and "1" separated by a minus sign.
- Step:** An empty input field.
- Unit:** An empty input field.

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

**Step 5** Define the command used to remotely control the street light switch status.

1. Click **Add Command**. In the dialog box displayed, set **Command Name** to **SWITCH\_LIGHT**.
2. Click **Add Command Parameter**. Set **Parameter Name** to **SWITCH\_LIGHT**, **Data Type** to **String**, **Length** to **3**, and **Enumerated Value** to **ON,OFF**. Click **OK**.

Figure 2-47 Adding a command parameter - SWITCH\_LIGHT

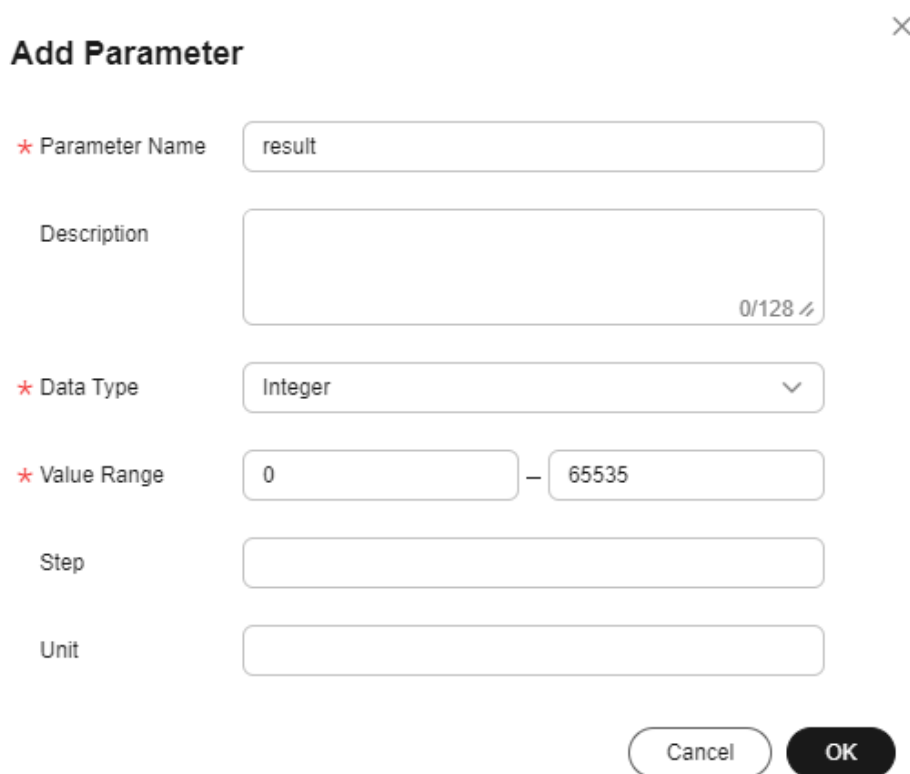
The screenshot shows a dialog box titled "Add Property" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- \* Property Name:** A text input field containing "SWITCH\_LIGHT".
- Description:** A text area with a character count of "0/128" and a refresh icon.
- \* Data Type:** A dropdown menu currently set to "String".
- \* Access Permissions:** Two toggle buttons labeled "Read" and "Write", both of which are currently checked.
- \* Length:** A text input field containing the value "3".
- Enumerated Values:** A text area containing "ON.OFF" with a character count of "6/1,024" and a refresh icon.

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

3. Click **Add Response Parameter**. Set **Parameter Name** to **result** and **Data Type** to **Integer**.

Figure 2-48 Adding a response parameter - result



**Add Parameter** ×

\* Parameter Name

Description  0/128 ↗

\* Data Type  ▾

\* Value Range  –

Step

Unit

Cancel OK

**Step 6** Click **OK**. The product model of the street light is created.

----End

## Developing a Codec Online

For purposes of power conservation, devices generally report binary data. The codec converts the binary data into JSON data based on properties defined in the product model so that the platform and application can identify the data. When an application remotely delivers a command, the platform converts the command in JSON format into binary and delivers the binary data to the device.

### NOTE

If a device reports data in JSON format, you do not need to define a codec.

**Step 1** On the product details page of the street light, click the **Codec Development** tab and click **Develop Codec**.

**Step 2** Click **Add Message** and configure a data reporting message as follows to report street light data:

- **Message Name:** Enter **LightData**.
- **Message Type:** Select **Data reporting**.
- **Add Response Field:** Select this option. After a response field is added, the platform delivers the configured response data to the device when receiving data reported by the device.

- **Response:** Retain the default value **AAAA0000**.

**Figure 2-49** Adding a message

**Add Message** ✕

**Basic Information**

\*Message Name  
LightData

Description  
0/1,024

\*Message Type  
 Data reporting  Command delivery

Add Response Field

**Fields** Add Field

Offset	Field Name	Description	Data Type	Length	Tagged as Address Fi...	Operation
--------	------------	-------------	-----------	--------	-------------------------	-----------

**No table data available.**  
No Fields data available. Add Field first.

Add Field

Response: AAAAA0000

Cancel OK

1. Click **Add Field** to add the **messaged** field, which indicates the message type.

Figure 2-50 Adding a field - messageId

### Add Field ✕

**1** When the field is tagged as address field, the field name is fixed at messageId. The names of other fields cannot be set to messageId.

Tagged as address field ?

\* Field Name

Description  0/1,024 ↕

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

2. Add the **LightIntensity** field to indicate the light intensity. Set **Data Type** to **int8u** (8-bit unsigned integer) and **Length** is **1**.

Figure 2-51 Adding a field - LightIntensity

The screenshot shows a dialog box titled "Add Field" with a close button (X) in the top right corner. Below the title is a checkbox labeled "Tagged as address field" with a help icon (question mark). The main form contains several fields:

- \* Field Name:** A text input field containing "LightIntensity".
- Description:** A text area containing "--Enter--" and a character count "0/1,024" with a refresh icon.
- Data Type (Big Endian):** A dropdown menu showing "int8u".
- Offset:** A text input field containing "1-2" with a help icon.
- \* Length:** A text input field containing "1" with a help icon.
- Default Value:** An empty text input field with a help icon.

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

3. Add the **LightStatus** field to indicate the street light switch status. Set **Data Type** to **int8u** (8-bit unsigned integer) and **Length** is **1**.

Figure 2-52 Adding a field - LightStatus

**Add Field** ×

Tagged as address field ?

\* Field Name

Description  0/1,024 ↕

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

**Step 3** Click **Add Message** again to develop a codec for command delivery messages.

- **Message Name:** Enter **SwitchStatus**.
- **Message Type:** Select **Command delivery**.
- **Add Response Field:** Select this option. After a response field is added, the device reports the command execution result when receiving the command.



Figure 2-53 Adding a message - SwitchStatus

The screenshot shows the 'Add Message' dialog box with the following elements:

- Basic Information:**
  - \*Message Name: SwitchStatus
  - \*Message Type:  Data reporting,  Command delivery
  - Add Response Field
- Description:** A large text area with a character count of 0/1,024.
- Fields:** A table with columns: Offset, Field Name, Description, Data Type, Length, Tagged as Address Fi..., and Operation. An 'Add Field' button is located to the right of the table.
- Response Field:** A table with columns: Offset, Field Name, Description, Data Type, Length, Tagged as Address Fi..., and Operation. An 'Add Response Field' button is located to the right of the table.
- Buttons:** 'Cancel' and 'OK' buttons at the bottom right.

Below the 'Fields' table, there is a message: "No table data available. No Fields data available. Add Field first." with an 'Add Field' button.

1. Click **Add Field** to add the **messaged** field, which indicates the message type.

Figure 2-54 Adding a field - messageld

### Add Field

×

**!** When the field is tagged as address field, the field name is fixed at messageld. The names of other fields cannot be set to messageld.

Tagged as address field ?

\* Field Name

Description  0/1,024 ↕

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

2. Add the **mid** field to associate the delivered command with the command execution result.

Figure 2-55 Adding a command field - mid

✕

### Add Field

**1** When the field is tagged as response ID field, the field name must be fixed at mid. The names of other fields cannot be set to mid.

Tagged as address field ?

Tagged as response ID field ?

\* Field Name

Description  0/1,024 ↕

Data Type (Big Endian)  ▾

Offset  ?

\* Length  ?

Default Value  ?

3. Add the **SwitchStatus** field. Set **Data Type** to **string** and **Length** is **3**.

Figure 2-56 Adding a command field - SwitchStatus

The screenshot shows a dialog box titled "Add Field" with a close button (X) in the top right corner. It contains several configuration options:

- Two checkboxes: "Tagged as address field" and "Tagged as response ID field", both of which are currently unchecked.
- A "Field Name" field with the value "SwitchStatus".
- A "Description" field with the value "--Enter--" and a character count "0/1,024" in the bottom right corner.
- A "Data Type (Big Endian)" dropdown menu set to "string".
- An "Offset" field with the value "3-6" and a help icon (question mark).
- A "Length" field with the value "3" and a help icon (question mark).
- A "Default Value" field which is currently empty and has a help icon (question mark).

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

4. Click **Add Response Field** to configure command delivery responses.
  - Add the **messageId** field to indicate the message type. The command execution result is an upstream message, which is differentiated from the data reporting message by the **messageId** field.

Figure 2-57 Adding a response field - messageId

✕

### Add Field

**i** When the field is tagged as address field, the field name is fixed at messageId. The names of other fields cannot be set to messageId.

Tagged as address field ?

Tagged as response ID field ?

Tagged as command execution state field ?

\* Field Name

Description  0/1,024 ✎

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

- Add the **mid** field to associate the delivered command with the command execution result.

Figure 2-58 Adding a response field - mid

**Add Field** ×

**1** When the field is tagged as response ID field, the field name must be fixed at mid. The names of other fields cannot be set to mid.

Tagged as address field ?

Tagged as response ID field ?

Tagged as command execution state field ?

\* Field Name

Description  0/1,024

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

- Add the **errcode** field to indicate the command execution status. **00** indicates success and **01** indicates failure. If this field is not carried, the command is executed successfully by default.

Figure 2-59 Adding a response field - errcode

### Add Field

×

**i** When the field is tagged as command execution state field, the field name is fixed at errcode. The names of other fields cannot be set to errcode.

Tagged as address field ?

Tagged as response ID field ?

Tagged as command execution state field ?

\* Field Name

Description  0/1,024 ↕

Data Type (Big Endian)  ▼

Offset  ?

\* Length  ?

Default Value  ?

- Add the **result** field to indicate the command execution result.

Figure 2-60 Adding a response field - result

✕

### Add Field

Tagged as address field ?

Tagged as response ID field ?

Tagged as command execution state field ?

\* Field Name

Description

0/1,024 ↗

Data Type (Big Endian)

Offset

?

\* Length

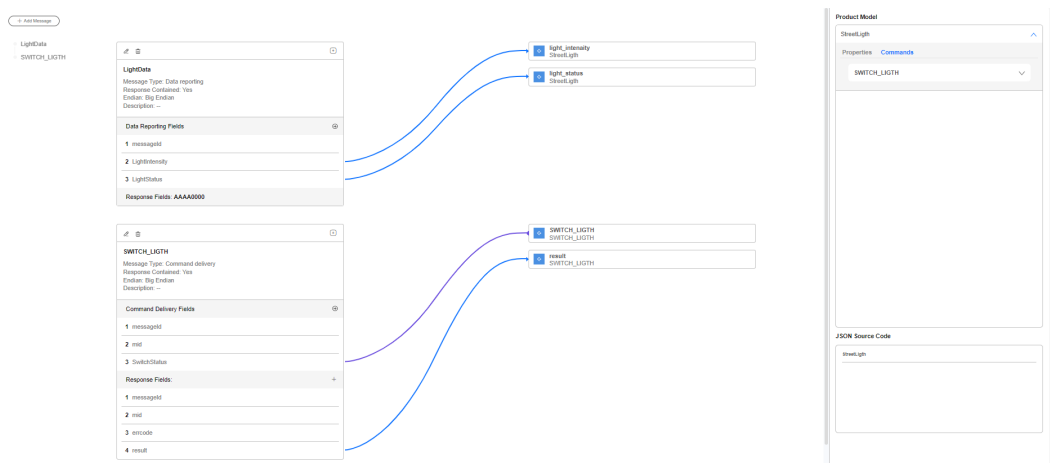
?

Default Value

?

**Step 4** Drag the property and command fields (defined in the product model) in the **Product Model** area on the right to map the fields in the data reporting and command delivery messages defined by the codec.

Figure 2-61 Developing a codec online





**Step 5** Click **Save** in the upper right corner and click **Deploy** to complete codec deployment.

----End

## Debugging the Codec Online Using a Physical Device

Simulators debug the functions of devices and applications. You can debug the defined product model and codec by simulating data reporting and command delivery.

**Step 1** On the product details page of the street light, click **Online Debugging** and click **Add Test Device**.

**Step 2** In the dialog box displayed, set the parameters and click **OK**.

- **Device Type:** Select **Physical device**.
- **Device Name:** Customize a name.
- **Node ID:** Enter the IMEI of the device. The node ID will be carried by the device for device access authentication. You can view the node ID on the NB-IoT module.
- **Secret:** If DTLS is used for access, keep the secret secure.

**Figure 2-62** Adding a test device

**Add Test Device** ×

Device Type  Physical device  Virtual device

Device Name

\* Node ID

Authentication Type  Secret

Secret

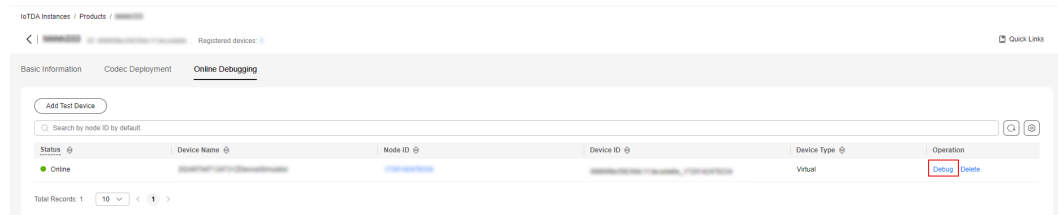
Confirm Secret

**NOTE**

The newly added device is in the inactive state. In this case, online debugging cannot be performed. For details, see [Device Connection Authentication](#). After the device is connected to the platform, perform the debugging.

**Step 3** Click **Debug** to access the debugging page.

**Figure 2-63** Entering debugging



**Step 4** Simulate a scenario where a control command is remotely delivered. In **Application Simulator**, set **Service** to **StreetLight**, **Command** to **SWITCH\_LIGHT**, and **Command Value** to **ON**, and click **Send**. The street lamp is turned on.

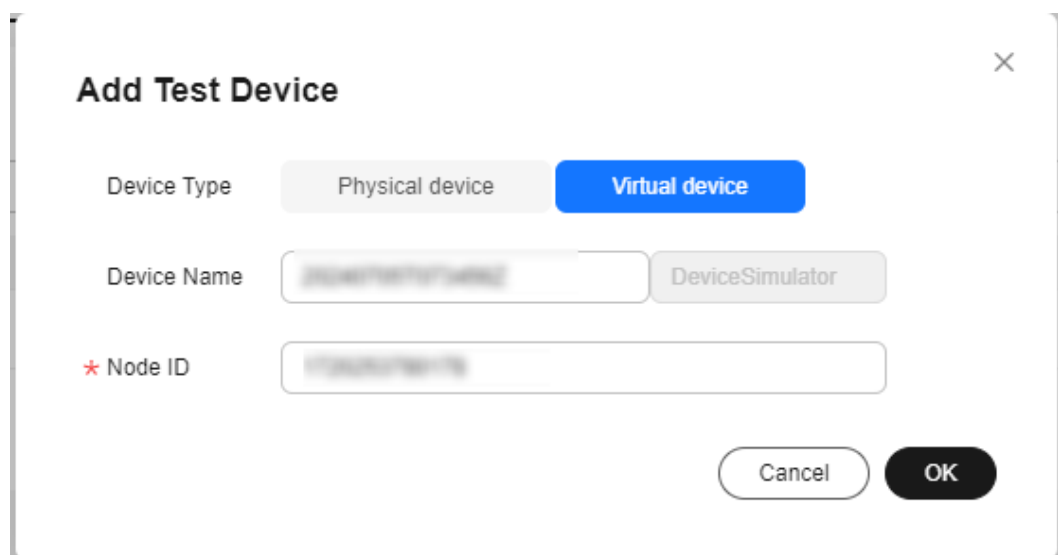
----End

## Debugging the Codec Online Using a Virtual Device

**Step 1** On the product details page of the street light, click **Online Debugging** and click **Add Test Device**.

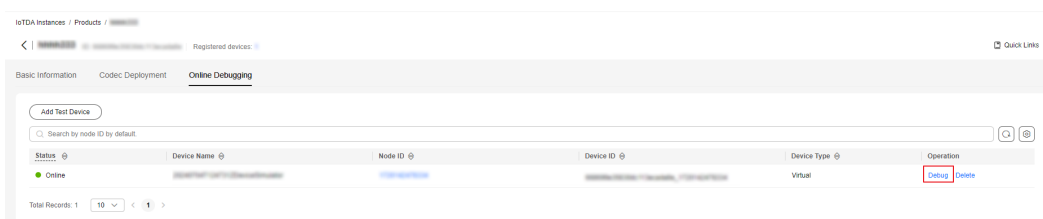
**Step 2** In the dialog box displayed, select **Virtual device** and click **OK**. The virtual device name contains **DeviceSimulator**. Only one virtual device can be created for each product.

**Figure 2-64** Creating a virtual device



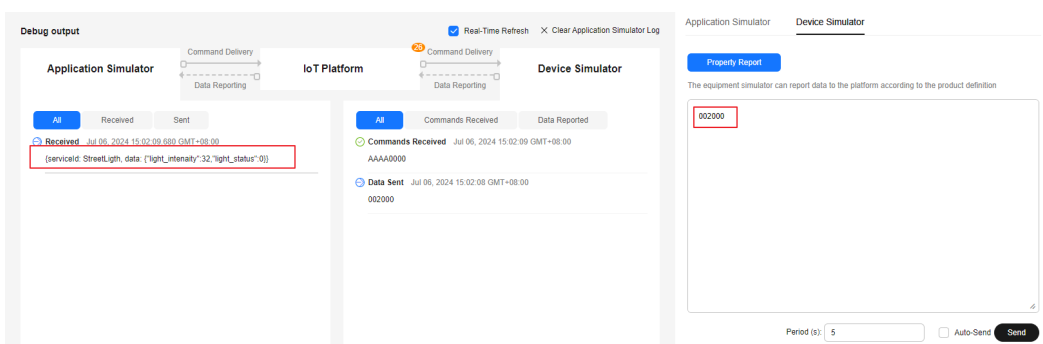
**Step 3** Click **Debug** to access the debugging page.

Figure 2-65 Entering debugging



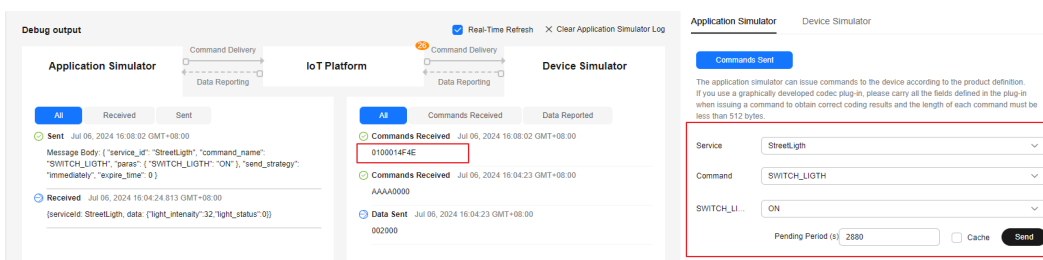
**Step 4** Simulate a data reporting scenario. Specifically, under **Device Simulator**, enter the hexadecimal code stream **002000**, and click **Send**. (The first byte **00** indicates the message ID, the second byte **20** indicates the light intensity, and the third byte **00** indicates the switch status.) Under **Application Simulator**, "**Light\_Intensity**": **32**, "**Light\_Status**": **0** (converted JSON data) should be displayed.

Figure 2-66 Simulating data reporting



**Step 5** Simulate a command delivery scenario. Specifically, under **Application Simulator**, select **StreetLight** for **Service**, **SWITCH\_LIGHT** for **Command**, **ON** for **SWITCH\_LIGHT**, and click **Send**. Under **Device Simulator**, **0100014F4E** (hexadecimal number converted from the ASCLL code) should be displayed.

Figure 2-67 Simulating command delivery



----End

## Debugging Using an Offline Simulator

The **NB-IoT device simulator** is used to simulate the access of NB-IoT devices (devices using LwM2M over CoAP) to the platform for data reporting and command delivery.

**Step 1** Obtain the node ID and secret generated during device registration in [2](#).

**Step 2** Download and decompress the NB-IoT device simulator, and double-click **NB-IoTDeviceSimulator\_zh.jar** to run the simulator.

**NOTE**

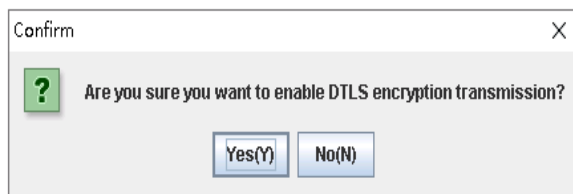
- The simulator can run on Windows, but not macOS.
- Ensure that the JDK has been installed. Otherwise, the JAR file cannot be executed.

Name	Size	Packed	Type	Modified	CRC32
..			File folder		
images			File folder	10/26/2020 9:5...	
NB-IoTDeviceSimulator_en.jar	4,464,721	4,062,301	JAR File	8/7/2017 2:07 ...	B47CBF61
Californium.properties	1,203	607	PROPERTIES File	7/29/2017 12:3...	01A69483
setting.properties	111	108	PROPERTIES File	8/7/2017 3:21 ...	3CEB9C9C

The package contains the following files:

- **NB-IoTDeviceSimulator\_en.jar**: simulator
- **Californium.properties**: simulator configuration file
- **setting.properties**: configuration file for connecting the simulator to the platform

**Step 3** When a message is displayed requesting you to confirm whether to enable DTLS encrypted transmission, click **No** if a secret is not entered during **device registration**, or **Yes** if a secret is entered.



**Step 4** Enter **IP address**, **VerifyCode**, and **psk**, and click **Register Device** to bind the simulator to the platform.

**Note:** If DTLS encryption transmission is disabled, you do not need to enter a PSK.

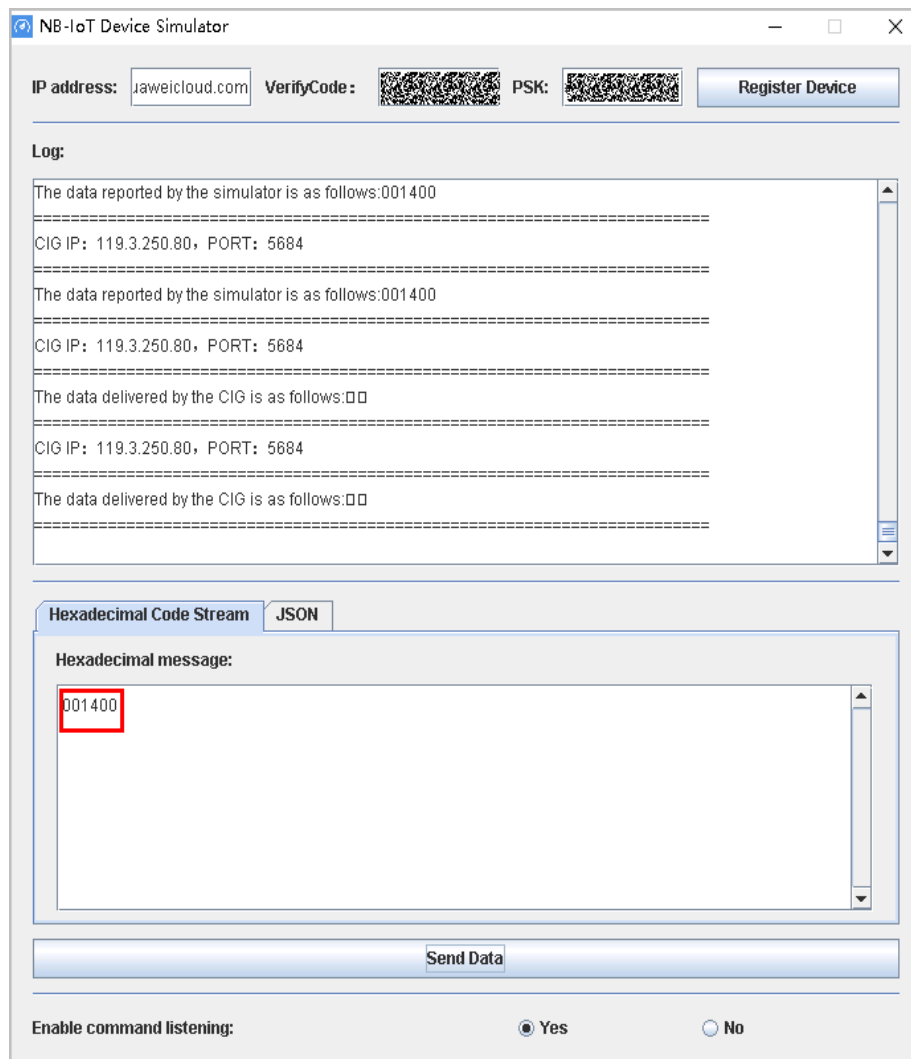
Set the three parameters as follows:

- **IP address**: access domain name displayed on the **IoTDA console**. (You can also use the IP address. You can run the **ping [domain name]** command to obtain the IP address.)
- **VerifyCode**: device ID, for example, **aaaaa11111**.
- **psk**: secret set during device registration, for example, **aaaaa11111aaaaa**.



On the IoTDA console, choose **Devices > All Devices** and view the device status. If the device is displayed as **Online**, the simulator is bound to the platform.

**Step 5** Simulate a data reporting scenario. Specifically, under the NB-IoT device simulator, enter the hexadecimal code stream **001400**, and click **Send Data**. (The first byte 00 indicates the message ID, the second byte 14 indicates the light intensity, and the third byte 00 indicates the switch status.)

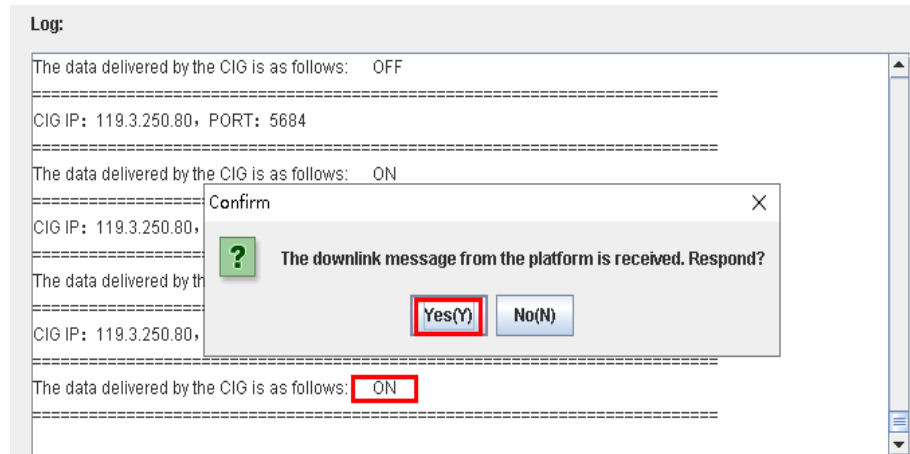


The data is reported successfully. You can return to the console and view the latest reported data on the device details page of **aaaaa11111**. The latest reported data is **"Light\_Intensity": 20, "Light\_Status": 0**.

- Step 6** Simulate the remote command delivery scenario. On the console, locate the target product and click the product to go to the product details page. Click the **Online Debugging** tab and click **Debug** on the right of the **aaaaa11111** device.

Under **Application Simulator**, select **StreetLight** for **Service**, **SWITCH\_LIGHT** for **Command**, **ON** for **SWITCH\_LIGHT**, and click **Send**.

The data delivered by the CIG is **ON** in the log area, and the simulator displays a message indicating that the downstream message from the platform is received and asking whether to respond. Click **Yes**. On the **Application Simulator** page of the IoTDA console, you can see that the command is in the **Delivered** state.



**NOTE**

Strings are parsed using ASCII in the codec. You need to deliver printable characters. Non-printable characters are not displayed in the simulator.

----End

## 2.5 Developing a Protocol Conversion Gateway for Access of Generic-Protocol Devices

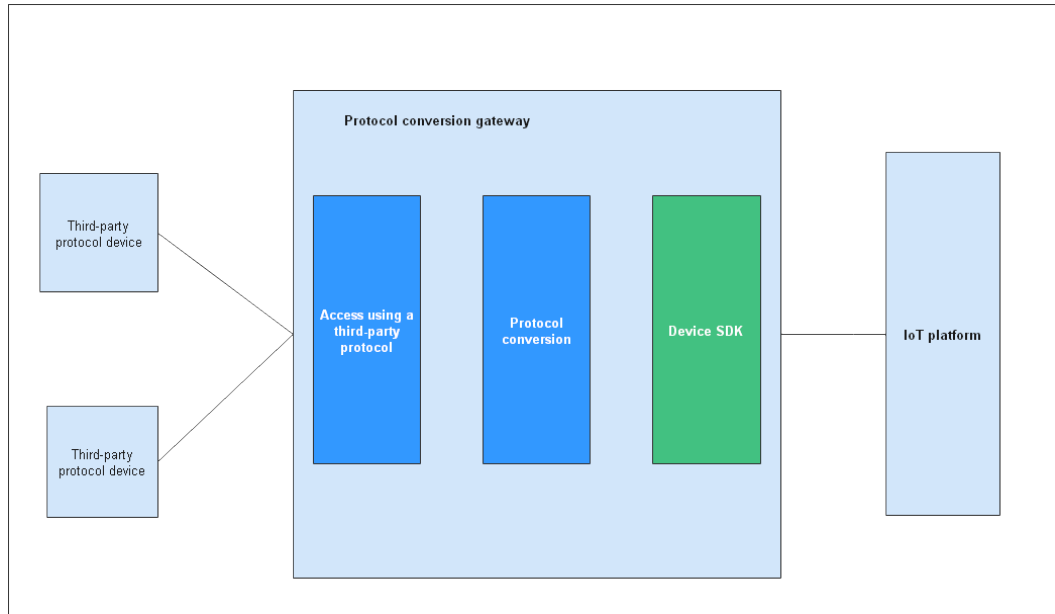
### Scenarios

Currently, the IoT platform supports only standard protocols such as MQTT, HTTP, and LwM2M. If a device uses other protocols (referred to as third-party protocols), it cannot access the platform directly.

To address this issue, protocol conversion must be performed outside the platform. It is recommended that a gateway be used to convert third-party protocols into MQTT. This gateway is called the protocol conversion gateway.

### Implementation Principle

The figure below shows the overall architecture of the solution.

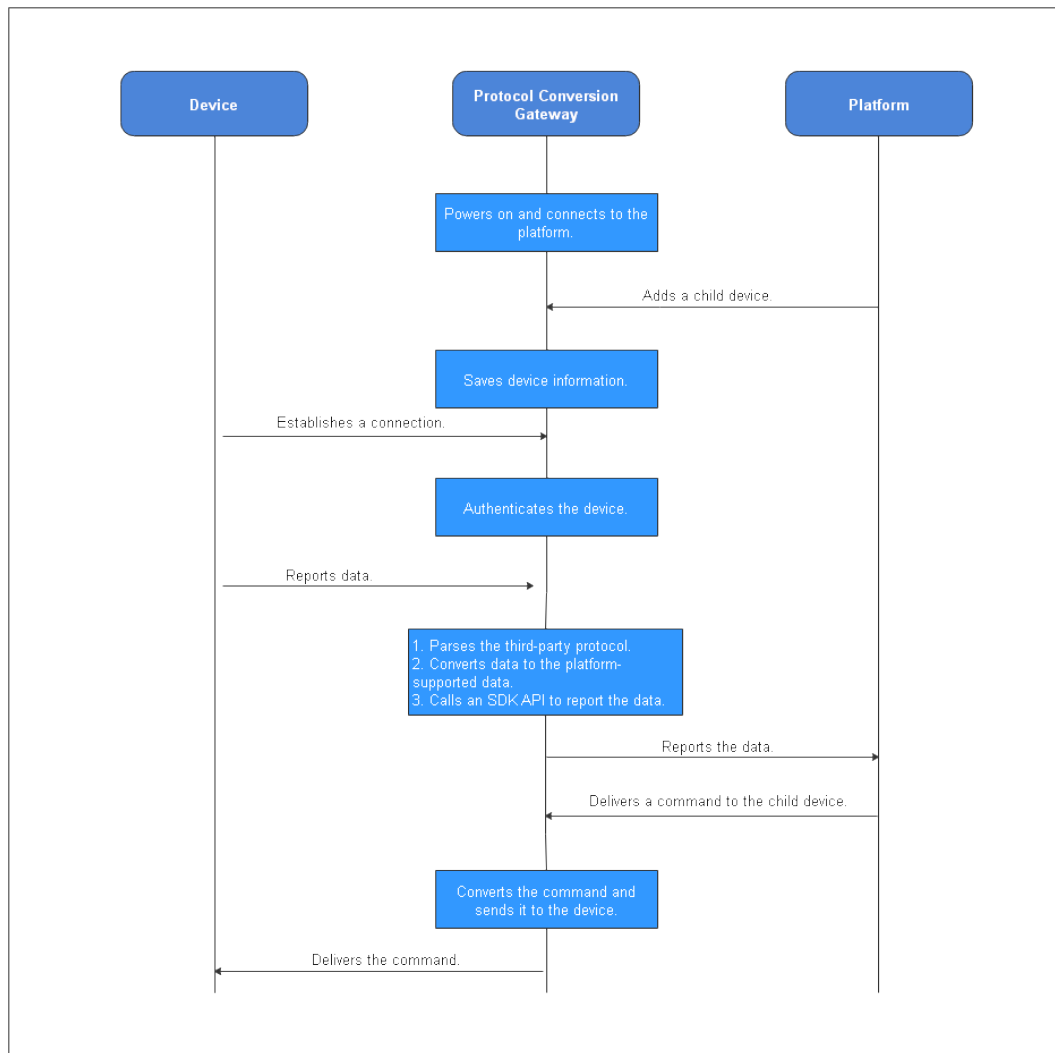


The protocol conversion gateway can be deployed in the cloud or locally. A third-party protocol device is connected to the platform as a child device of the protocol conversion gateway.

The protocol conversion gateway consists of the following components:

1. Third-party protocol access component: This component parses and authenticates third-party protocols.
2. Protocol conversion component: This component converts between third-party protocol data and platform data.
  - In the upstream direction, the component converts third-party protocol data into platform-supported data and invokes the device SDK APIs to report the data.
  - In the downstream direction, the component, after receiving data from the platform, converts the data into third-party protocol data and forwards the data to third-party protocol devices.
3. Device SDK component: The component is the device access SDK provided by the platform and provides common gateway functions. You can implement your own gateways based on the SDK.

## Service Flow



1. Register a gateway with the platform. For details, see [Device Authentication](#).
2. Power on the gateway and connect it to the platform. Obtain the authentication parameters required for connection during [gateway registration](#).
3. Register a child device on the IoTDA console. The platform delivers a child device addition event to the gateway. The gateway saves the child device information and makes the information persistent. (The SDK provides the default persistent implementation. You can customize the implementation.)
4. The child device connects to the gateway, and the gateway authenticates the child device.
5. The child device reports data to the gateway. The gateway converts the data to the format supported by the platform and then calls an SDK API for reporting child device properties or messages to the platform.
6. The platform delivers a command to the child device. The gateway converts the command into a command compliant with the third-party protocol and forwards it to the child device. The child device processes the command.



## Implementation of Protocol Conversion Gateway

For details on the implementation and usage of the gateway, see [IoT Device SDK \(Java\)](#) and [IoT Device SDK \(C\)](#).

## 2.6 Connecting a Device That Uses the X.509 Certificate Based on MQTT.fx

This section uses MQTT.fx as an example to describe how to connect devices to IoTDA using the native MQTT protocol. MQTT.fx is a widely used MQTT client. Using MQTT.fx, you can easily verify whether devices can interact with IoTDA to publish or subscribe to messages.

An X.509 certificate is a digital certificate used for communication entity authentication. IoTDA allows devices to use their X.509 certificates for authentication. The use of X.509 certificate authentication protects devices from being spoofed.

### Prerequisites

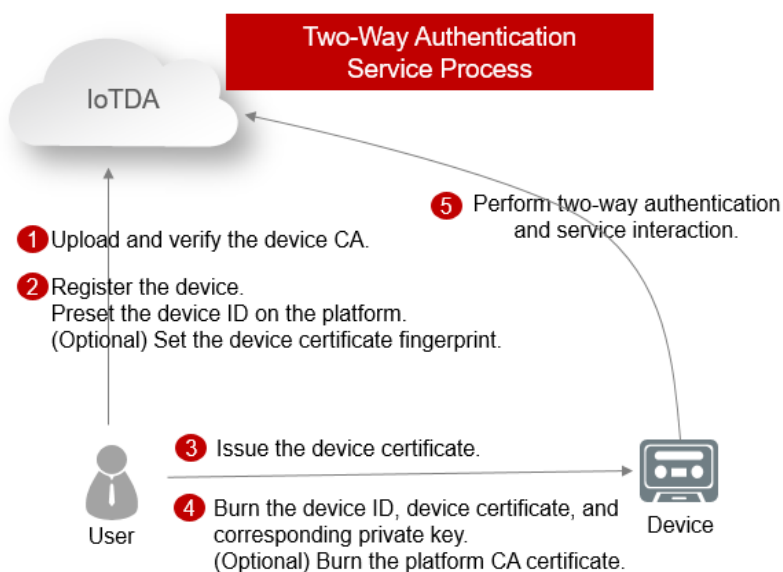
- You have [registered a Huawei Cloud account](#).
- You have subscribed to IoTDA. If you have not subscribed to the service, go to the [IoTDA](#) service page, and click [Access Console](#) to subscribe to the service.

### Constraints

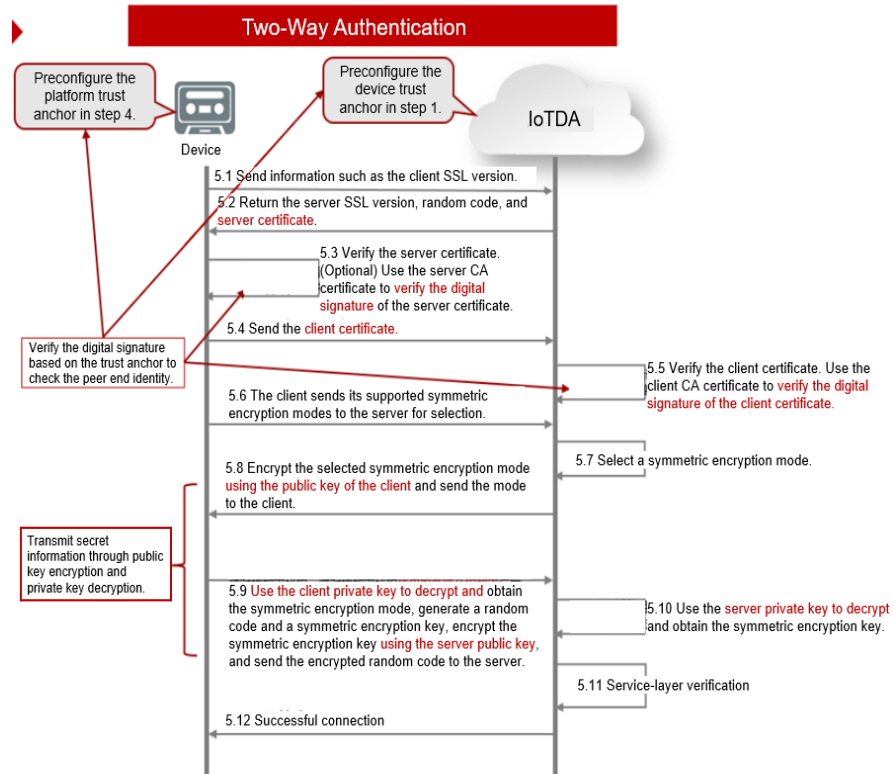
- Only MQTT devices can use X.509 certificates for identity authentication.
- You can upload a maximum of 100 device CA certificates.

### X.509-based Authentication

- The complete two-way certificate authentication is as follows:



- Two-way certificate authentication:



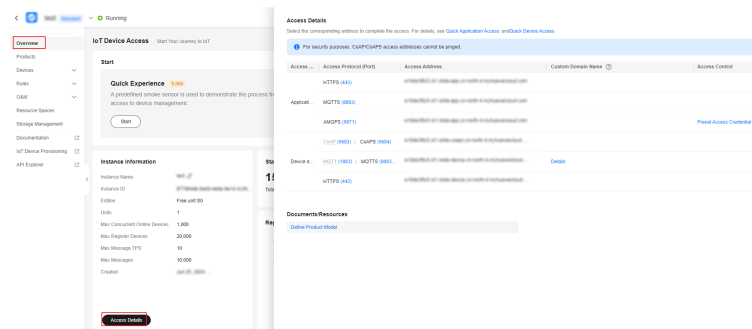
## Obtaining Device Access Information

Perform the following procedure to obtain device access information on the IoTDA console:

**Step 1** Log in to the [IoTDA console](#).

**Step 2** In the navigation pane, choose **Overview** and click **Access Details** in the **Instance Information** area to check the device access information and record domain names and ports.

**Figure 2-68** Obtaining access information



**NOTE**

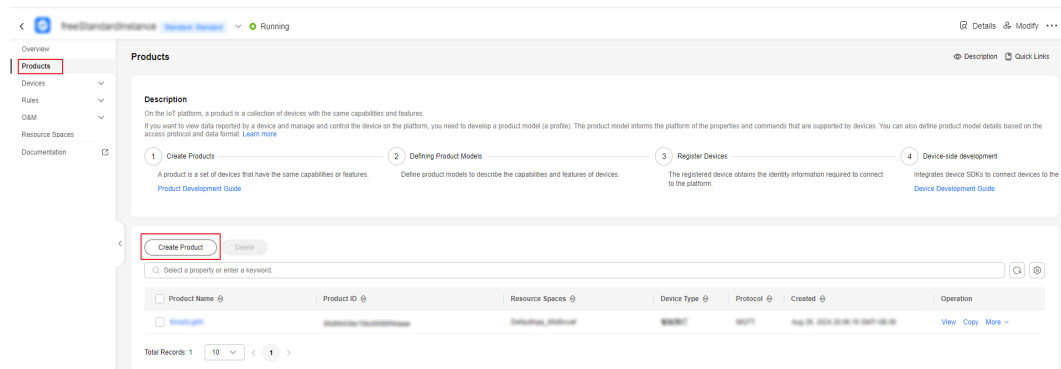
For devices that cannot be connected to the platform using a domain name, run the **ping Domain name** command in the CLI to obtain the corresponding IP address. Then you can connect the devices to the platform using the IP address. The IP address is variable and needs to be set using a configuration item.

----End

## Creating a Product

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

**Figure 2-69** Creating a product



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

Basic Information	
Resource Space	The platform automatically allocates the created product to the default resource space. You can also select an existing resource space from the drop-down list or <b>create one</b> .
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
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, the platform uses the specified product ID. If this parameter is not specified, the platform allocates a product ID.

Figure 2-70 Creating a product - MQTT

**Create Product** ×

\* Resource Space ?

To create a new resource space, you can [go to the instance details page](#).

\* Product Name

Protocol ?

\* Data Type ?

Device Type Selection  Standard profile  Custom

\* Device Type ?

Advanced Settings  Custom Product ID | Description

Product ID ?

Description   
0/128

----End

## Developing a Product Model

- Step 1** Click the created product. The product details page is displayed.
- Step 2** On the **Basic Information** tab page, click **Customize Model** to add services of the product.
- Step 3** Add the **Connectivity** service.
  1. On the **Add Service** page, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.
    - **Service ID:** Enter **Connectivity**.
    - **Service Type:** You are advised to set this parameter to the same value as **Service ID**.

- **Description:** Enter **Connectivity**.

**Figure 2-71** Adding a service - Connectivity

**Add Service** ×

\* Service ID

Service Type  ⓘ

Description  12/128 ↗

2. Choose **Connectivity**, click **Add Property**, enter related information, and click **OK**.
  - **Property Name:** Enter **dailyActivityTime**.
  - **Data Type:** Select **Integer**.
  - **Access Permissions:** Select **Read**.
  - **Value Range:** Set it to 0 to 65535.
  - **Step:** Enter **1**.
  - **Unit:** Enter **s**.

Figure 2-72 Adding a property - dailyActivityTime

**Add Property** ×

\* Property Name

Description  0/128 ↗

\* Data Type  ▼

\* Access Permissions  Read  Write

\* Value Range  –

Step

Unit

**Step 4** Add the **Battery** service.

1. On the **Basic Information** tab page, click **Add Service**, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.
  - **Service ID**: Enter **Battery**.
  - **Service Type**: You are advised to set this parameter to the same value as **Service ID**.
  - **Description**: Enter **Battery**.

Figure 2-73 Adding a service - Battery

### Add Service

\* Service ID

Service Type  ?

Description  7/128 ↕

Cancel

OK

2. Choose **Battery**, click **Add Property**, enter related information, and click **OK**.
  - **Property Name:** Enter **batteryLevel**.
  - **Data Type:** Select **Integer**.
  - **Access Permissions:** Select **Read**.
  - **Value Range:** Set it to 0–100.
  - **Step:** Enter **1**.
  - **Unit:** Enter **%**.

Figure 2-74 Adding a property - batteryLevel (Battery)

**Add Property** ×

\* Property Name

Description  0/128 ↗

\* Data Type  ▾

\* Access Permissions

\* Value Range  –

Step

Unit

----End

## Uploading a Device CA certificate

- Step 1** In the navigation pane, choose **Devices > Device Certificates**. On the **Device CA Certificates** tab page, select a resource space and click **Upload Certificate**.
- Step 2** In the displayed dialog box, click **Select File** to add a file, and then click **OK**.

Figure 2-75 Device CA certificate - Uploading a certificate

**Upload Certificate** ×

\* CA Certificate ?  ×



 NOTE

- Device CA certificates are provided by device vendors. You can **prepare a commissioning certificate** during commissioning. For security reasons, you are advised to replace the commissioning certificate with a commercial certificate during commercial use.
- CA certificates cannot be used to verify server certificates upon expiration. Replace these certificates before expiration dates to ensure that devices can connect to the IoT platform properly.

----End

## Making a Device CA Commissioning Certificate

This section uses the Windows operating system as an example to describe how to use OpenSSL to make a commissioning certificate. The generated certificate is in PEM format.

1. Download and install **OpenSSL**.
2. Open the CLI as user **admin**.
3. Run **cd c:\openssl\bin** (replace **c:\openssl\bin** with the actual OpenSSL installation directory) to access the OpenSSL view.
4. Generate a public/private key pair.  

```
openssl genrsa -out rootCA.key 2048
```
5. Use the private key in the key pair to generate a CA certificate.  

```
openssl req -x509 -new -nodes -key rootCA.key -sha256 -days 1024 -out rootCA.pem
```

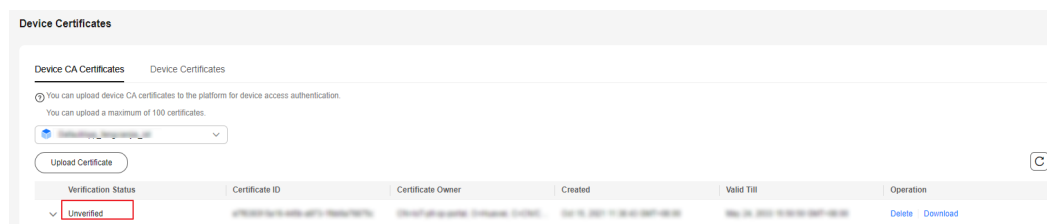
The system prompts you to enter the following information. All the parameters can be customized.

- Country Name (2 letter code) [AU]: country, for example, CN
- State or Province Name (full name) []: state or province, for example, GD
- Locality Name (for example, city) []: city, for example, SZ
- Organization Name (for example, company) []: organization, for example, Huawei
- Organizational Unit Name (for example, section) []: organization unit, for example, IoT
- Common Name (e.g. server FQDN or YOUR name) []: common name, for example, zhangsan
- Email Address []: email address, for example, 1234567@163.com

Obtain the generated CA certificate **rootCA.pem** from the **bin** folder in the OpenSSL installation directory.

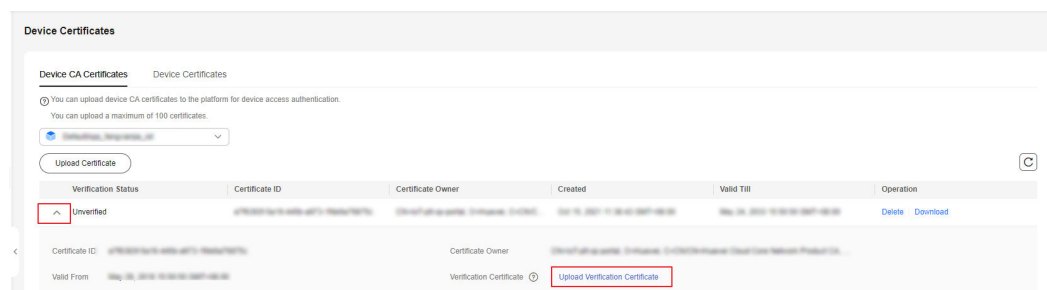
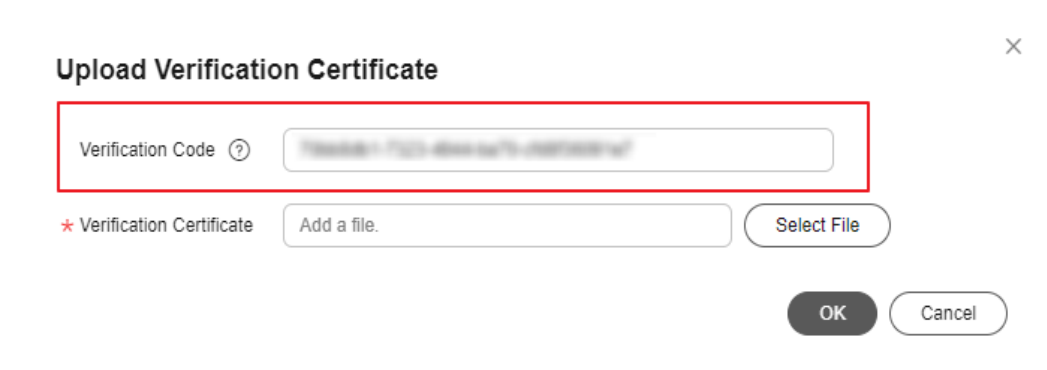
## Uploading a Verification Certificate

If the uploaded certificate is a commissioning certificate, the certificate status is **Unverified**. In this case, upload a verification certificate to verify that you have the CA certificate.

**Figure 2-76** Device CA certificate - Unverified certificate

The verification certificate is created based on the private key of the device CA certificate. Perform the following operations to create a verification certificate:

**Step 1** Obtain the verification code to verify the certificate.

**Figure 2-77** Device CA certificate - Verifying a certificate**Figure 2-78** Device CA certificate - Obtaining the verification code

**Step 2** Generate a key pair for the verification certificate.

```
openssl genrsa -out verificationCert.key 2048
```

**Step 3** Create a certificate signing request (CSR) for the verification certificate.

```
openssl req -new -key verificationCert.key -out verificationCert.csr
```

The system prompts you to enter the following information. Set **Common Name** to the verification code and set other parameters as required.

- Country Name (2 letter code) [AU]: country, for example, CN
- State or Province Name (full name) []: state or province, for example, GD
- Locality Name (for example, city) []: city, for example, SZ
- Organization Name (for example, company) []: organization, for example, Huawei

- Organizational Unit Name (for example, section) []: organization unit, for example, IoT
- Common Name (e.g. server FQDN or YOUR name) []: verification code for verifying the certificate. For details on how to obtain the verification code, see [Step 1](#).
- Email Address []: email address, for example, 1234567@163.com
- Password[]: password, for example, 1234321
- Optional Company Name[]: company name, for example, Huawei

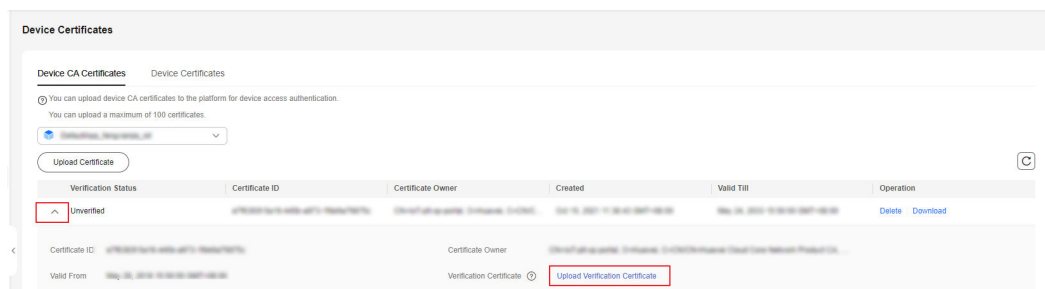
**Step 4** Use the CSR to create a verification certificate.

```
openssl x509 -req -in verificationCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out verificationCert.pem -days 500 -sha256
```

Obtain the generated verification certificate **verificationCert.pem** from the **bin** folder of the OpenSSL installation directory.

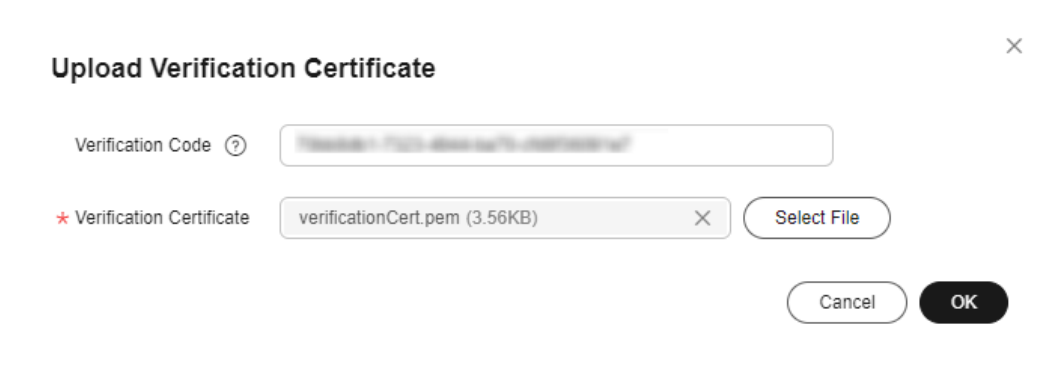
**Step 5** Locate the target certificate, click , and click **Upload Verification Certificate**.

**Figure 2-79** Device CA certificate - Verifying a certificate



**Step 6** In the displayed dialog box, click **Select File** to add a file, and then click **OK**.

**Figure 2-80** Device CA certificate - Uploading a verified certificate



After the verification certificate is uploaded, the certificate status changes to **Verified**, indicating that you have the CA certificate.

----End

## Presetting an X.509 Certificate

Before registering an X.509 device, preset the X.509 certificate issued by the CA on the device.

 NOTE

The X.509 certificate is issued by the CA. If no commercial certificate issued by the CA is available, you can [create a device CA commissioning certificate](#).

### Creating an X.509 Commissioning Certificate

1. Run **cmd** as user **admin** to open the CLI and run **cd c:\openssl\bin** (replace **c:\openssl\bin** with the actual OpenSSL installation directory) to access the OpenSSL view.

2. Generate a public/private key pair.

```
openssl genrsa -out deviceCert.key 2048
```

3. Create a CSR.

```
openssl req -new -key deviceCert.key -out deviceCert.csr
```

The system prompts you to enter the following information. All the parameters can be customized.

- Country Name (2 letter code) [AU]: country, for example, CN
- State or Province Name (full name) []: state or province, for example, GD
- Locality Name (for example, city) []: city, for example, SZ
- Organization Name (for example, company) []: organization, for example, Huawei
- Organizational Unit Name (for example, section) []: organization unit, for example, IoT
- Common Name (e.g. server FQDN or YOUR name) []: common name, for example, zhangsan
- Email Address []: email address, for example, 1234567@163.com
- Password[]: password, for example, 1234321
- Optional Company Name[]: company name, for example, Huawei

4. Create a device certificate using CSR.

```
openssl x509 -req -in deviceCert.csr -CA rootCA.pem -CAkey rootCA.key -CAcreateserial -out deviceCert.pem -days 500 -sha256
```

Obtain the generated device certificate **deviceCert.pem** from the **bin** folder in the OpenSSL installation directory.

## Registering a Device Authenticated by an X.509 Certificate

**Step 1** Log in to the [IoTDA console](#).

**Step 2** In the navigation pane, choose **Devices > All Devices**, click **Register Device**, set parameters based on the table below, and click **OK**.

**Figure 2-81** Device - Registering an X.509 device

Parameter	Description
Resource Space	Select the resource space to which a device belongs.
Product	Select the product to which the device belongs. If no product is available, <a href="#">create a product</a> first.
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the device name.
Authentication Type	<b>X.509 certificate:</b> The device uses an X.509 certificate for identity verification.
Fingerprint	This parameter is displayed when <b>Authentication Type</b> is set to <b>X.509 certificate</b> . Import the fingerprint corresponding to the <a href="#">preset device certificate on the device side</a> . You can run <b>openssl x509 -fingerprint -sha256 -in deviceCert.pem</b> in the OpenSSL view to query the fingerprint. <b>Note: Delete the colon (:)</b> from the obtained fingerprint when filling it. <pre>[root@es-iot-v12-2-jump-cert1223]# openssl x509 -fingerprint -sha256 -in deviceCert.pem SHA256 Fingerprint=f7:91:90:45:88:88:37:E6:A7:E7:70:4A:90:75:F3:87:0A:27:5B:7C:49:3E:FE:59:7A:6F:4F:08:40:F8:54:E8</pre>

----End

## Performing Connection Authentication

You can activate the device registered with the platform by using MQTT.fx. For details, see [Device Connection Authentication](#).

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

 **NOTE**

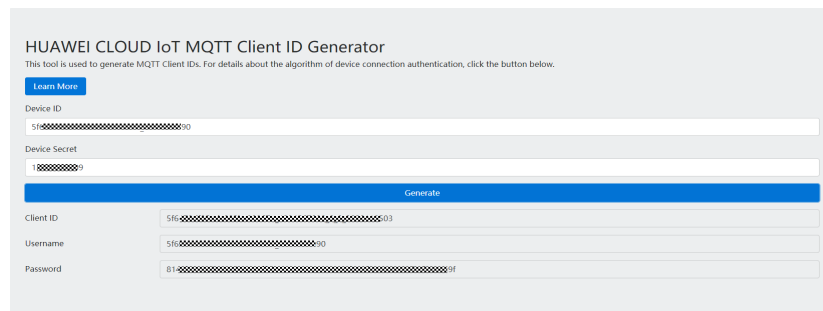
- Install the latest MQTT.fx tool. [Download](#) it.
- MQTT.fx 1.7.0 and earlier versions have problems in processing topics containing \$. Use the latest version for test.

**Step 2** Go to the [IoTDA client ID generator page](#), enter the device ID and secret generated after [registering a device](#) to generate connection information (including **ClientId**, **Username**, and **Password**).

 **NOTE**

You can set **DeviceSecret** to any value, for example, **12345678**.

**Figure 2-82** Obtaining ClientId



The screenshot displays the 'HUAWEI CLOUD IoT MQTT Client ID Generator' web page. At the top, there is a 'Learn More' button and a brief description of the tool's purpose. Below this, there are two input fields: 'Device ID' and 'Device Secret', both containing masked text. A blue 'Generate' button is positioned below these fields. The output section shows three fields: 'Client ID' (masked), 'Username' (masked), and 'Password' (masked).

Parameter	Mandatory	Type	Description
ClientId	Yes	String(256)	<p>The value of this parameter consists of a device ID, device type, password signature type, and timestamp. They are separated by underscores (_).</p> <ul style="list-style-type: none"> <li>• Device ID: A device ID uniquely identifies a device and is generated when the device is registered with IoTDA. The value usually consists of a device's product ID and node ID which are separated by an underscore (_).</li> <li>• Device type: The value is fixed at <b>0</b>, indicating a device ID.</li> <li>• Password signature type: The length is 1 byte, and the value can be <b>0</b> or <b>1</b>. <ul style="list-style-type: none"> <li>- <b>0</b>: The timestamp is not verified using the HMAC-SHA256 algorithm.</li> <li>- <b>1</b>: The timestamp is verified using the HMAC-SHA256 algorithm.</li> </ul> </li> <li>• Timestamp: The UTC time when the device was connected to IoTDA. The format is YYYYMMDDHH. For example, if the UTC time is 2018/7/24 17:56:20, the timestamp is <b>2018072417</b>.</li> </ul>
Username	Yes	String(256)	Device ID.

Each device performs authentication using the MQTT CONNECT message, which must contain all information of the client ID. After receiving a CONNECT message, IoTDA checks the authentication type and password digest algorithm of the device.

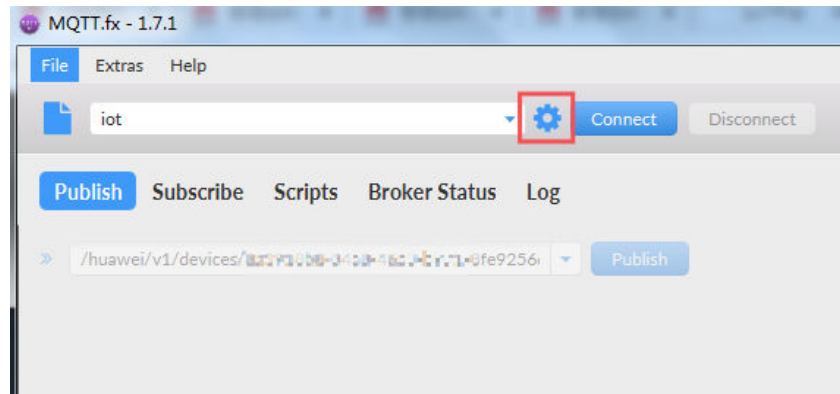
The generated client ID is in the format ***Device ID\_0\_0\_Timestamp***. By default, the timestamp is not verified.

- If the timestamp needs to be verified using the HMAC-SHA256 algorithm, the platform checks whether the message timestamp is consistent with the platform time and then checks whether the password is correct.
- If the timestamp does not need to be verified using the HMAC-SHA256 algorithm, the timestamp must also be contained in the CONNECT message, but the platform does not check whether the time is correct. In this case, only the password is checked.

If the authentication fails, the platform returns an error message and automatically disconnects the MQTT connections.

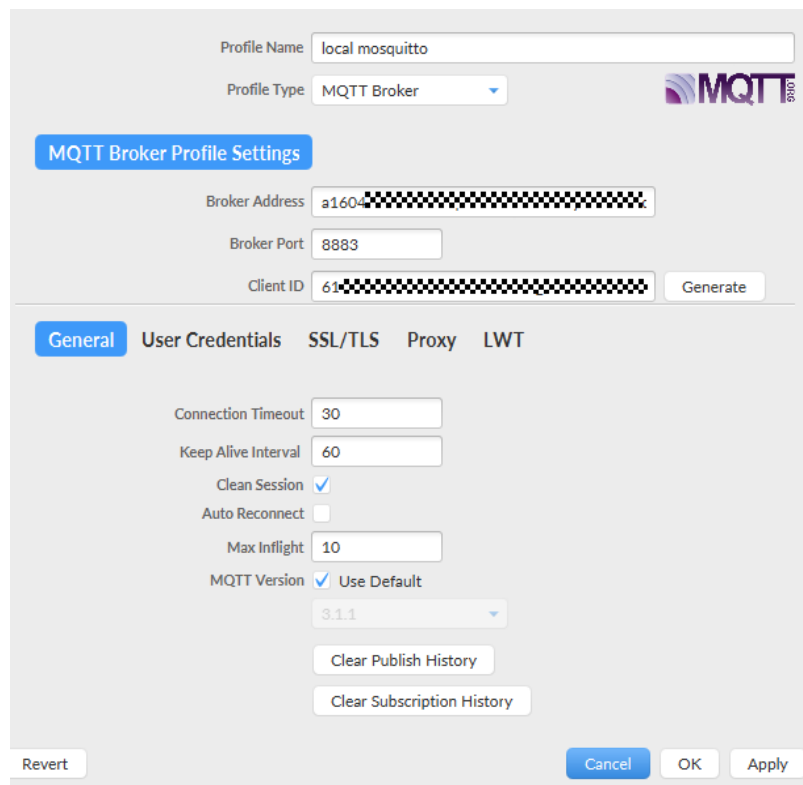
**Step 3** Open the MQTT.fx tool and click the setting icon.

**Figure 2-83** Settings



**Step 4** Enter **Connection Profile** information.

**Figure 2-84** Using default settings for parameters on the General tab page



Parameter	Description
Broker Address	Enter the <b>device access address</b> (domain name) obtained from the IoTDA console. If the device cannot be connected using a domain name, enter the IP address obtained in <b>2</b> .
Broker Port	Enter <b>8883</b> .
Client ID	Enter the device client ID obtained in <b>2</b> .



**Step 5** Click **User Credentials** and specify **User Name**.

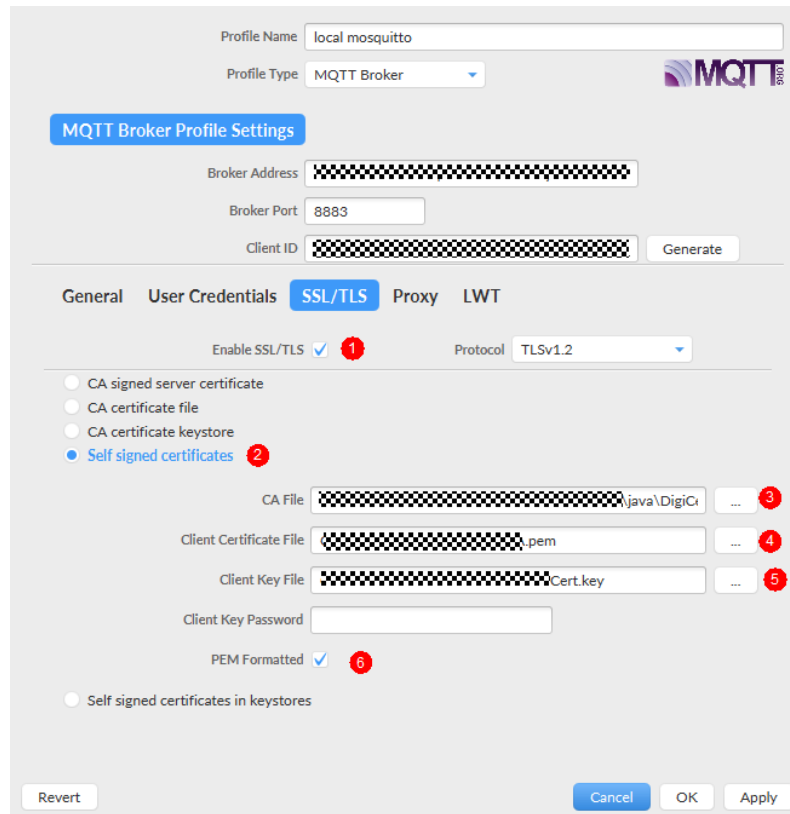
**Figure 2-85** Entering the device ID

The screenshot shows the 'MQTT Broker Profile Settings' dialog box. At the top, 'Profile Name' is 'local mosquitto' and 'Profile Type' is 'MQTT Broker'. A blue button labeled 'MQTT Broker Profile Settings' is present. Below, 'Broker Address' is masked, 'Broker Port' is '8883', and 'Client ID' is masked with a 'Generate' button. The 'User Credentials' tab is active, showing 'User Name' as a masked device ID and an empty 'Password' field. At the bottom are 'Revert', 'Cancel', 'OK', and 'Apply' buttons.

Parameter	Description
User Name	Enter the device ID obtained in 2.
Password	Leave it blank when the X.509 certificate is used for authentication.

**Step 6** Click **SSL/TLS**, set authentication parameters, and click **Apply**. Select **Enable SSL/TLS**, select **Self signed certificates**, and enter the certificate information.

**Figure 2-86** Setting SSL/TLS parameters



**NOTE**

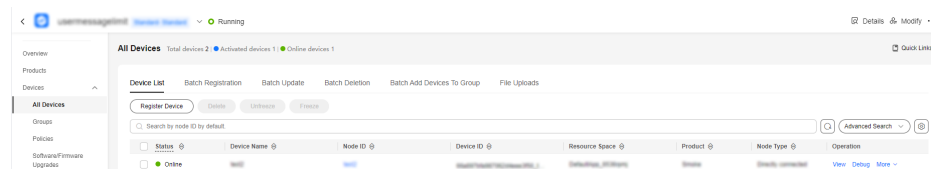
**CA File:** corresponding CA certificate. Download the certificate from [Obtaining Resources](#) and load the PEM certificate.

**Client Certificate File:** device certificate (deviceCert.pem).

**Client Key File:** private key (deviceCert.key) of the device.

**Step 7** Click **Connect**. If the device authentication is successful, the device is displayed online on the platform.

**Figure 2-87** Device list - Device online status



----End

## Reporting Data

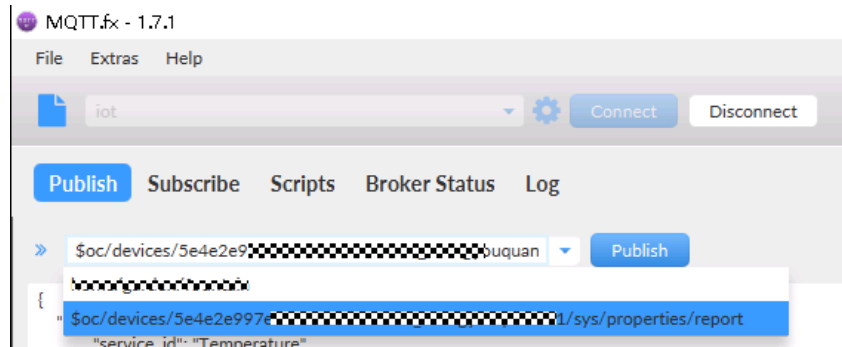
Use MQTT.fx to report data to the platform. For details, see [Reporting Device Properties](#) in the *API Reference*.

If the device reports data through the MQTT channel, the data must be sent to the specified topic. The format of the topic for reporting the data is **\$oc/devices/**

`{device_id}/sys/properties/report`, where `{device_id}` is the device ID returned after device registration.

- Step 1** Enter the API address in the format of "`$oc/devices/{device_id}/sys/properties/report`", for example, `$oc/devices/5e4e2e92ac-164aefa8fouquan1/sys/properties/report`.

**Figure 2-88** Entering the API address



- Step 2** Enter the data to report.

**Request parameters**

Parameter	Mandatory	Type	Description
services	Yes	List<ServiceProperty>	Service data list. (For details, see the <b>ServiceProperty</b> structure below.)

ServiceProperty structure

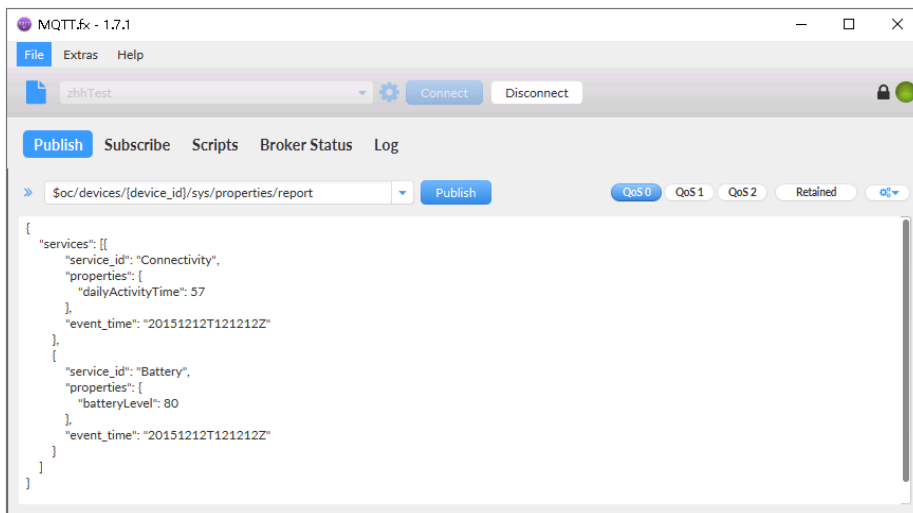
Parameter	Mandatory	Type	Description
service_id	Yes	String	Service ID.
properties	Yes	Object	Service properties, which are defined in the product model of the device.
event_time	No	String	UTC time when the device reports data. The format is yyyyMMddTHH:mm:ssZ, for example, <b>20161219T114920Z</b> . If this parameter is not carried in the reported data or is in incorrect format, the time when IoTDA receives the data is used.

**Example request**

```
{
  "services": [{
```

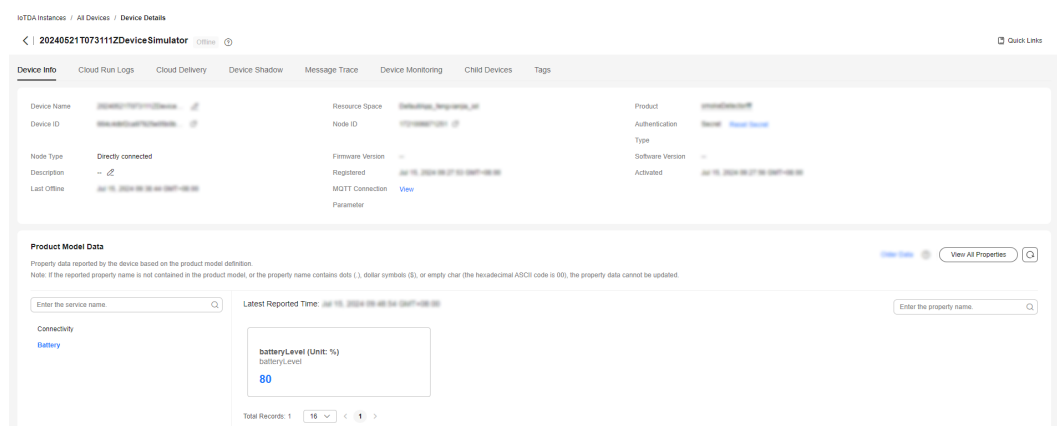
```
"service_id": "Connectivity",  
  "properties": {  
    "dailyActivityTime": 57  
  },  
  "event_time": "20151212T121212Z"  
},  
{  
  "service_id": "Battery",  
  "properties": {  
    "batteryLevel": 80  
  },  
  "event_time": "20151212T121212Z"  
}  
}
```

Figure 2-89 Example request



**Step 3** Click **Publish**. Then you can check whether the device successfully reports data on the platform.

Figure 2-90 Viewing reported property - batteryLevel



----End

## 2.7 Connecting to IoTDA Based on the BearPi-HM\_Nano Development Board and OpenHarmony 3.0

This section uses BearPi-HM\_Nano development board as an example to describe how to connect an OpenHarmony 3.0 device to IoTDA using the huaweicloud\_iot\_link SDK.

### Prerequisites

- You have [registered a Huawei Cloud account](#).
- You have subscribed to IoTDA. If you have not subscribed to the service, go to the [IoTDA](#) service page, and click **Access Console** to subscribe to the service.

### Hardware Environment

You need to prepare a BearPi-HM\_Nano development board, E53\_IA1 expansion module, Type-C data cable, and PC.

### Software Environment

Set up the environment in IDE-based mode or CLI-based mode. The BearPi-HM\_Nano main control chip is Hi3861. Install the corresponding environment.

#### NOTE

If you install `gcc_riscv32` in the environment of the Hi3861 development board, download the `gcc_riscv32` image. Otherwise, some plug-ins may fail to be downloaded or installed.

#### NOTICE

It may take a long time to download a large amount of open-source code. Therefore, reserve sufficient time.

### Creating a Product

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

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

Basic Information	
Resource Space	The platform automatically allocates the created product to the default resource space. You can also select an existing resource space from the drop-down list or <a href="#">create one</a> .
Product Name	Customize the value. The product name must be unique in the same resource space. The value can contain up to 64 characters. Only letters, digits, and special characters ( <code>_?'#(),.&amp;%@!-</code> ) are allowed.

Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
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, the platform uses the specified product ID. If this parameter is not specified, the platform allocates a product ID.

Figure 2-91 Creating a product - MQTT

The screenshot shows a 'Create Product' dialog box with the following fields and options:

- Resource Space**: A dropdown menu with a question mark icon. Below it, a note says: "To create a new resource space, you can [go to the instance details page](#)."
- Product Name**: A text input field.
- Protocol**: A dropdown menu with a question mark icon, currently set to "MQTT".
- Data Type**: A dropdown menu with a question mark icon, currently set to "JSON".
- Device Type Selection**: Two buttons: "Standard profile" (disabled) and "Custom" (active).
- Device Type**: A text input field.
- Advanced Settings**: A section header with an upward arrow, containing:
  - Product ID**: A text input field with a question mark icon.
  - Description**: A text area with a character count "0/128" and a slash icon.

At the bottom right, there are two buttons: "Cancel" and "OK".

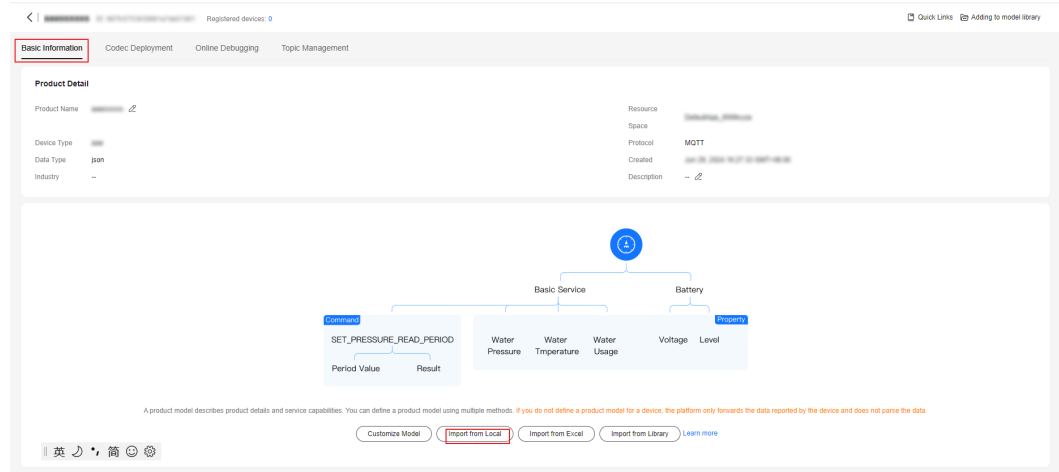
----End

## Developing a Product Model

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

**Step 2** On the **Basic Information** tab page, upload the model file [BearPi\\_Agriculture.zip](#).

**Figure 2-92** Uploading a product model - MQTT



**NOTE**

1. In the product list, click the name of a product to access its details page. On the displayed page, you can view basic product information, such as the product ID, product name, device type, data format, manufacturer name, resource space, and protocol type.
2. You can click **Delete** to delete a product that is no longer used. After the product is deleted, its resources such as the product models and codecs will be cleared. Exercise caution when deleting a product.

----End

## Registering a Device

**Step 1** Log in to the [IoTDA console](#).

**Step 2** In the navigation pane, choose **Devices > All Devices**, click **Register Device**, set parameters based on the table below, and click **OK**.



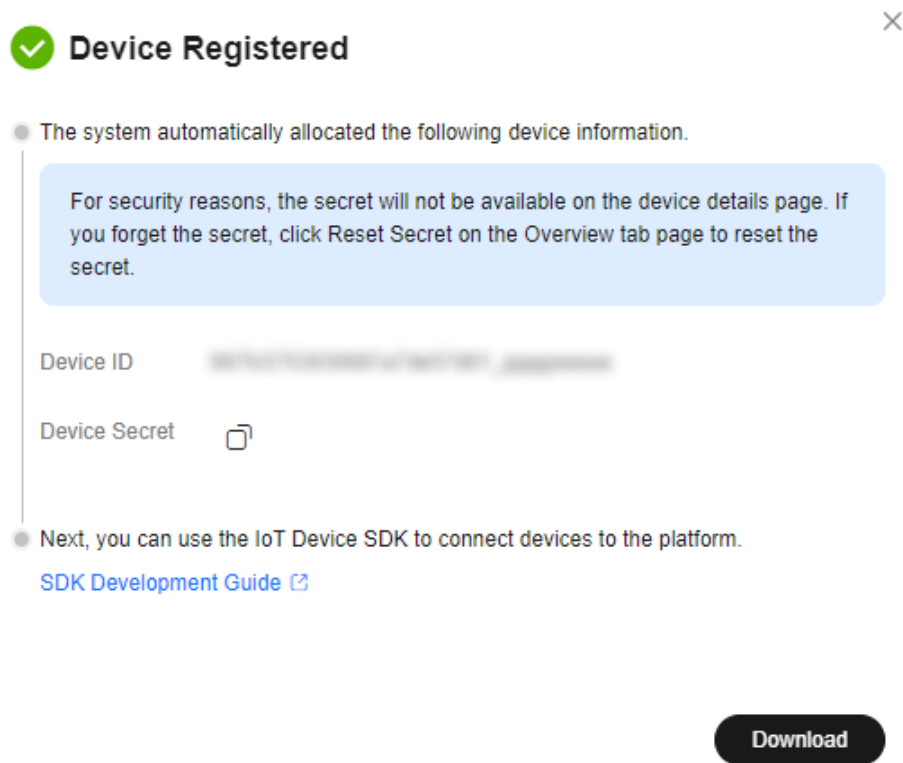
**Figure 2-93** Registering a device - MQTT

The screenshot shows a 'Register Device' dialog box with the following fields and options:

- Resource Space:** A dropdown menu with a question mark icon.
- Product:** A dropdown menu showing 'Test\_1' and a question mark icon. Below it, a note states: 'Mqtt devices have subscribed to the platform preset topic by default. [Subscribed topics](#)'.
- Node ID:** A text input field containing 'Test\_1' and a question mark icon.
- Device ID:** A text input field with a question mark icon.
- Device Name:** A text input field with a tooltip that says 'Enter and allow'.
- Description:** A text area with a character count '0/2,048'.
- Authentication Type:** Radio buttons for 'Secret' (selected) and 'X.509 certificate'.
- Secret:** A password input field with an eye icon.
- Confirm Secret:** A password input field with an eye icon.
- Buttons:** 'Cancel' and 'OK' buttons at the bottom right.

Parameter	Description
Resource Space	Select the resource space to which a device belongs.
Product	Select the product to which the device belongs. If no product is available, <b>create a product</b> first.
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string of 4 to 64 characters. Only letters, digits, underscores (_), and hyphens (-) are allowed.
Device Name	Customize the device name.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

Figure 2-94 Device registered



**NOTE**

1. Save the device ID and secret. They are used for authentication when the device attempts to access IoTDA.
2. If the secret is lost, you can [reset the secret](#). The secret generated during device registration cannot be retrieved.
3. You can [delete a device](#) that is no longer used from the device list. Deleted devices cannot be recovered. Exercise caution when performing this operation.

----End

## Using the Huaweicloud\_iotlink SDK

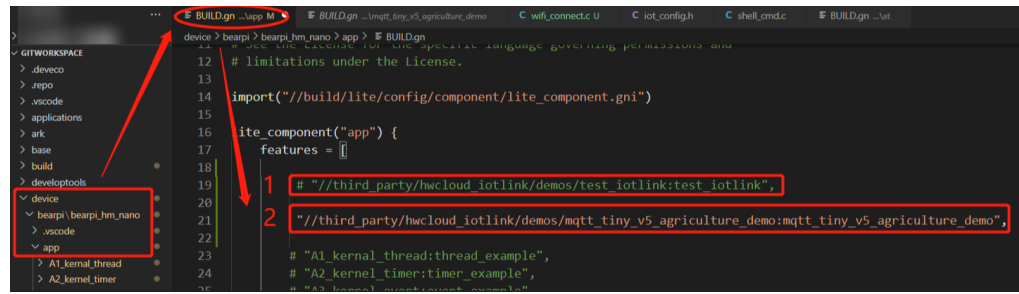
**Step 1** Download the source code [oh3.0\\_hwcloud\\_iotlink-master.zip](#).

**Step 2** Copy the preceding source code to the `src > third_party` directory of the OpenHarmony source code. Note that the third-party libraries of OpenHarmony and Huaweicloud\_iotlink SDK use the OpenHarmony library files, such as cJSON and Mbed TLS.

**Step 3** Add the following code to the OpenHarmony 3.0 source code file `device\bearpi\bearpi_hm_nano\app\BUILD.gn` and select the demo using the comment symbol (#).

```
"/third_party/hwcloud_iotlink/demos/mqtt_tiny_v5_agriculture_demo:mqtt_tiny_v5_agriculture_demo",
```

Figure 2-95 Selecting a demo



**NOTE**

2 in the preceding figure is the demo of connecting an MQTT device to Huawei Cloud. Check the **BUILD.gn** file, as shown in **Figure 2-96**. **A** contains the library files related to the development board hardware and Wi-Fi. **B** contains the library files required for connecting an MQTT device to Huawei Cloud, including cJSON, MQTT- and OSAL-related files, and configuration library files. **C** indicates that the hwcloud\_iotlink library needs to be compiled when the file is compiled. Necessary libraries and C files of the file are located based on the specified path for compilation.

Figure 2-96 Code compilation file

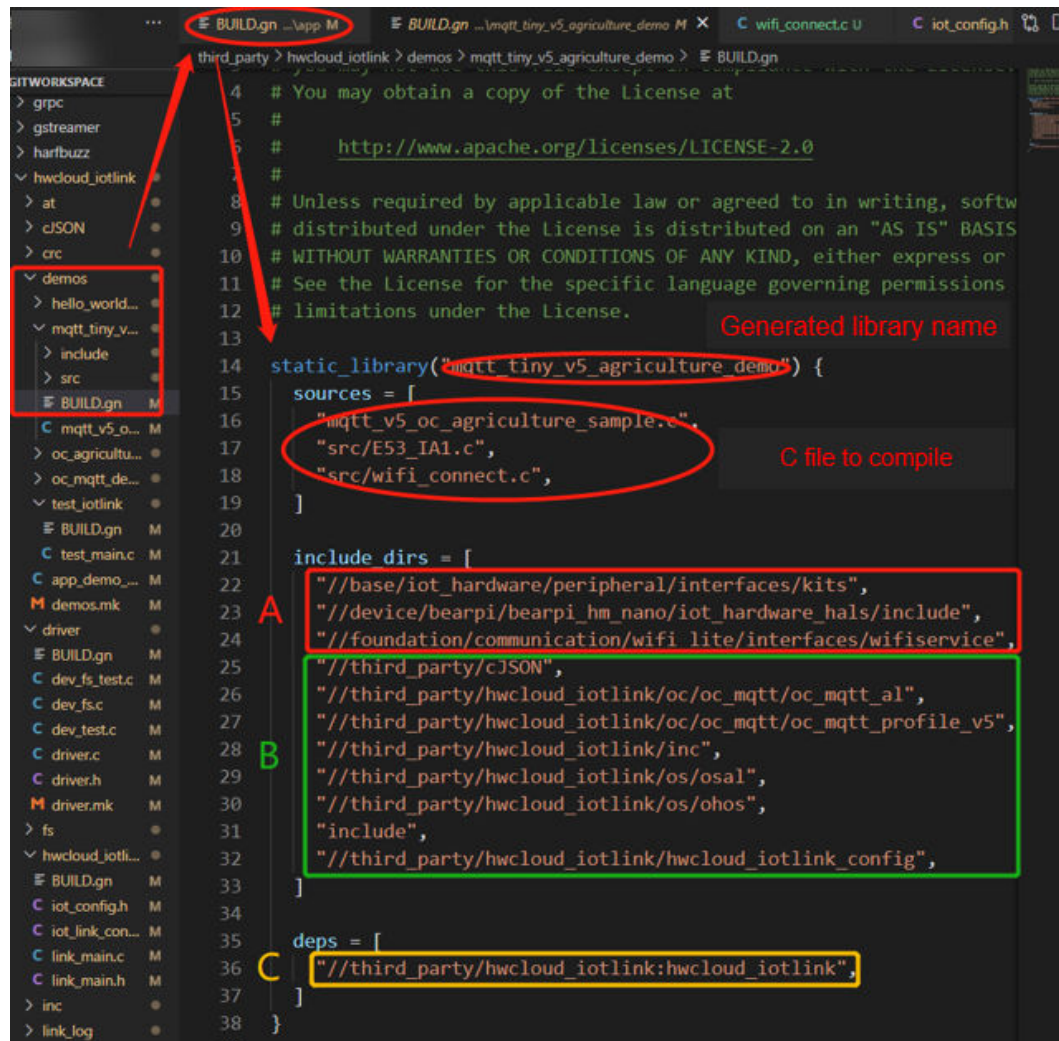


Figure 2-97 Main function file of the demo

```

22
23 #include <dtls_al.h>
24 #include <mqtt_al.h>
25 #include <oc_mqtt_al.h>
26 #include <oc_mqtt_profile.h>
27 #include "E53_IA1.h"
28 #include "wifi_connect.h"
...
356 static void IotMainTaskEntry(void)
357 {
358     link_main_task_entry();
359     (void)osal_task_create("CloudMainTaskEntry", CloudMainTaskEntr
360     (void)osal_task_create("SensorTaskEntry", SensorTaskEntry, NULL
361 }
362
363 APP_FEATURE_INIT IotMainTaskEntry;
    
```

**NOTE**

A in Figure 2-97 shows the library files and DTLS library files required for connecting an MQTT device to Huawei Cloud. The `link_main_task_entry()` function must be called in the entrypoint function `IotMainTaskEntry()` first to ensure the OSAL installation and initialization of other configurations.

**Step 4** Set parameters.

Figure 2-98 Modifying parameters

```

26 #include <oc_mqtt_profile.h>
27 #include "E53_IA1.h"
28 #include "wifi_connect.h"
29 // #include "link_main.h"
30 #define CONFIG_WIFI_SSID ""
31 #define CONFIG_WIFI_PWD ""
32 #define CONFIG_APP_SERVERIP "" access address
33
34 #define CONFIG_APP_SERVERPORT "1883"
35
36 #define CONFIG_APP_DEVICEID ""
37
38 #define CONFIG_APP_DEVICEPWD ""
39
40
41
    
```

**CAUTION**

- (1) To connect to the cloud, you need to modify the network configuration information, Wi-Fi hotspot account and password, and device ID and secret registered on the cloud based on your device. Note that this device supports only 2.4 GHz Wi-Fi.
- (2) Change the interconnection address to the MQTT access address displayed in the **Access Details** dialog box on the **Overview** page of the **console**.
- (3) Compilation and burning can be performed in IDE mode or CLI mode.
- (4) After burning, press the **RESET** button on the development board to start the device.
- (5) The preceding code is based on OpenHarmony 3.0. For other versions, modify the OpenHarmony source code path introduced in the **BUILD.gn** file as required.

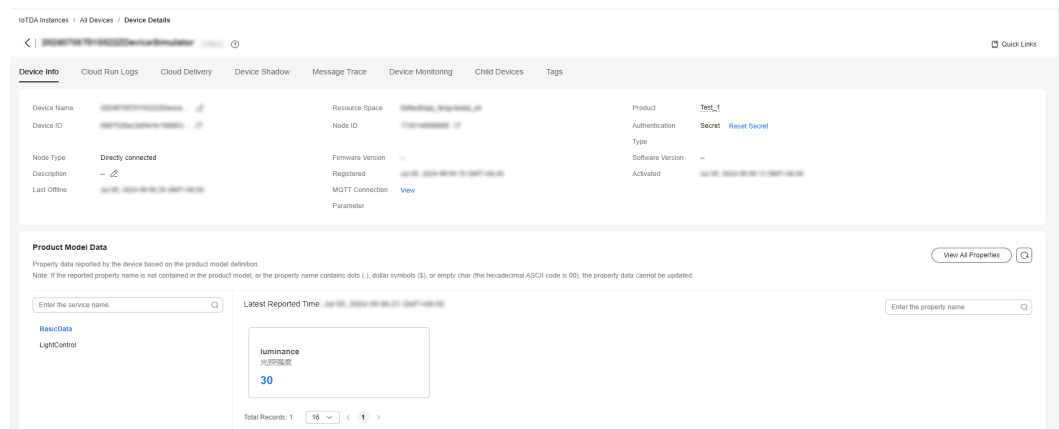
----End

## Connecting the Device to the Platform

After the code is burnt to the device, restart the device. You need to restart the device twice for the first time. During the first burning, you may need to configure the internal information. After the device is restarted twice, it can connect to Huawei Cloud.

On the platform, you can view details about the reported data and deliver commands to control devices, as shown in the following figures.

**Figure 2-99** Viewing reported data - MQTT



**Figure 2-100** Simulating command delivery

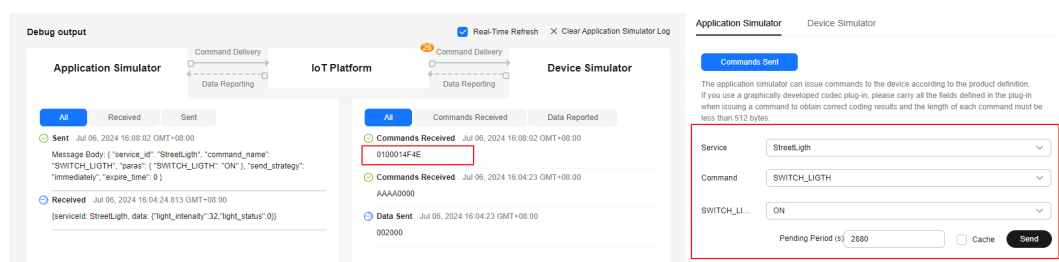


Figure 2-101 Log information

```
clients <I> :
  mac_idx mac      addr      state  lease  tries  rto
  0        201131081a50  [redacted] 10     0      1      4
[DEBUG] [3690] [hub_step] hub_step:enter
[DEBUG] [3690] [dmp_connect] oc_mqtt_connect:server:[redacted] port:[redacted]
[DEBUG] [3700] [dmp_connect] oc_mqtt_connect:client_id:[redacted]
[DEBUG] [3710] [dmp_connect] oc_mqtt_connect:user:[redacted]
passwd:[redacted]
[DEBUG] [3720] [dtls_ssl_new] setting up the SSL structure
[DEBUG] [3730] [dtls_ssl_new] set SSL structure succeed
[DEBUG] [3730] [dtls_shakehand] connecting to server
[DEBUG] [3930] [dtls_shakehand] performing the SSL/TLS handshake
[DEBUG] [4370] [dtls_shakehand] handshake succeed
[DEBUG] [4470] [dmp_connect] oc_mqtt_connect:retcode:0 :SUCCESS
[DEBUG] [4470] [dmp_subscribe] oc_mqtt_subscribe:start
[DEBUG] [4480] [dmp_subscribe] oc_mqtt_default_subscribe:topic:$/oc/devices/[redacted] /sys/commands/#
[DEBUG] [5590] [dmp_subscribe] oc_mqtt_default_subscribe:retcode:0:success
[DEBUG] [5590] [dmp_subscribe] oc_mqtt_subscribe:topic:$/oc/devices/[redacted] /sys/messages/down
[DEBUG] [6680] [dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG] [6680] [dmp_subscribe] oc_mqtt_subscribe:topic:$/oc/devices/[redacted] /sys/properties/set/#
[DEBUG] [7810] [dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG] [7810] [dmp_subscribe] oc_mqtt_subscribe:topic:$/oc/devices/[redacted] /sys/properties/get/#
[DEBUG] [8940] [dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG] [8940] [dmp_subscribe] oc_mqtt_subscribe:topic:$/oc/devices/[redacted] /sys/shadow/get/response/#
[DEBUG] [10070] [dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG] [10070] [dmp_subscribe] oc_mqtt_subscribe:topic:$/oc/devices/[redacted] /sys/events/down
[DEBUG] [11090] [dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG] [11090] [hub_step] hub_step:ok exit
ret = 0
oc_mqtt_profile_connect succeed!
```

## 2.8 Testing MQTT Performance Using JMeter

### Scenarios

The number of global IoT device connections is increasing with the development of IoT technologies. The access and management of devices at scale pose great challenges to the network bandwidth, communications protocols, and platform architecture. It is important to test the platform performance during IoT architecture selection. This section describes how to use JMeter to perform a performance pressure test on the MQTT access capability of the platform.

The test plan is described as follows:

Test scenario:

- Simulate 10,000 concurrently online devices to verify the stability of platform persistent connections.
- Simulate a scenario where devices initiate 100 message reporting requests per second to verify the message processing capability of the platform.

Test environment:

- Test object: Huawei Cloud IoTDA SU2 (10,000 online devices and 100 TPS upstream and downstream messages)
- Test executor: One JMeter executor. The specifications are as follows:

Table 2-11 Test executor

Instance Type	Flavor	Number of vCPUs	Memory
General computing S6	s6.xlarge.2	4 vCPUs	8 GiB

 NOTE

A single JMeter executor can simulate up to 50,000 online devices. To simulate more online devices, use [Huawei Cloud CPTS](#) and deploy multiple JMeter executors.

## Prerequisites

- You have [registered a Huawei ID](#) and enabled Huawei Cloud services.
- You have subscribed to IoTDA. If not, access [IoTDA](#) and buy an SU2 unit (10,000 online devices and 100 TPS upstream and downstream messages).

## Preparations

- Install the Java runtime environment on the JMeter executor. Visit the [Java website](#), and download and install the Java runtime environment.
- [Download](#) and install JMeter 5.1.1 or later.
- [Download](#) the `mqtt-jmeter` plug-in package and store it in the `lib/ext` directory of the JMeter installation directory.

## Service Flow

The process of using JMeter to perform an MQTT performance pressure test on the platform is as follows:

- Step 1** [Create an MQTT product](#).
- Step 2** [Register 10,000 devices](#) in batch import mode.
- Step 3** [Import the test plan](#) created for the IoT performance test.
- Step 4** [Initiate a platform performance pressure test](#) based on service specifications.
- Step 5** [View the test result](#). Check whether the test result meets the expectation based on the monitoring metrics displayed on the IoT platform.

----End

## Creating a Product

- Step 1** Log in to the [console](#), choose **Products** in the navigation pane, and click **Create Product** on the left.
- Step 2** Set the parameters as prompted and click **OK**.

**Table 2-12** Parameters

Basic Information	
Resource Space	The platform automatically allocates the created product to the default resource space. You can also select an existing resource space from the drop-down list or <a href="#">create one</a> .
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).

Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Industry	Set this parameter as required.
Device Type	Set this parameter as required.
<b>Advanced Settings</b>	
Product ID	Set a unique identifier for the product. If this parameter is specified, the platform uses the specified product ID. If this parameter is not specified, the platform allocates a product ID.

----End

## Registering a Batch of Devices

- Step 1** Visit the [IoTDA](#) service page and click **Access Console**.
- Step 2** In the navigation pane, choose **Devices > All Devices**, click the **Batch Registration** tab, and then click **Batch Register**.
- Step 3** Download and fill in the batch device registration template based on the following table. You can download the [sample file](#).

**Table 2-13** Parameters

Parameter	Description
node_id	Device identifier. Set this parameter in ascending order, for example, <b>10001</b> , <b>10002</b> , and <b>10003</b> .
product_id	Product ID generated in <a href="#">Creating a Product</a> .
app_id	Resource space. For details about how to obtain the resource space, see <a href="#">Resource Spaces</a> .
device_name	Device name, which can be the same as the value of <b>node_id</b> .
secret	Device secret. You can enter a fixed secret during a performance test.

- Step 4** In the **Batch Registration** dialog box, click **Select File** to upload the prepared batch registration template, and click **OK**.



**Figure 2-102** Device - Registering devices in batches

**Batch Registration** ✕

★ Task Name

★ File

Download the template, enter the content in text format, and upload the file.

[↓ Batch Device Registration Template](#)

**Step 5** After the batch registration is successful, save the device IDs and secrets.

----End

## Importing a Test Plan

**Step 1** **Download** the JMeter test plan.

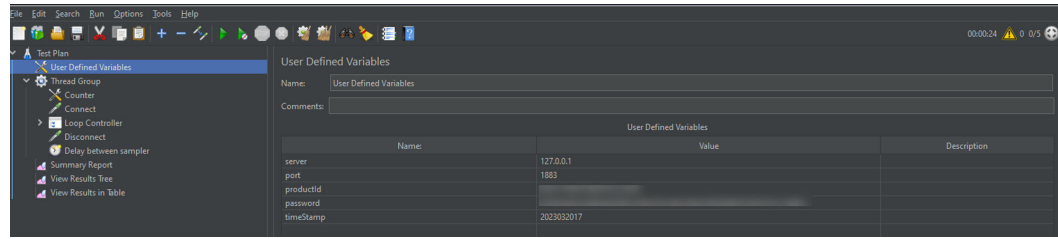
**Step 2** Open JMeter and click **Open** to import the downloaded test plan.

**Step 3** In the JMeter directory on the left, choose **User Defined Variables**. On the **User Defined Variables** page, configure the following parameters:

**Table 2-14** Parameters

Parameter	Description
server	MQTT server address. For details about how to obtain the value, see <a href="#">Obtaining Resources</a> .
port	MQTT port number. Set this parameter to <b>8883</b> .
productId	Product ID generated in <a href="#">Creating a Product</a> .
password	MQTT connection password, which is the value of the device secret encrypted by using the HMAC-SHA256 algorithm with the timestamp as the key. <b>secret</b> is the secret entered during <a href="#">batch device registration</a> . You can use this <a href="#">tool</a> to obtain the encrypted value.
timeStamp	Timestamp used for encrypting the password. The time format is YYYYMMDDHH.

Figure 2-103 Example

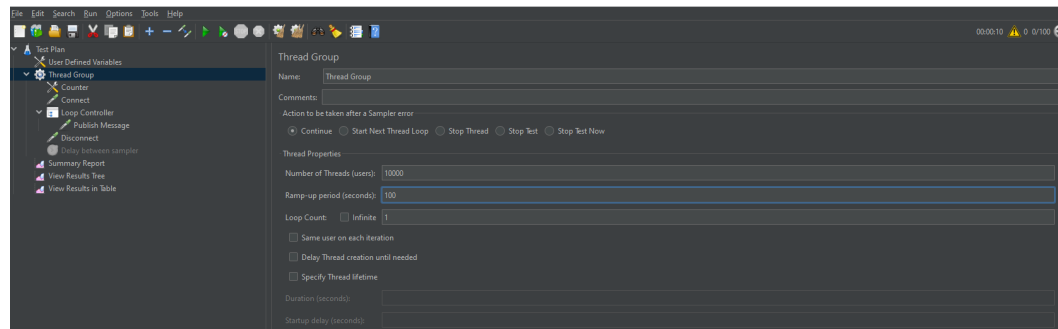


----End

## Initiating a Pressure Test

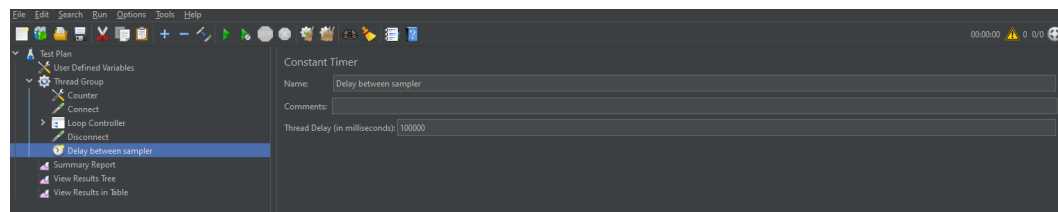
- Step 1** In the JMeter directory on the left, choose **Thread Group**, set **Number of Threads** to **10000**. (A thread corresponds to an online device. **10000** indicates that 10,000 devices are online on the IoT platform.)

Figure 2-104 Configuring devices



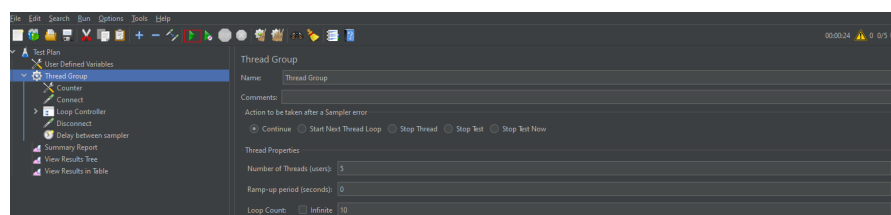
- Step 2** In the JMeter directory on the left, choose **Thread Group > Delay between sampler**. Set **Thread Delay (in milliseconds)** to **100000** (indicating that each device publishes a message every 100 seconds).

Figure 2-105 Configuring devices



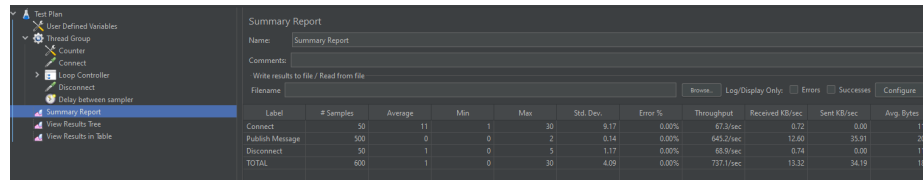
- Step 3** Click  on the JMeter toolbar to start the performance test.

Figure 2-106 Performance test



**Step 4** In the JMeter directory on the left, choose **Summary Report**. The throughputs of **Connect** and **Publish Message** are displayed. You can modify the values of **Number of Threads** and **Thread Delay (in milliseconds)** to adjust the throughputs.

**Figure 2-107** Performance test



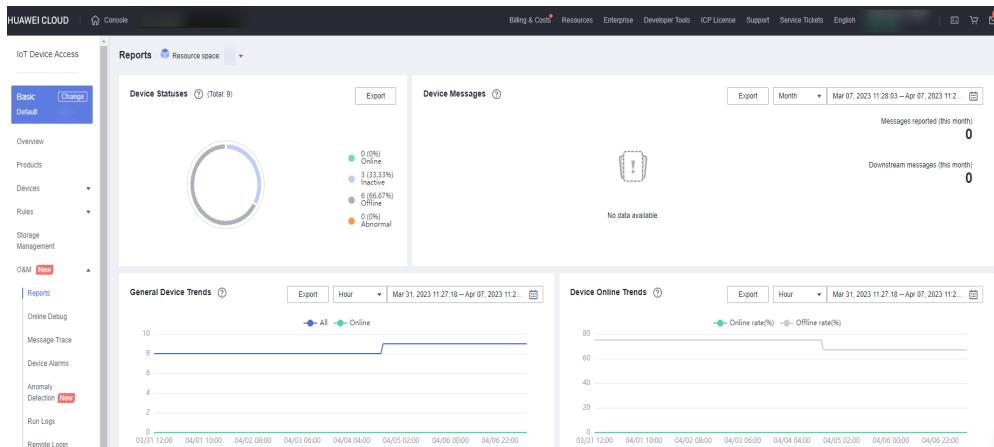
**Step 5** After the JMeter test plan is debugged, import the test plan to **CodeArts PerfTest** for distributed deployment to meet requirements of higher-level performance tests.

----End

## Viewing the Pressure Test Result

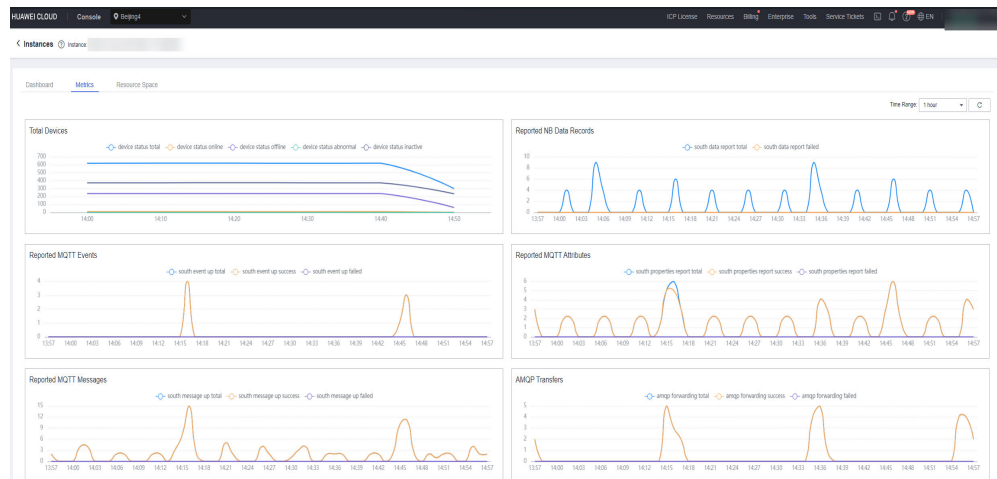
1. Log in to the **IoTDA console** and choose **O&M > Reports** in the navigation pane to view statistical metrics of the platform.

**Figure 2-108** Viewing statistical reports



2. For more reports, log in to the **AOM console**, choose **Monitoring > Cloud Service Monitoring**, and select **IoT Device Access (IoTDA)**.

Figure 2-109 Viewing metrics



# 3 Device Management

## 3.1 Automatically Adjusting Air Conditioner Temperature Through Device Shadow

### Scenarios

Using a constant-temperature control system, you can adjust the default temperature of air conditioners (regardless of whether they are powered on). After being powered on, the air conditioners automatically run at the default temperature. This is achieved by using the device shadow of the IoT platform. For any connected air conditioner, you can set the device shadow on the application or IoTDA console to deliver the preset temperature to the air conditioner. The air conditioner automatically adjusts the temperature after receiving the property modification request.

### Developing a Product

- Step 1** Visit the [IoTDA](#) service page and click **Access Console**. Click the target instance card.
- Step 2** In the navigation pane, choose **Products**. In the search box, select the resource space to which the new product belongs.
- Step 3** Click **Create Product** on the left to create a constant-temperature air conditioner product, set the parameters, and click **OK**.

Basic Information	
Product Name	Enter a value, for example, <b>aircondition</b> .
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Industry	Customize the values.

Device Type	
-------------	--

**Step 4** After the product is created, click the corresponding product to access its details.

**Step 5** On the **Basic Information** tab page, click **Customize Model** and configure the product model based on the table below.

Service data	
Service	<p><b>Service ID:</b> Enter <b>temperature</b>.</p> <p><b>Service Type:</b> You are advised to set this parameter to the same value as <b>Service ID</b>.</p>
Property	<p><b>Property Name:</b> Enter <b>temperature</b>.</p> <p><b>Data Type:</b> Select <b>Integer</b>.</p> <p><b>Access Permissions:</b> Select <b>Read</b> and <b>Write</b>.</p> <p><b>Length:</b> Enter <b>1</b>.</p>

**Step 6** In the navigation pane, choose **Devices > All Devices**, and click **Register Device**. In the dialog box displayed, set the parameters based on the table below.

**Figure 3-1** Registering a device - MQTT

Parameter	Description
Product	Select the product created in <a href="#">Step 3</a> .
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

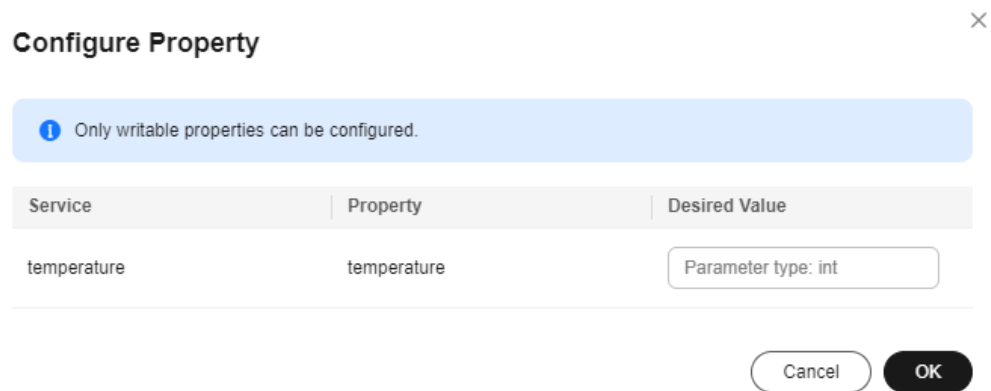
----End

## Configuring a Device Shadow

You can set a device shadow on the IoTDA console or by calling the API for [configuring desired properties in a device shadow](#) on the application side. This section describes how to set a device shadow on the IoTDA console.

- Step 1** Log in to the IoTDA [console](#), choose **Devices > All Devices** in the navigation pane, and click **View** in the row of the device registered in [Step 6](#) to access its details.
- Step 2** On the **Device Shadow** tab page, click **Configure Property**.
- Step 3** In the dialog box displayed, enter the desired value of a property. In this example, the value of **temperature** is set to **25**.

**Figure 3-2** Configuring a device shadow



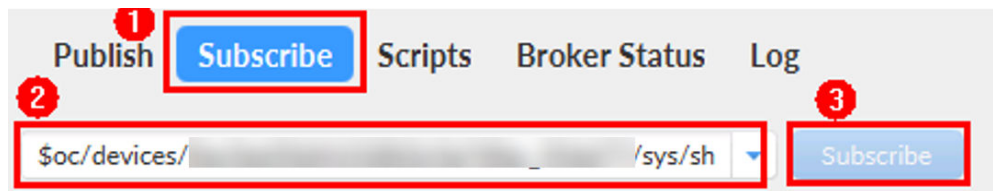
----End

## Verifying Configurations

Method 1:

You can use MQTT.fx to simulate device verification.

1. Use MQTT.fx to simulate a constant-temperature air conditioner and connect it to the platform. For details, see [Developing an MQTT-based Simulated Smart Street Light Online](#).
2. On the **Subscribe** tab page, enter **topic=\$oc/devices/{device\_id}/sys/shadow/get/response/#** of the device shadow (replace *{device\_id}* with the device ID in [Step 6](#)), and click **Subscribe**.

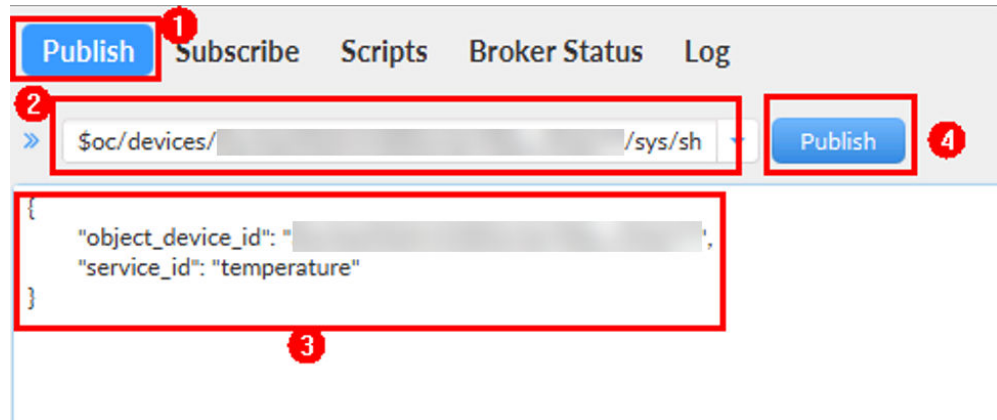


3. On the **Publish** tab page, enter **Topic=\$oc/devices/{device\_id}/sys/shadow/get/request\_id={request\_id}** of the device shadow.
4. Enter a request for obtaining the device shadow and click **Publish**.

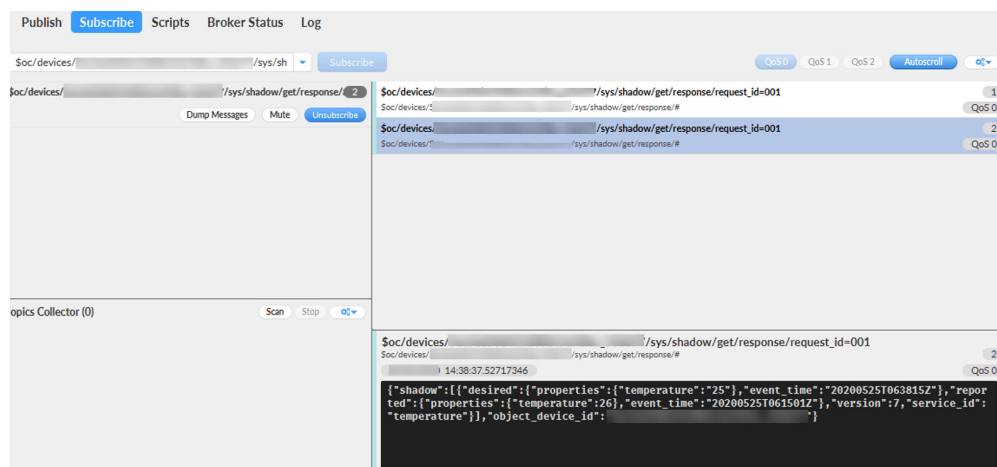
Example:

```
{
  "object_device_id": "*****",
  "service_id": "temperature"
}
```





- Click the **Subscribe** tab. The device shadow data delivered by the platform is displayed.



**Method 2:**

You can connect a physical device to the platform. The device will receive the device shadow configuration delivered by the platform and change the preset temperature accordingly.

## 3.2 Managing Indoor Air Conditioners Using Custom Topics

### Scenarios

Custom topics apply to MQTT devices connected to IoTDA. Topics for **message reporting** and **message delivery** can be customized. Applications can implement different service logic processing based on topics. Custom topics can also be used in the scenario where a device cannot report properties or receive delivered commands defined in the product model.

In this example, an application receives data reported by a device and determines whether to turn on or off the indoor air conditioner.

## Prerequisites

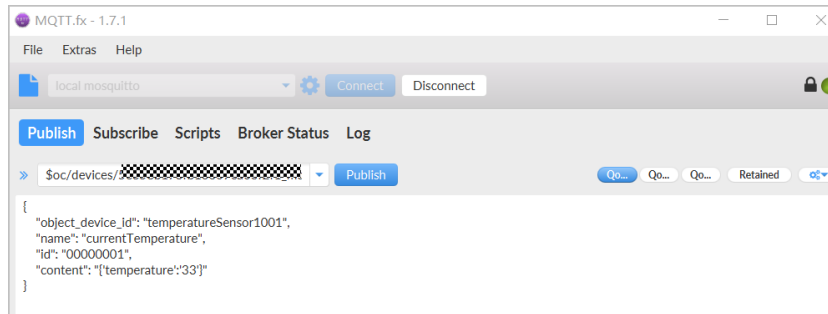
- You have [registered a Huawei ID](#) and enabled Huawei Cloud services.
- You have subscribed to IoTDA.
- You have created an MQTT product, developed a product model, and registered a device on the IoTDA [console](#). For details, see [Products](#), [Developing a Product Model Online](#), [Registering an Individual Device](#), or [Registering a Batch of Devices](#).
- The connection between the device and platform has been established. For details, see [Device Connection Authentication](#).

## Customizing a Topic

For details, see [Custom Topic Communications](#).

## Message Reporting

1. Visit the [IoTDA](#) service page and click **Access Console**. Click the target instance card.
2. In the navigation pane, choose **Devices > All Devices**, locate the target device, and click **View** to access its details page.
3. Click the **Message Trace** tab, click **Start Trace**, and set the trace duration as required.
4. Use the MQTT.fx simulator to simulate a device to report a custom topic message. For details, see [Quick Device Access](#).

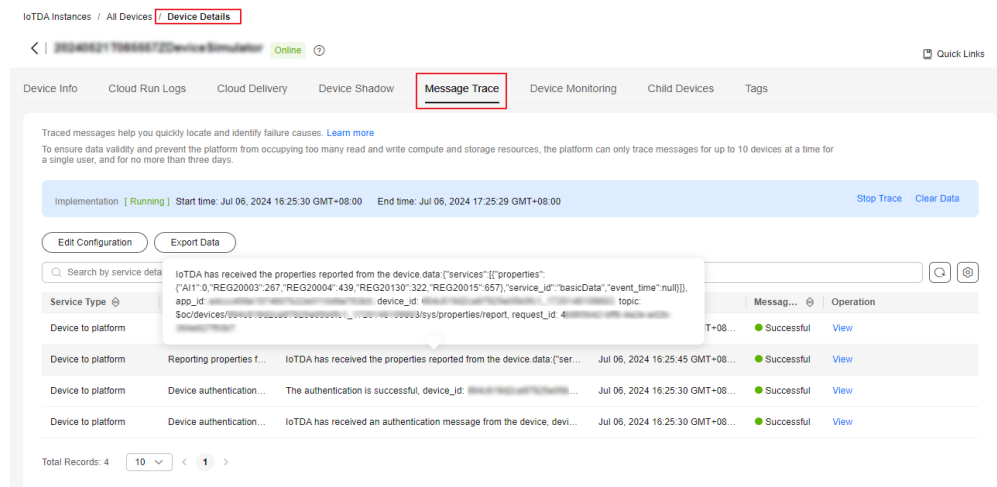


### NOTE

For devices connected using the IoT Device SDK or native MQTT protocol, you need to set the custom topic name reported by the device in the device program.

5. On the **Message Trace** page, view the custom topic messages reported by the device.

**Figure 3-3** Viewing message trace

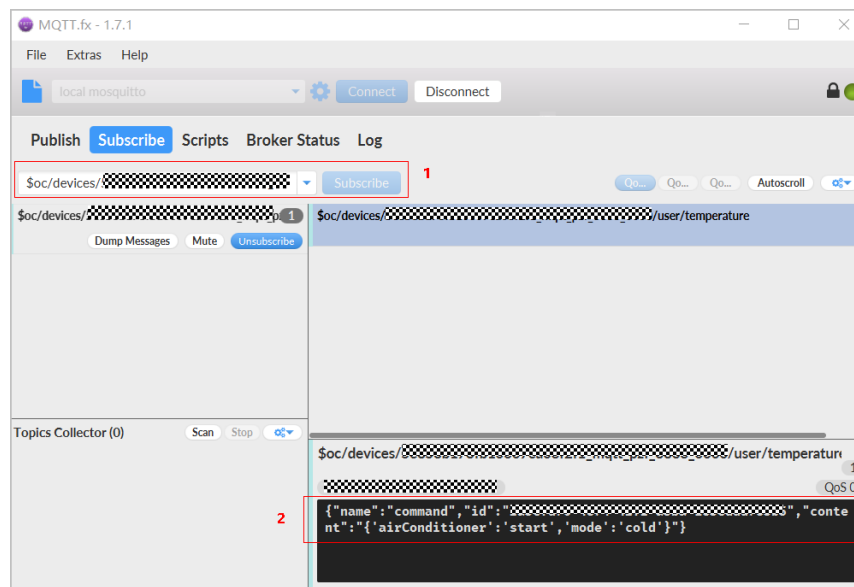


6. An application obtains the custom topic message reported by the device through data forwarding. For details, see [Data Forwarding](#). You can also refer to [Forwarding Device Data to OBS for Long-Term Storage](#).

## Message Delivery

In this example, the Postman is used to deliver an instruction for starting the indoor air conditioner.

1. Use the MQTT.fx simulator to subscribe to a custom topic.



### NOTE

- Ensure that the device operation permissions contain the subscription function when you create a custom topic. For details, see [Custom Topic Communications](#).
- For devices connected using the IoT Device SDK or native MQTT protocol, you need to set the name of the custom topic to which the device subscribes in the device program.



to be upgraded due to security vulnerabilities, software bugs, function optimization, and device performance improvement. This section describes how to upgrade the firmware in Huawei Cloud IoTDA using MQTT.fx to simulate a device.

**NOTE**

The software upgrade process is the same as that of the firmware upgrade. The only difference is that the parameters for reporting version numbers are different. For firmware upgrade, the parameter that specifies the version number is **fw\_version**. For software upgrade, the parameter is **sw\_version**. For details, see [Device Reporting the Software and Firmware Versions](#).

## Development Process

Figure 3-4 Upgrade process

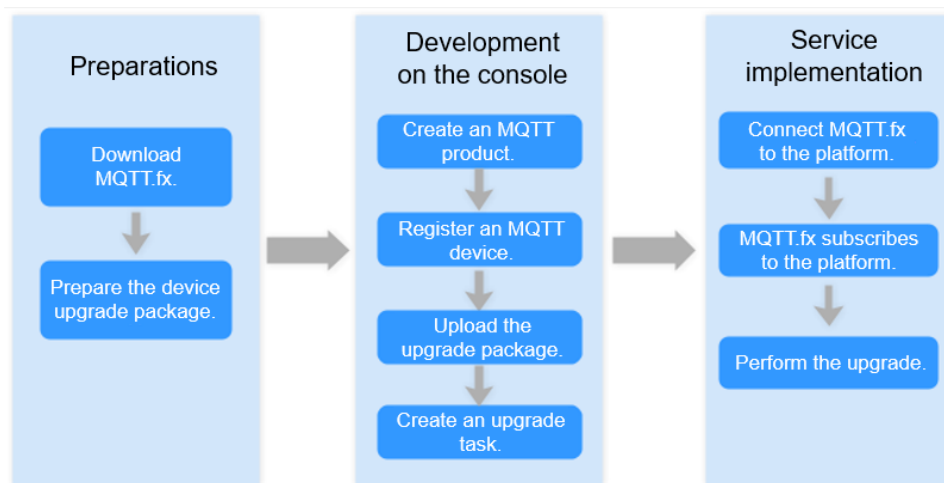
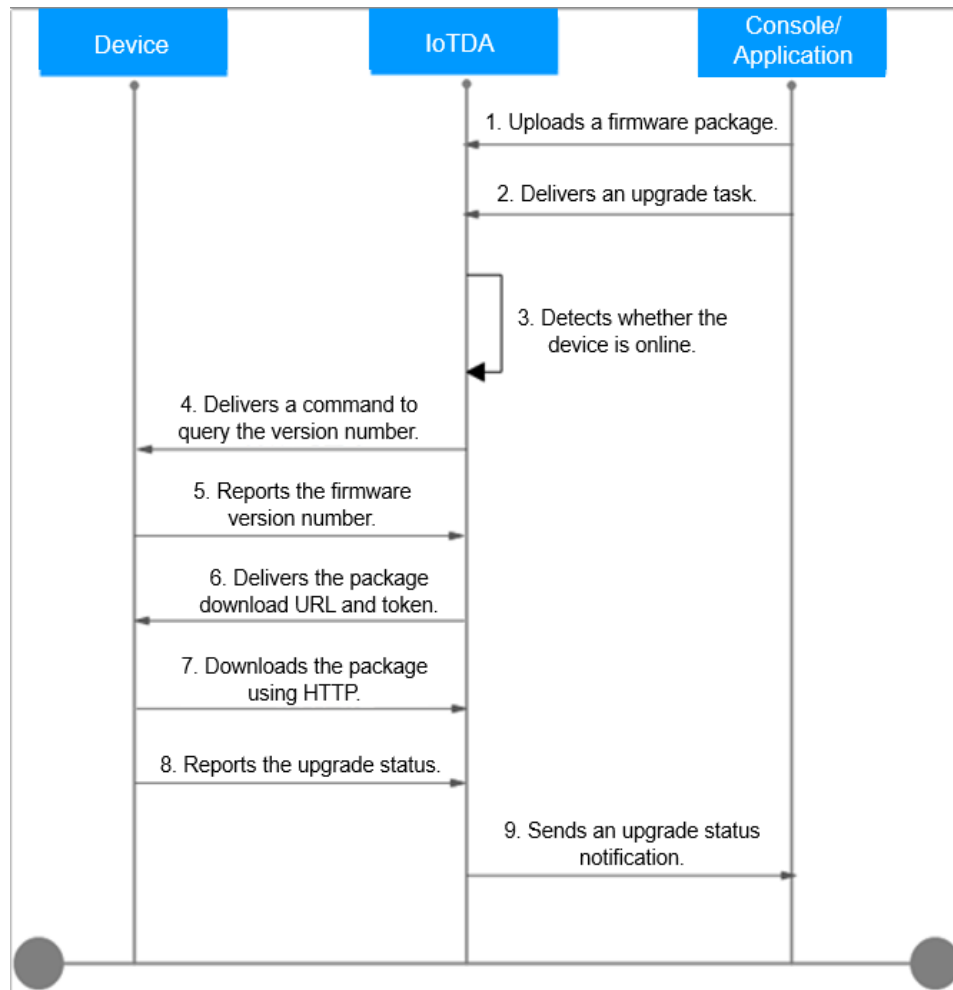


Figure 3-5 MQTT device upgrade process



## Development Environment

- Software: Huawei Cloud **IoTDA**, 64-bit Windows 7 or later (64-bit Windows 10 is used in the following demonstration), and **MQTT.fx** simulator.

## Prerequisites

- You have **registered a Huawei Cloud account**.
- You have completed real-name authentication. Otherwise, IoTDA cannot be used.
- You have subscribed to IoTDA. If you have not subscribed to the service, go to the **IoTDA** service page, and click **Access Console** to subscribe to the service.

## Preparations

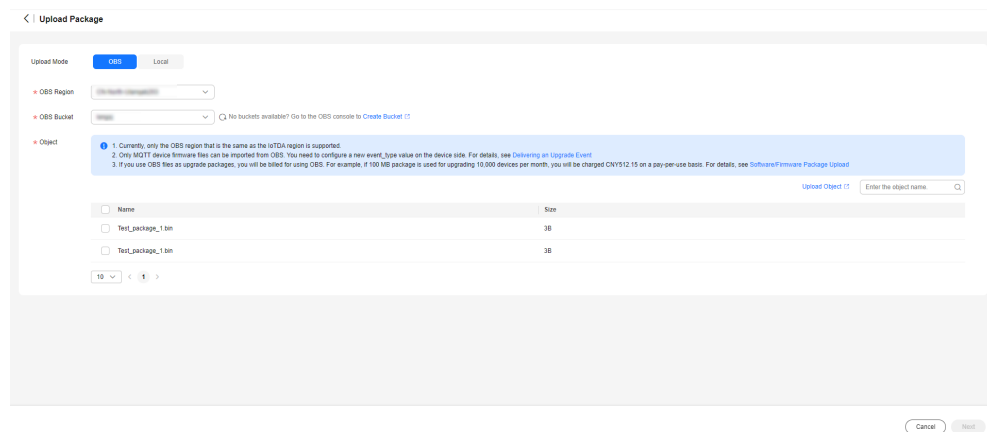
- 1. Download and install MQTT.fx (version 1.7.1 or later).
- Prepare the upgrade package. Obtain the firmware upgrade package and its version number from the module vendor. Temporary files are used in this demonstration.
- **Create an MQTT product**. (If an MQTT product already exists, skip this step.)

- **Register an individual device.**
- Add the registered device to a **device group**, which will be used during upgrade task creation.

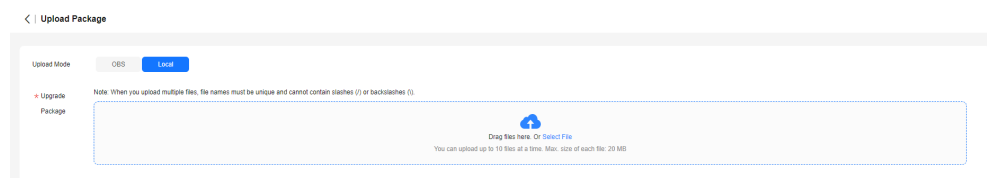
## Uploading an Upgrade Package

- Step 1** Visit the **IoTDA** service page and click **Access Console**. Click the target instance card.
- Step 2** In the navigation pane, choose **Devices > Software/Firmware Upgrades**.
- Step 3** Click the **Manage Resource Package** tab and click **Firmware List**.
- Step 4** Click **Upload**. On the displayed page, upload a firmware package from OBS or your local PC.

**Figure 3-6** Uploading the upgrade package - OBS file

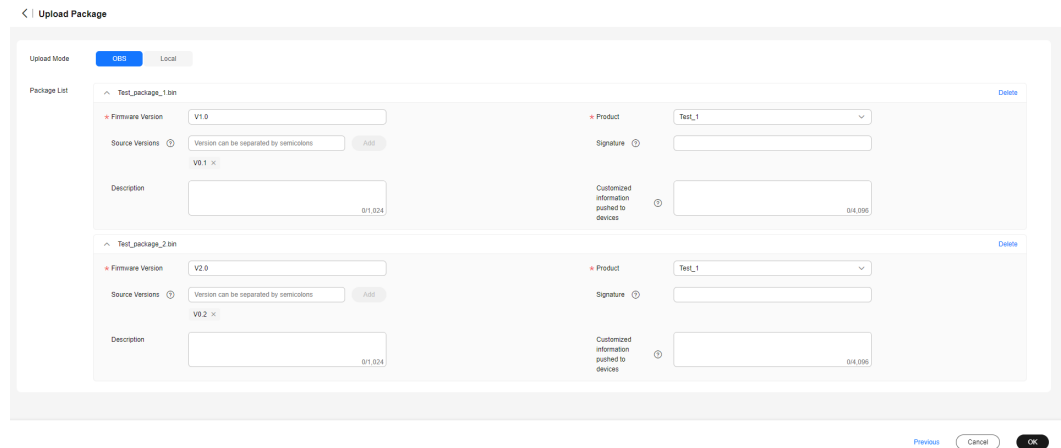


**Figure 3-7** Uploading the upgrade package - local file

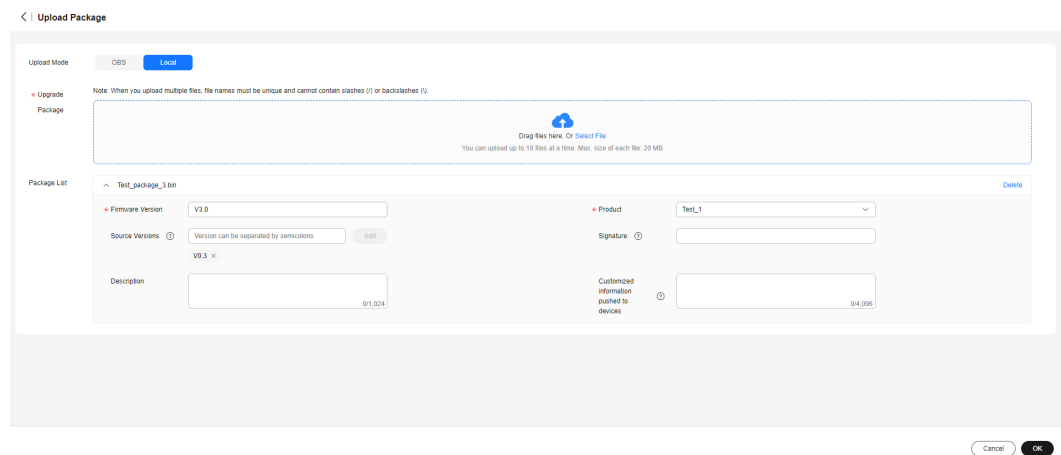


- Step 5** After the upgrade package is uploaded, configure package parameters and click **OK**.

**Figure 3-8** Uploading the upgrade package - OBS file parameters

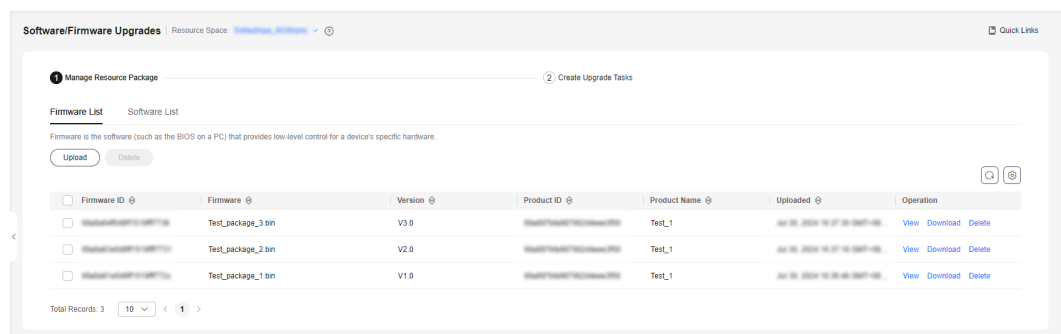


**Figure 3-9** Uploading the upgrade package - local file parameters



**Step 6** The uploaded upgrade package is displayed in the firmware list.

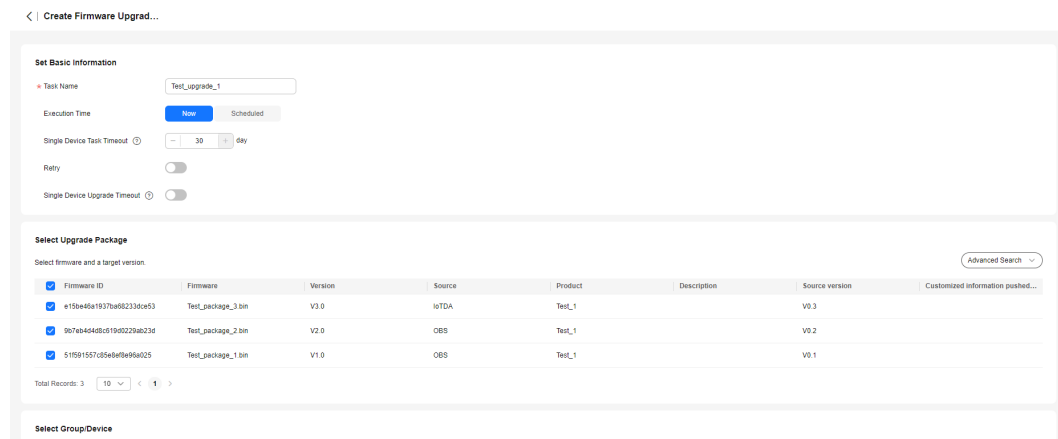
**Figure 3-10** Software/Firmware upgrade - Firmware list



**Step 7** Click **Create Upgrade Task**. On the **Firmware Upgrades** tab page, click **Create Task**. On the displayed page, configure parameters, select upgrade packages, select devices to upgrade, and click **Create Now**.



**Figure 3-11** Creating a firmware upgrade task - Test\_upgrade\_1

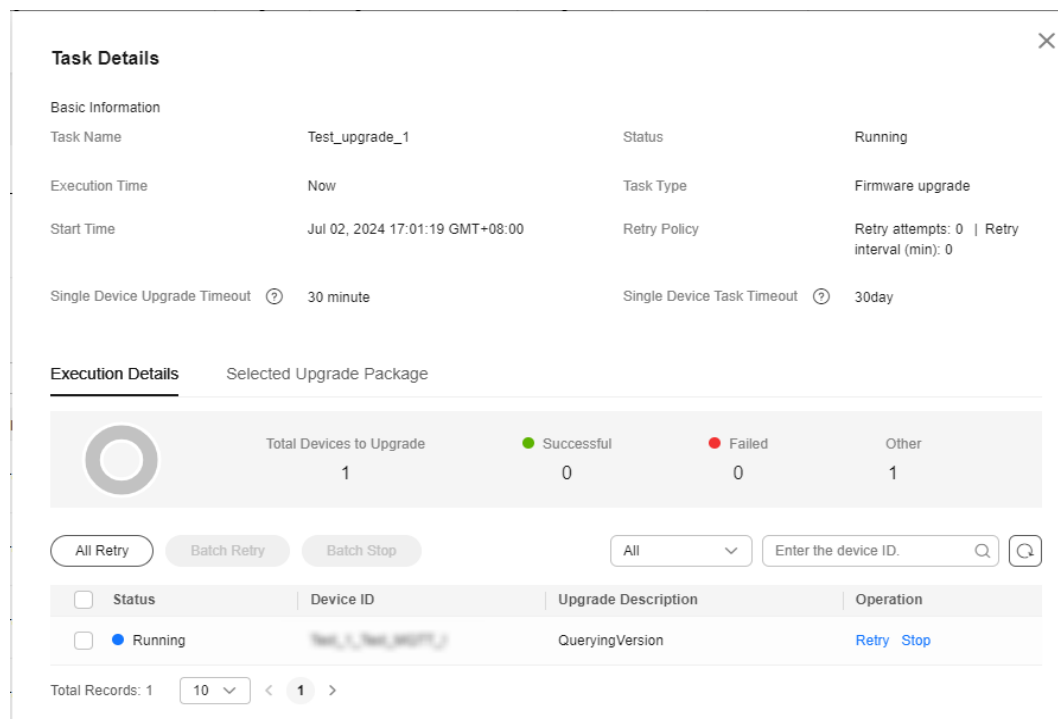


**NOTE**

When creating an upgrade task, you can select up to 10 upgrade packages. Supported source versions of the upgrade packages must be unique. If no source version is specified for an upgrade package, the package can be used to upgrade devices of all source versions by default.

- Step 8** View the created task in the task list. You can click **View** to view the task details. On the task details page, you can stop executing upgrade tasks for up to 100 devices in batches or an upgrade task for a single device. You can retry failed upgrade tasks for up to 100 devices in batches or an upgrade task for a single device. You can click **All Retry** to retry the upgrade task for all failed devices.

**Figure 3-12** Task details - Test\_upgrade\_1



----End

## Service Implementation

- Step 1** Use MQTT.fx to simulate device access to the platform. For details, see "Performing Connection Authentication".
- Step 2** Use MQTT.fx to subscribe to the downstream message topic of the platform. MQTT.fx receives the version query command delivered by the platform.

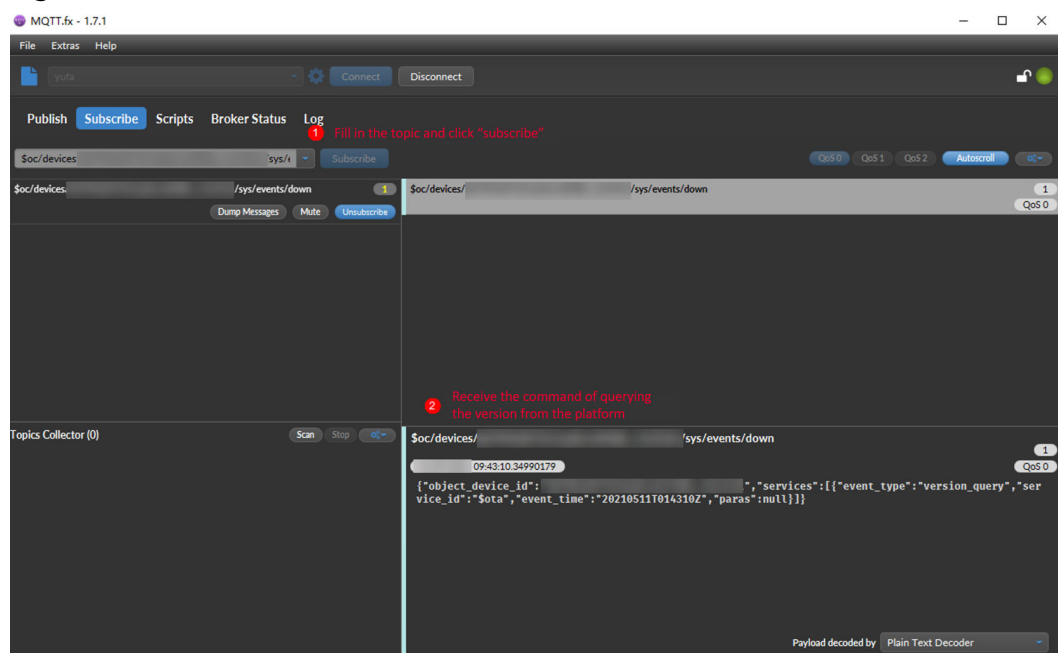
### Topic

Downstream: **`$oc/devices/{device_id}/sys/events/down`**

### Parameters

For details, see [Platform Delivering an Event to Obtain Version Information](#).

**Figure 3-13** Subscribe/Push



- Step 3** Report the software and firmware version information using MQTT.fx.

### Topic

Upstream: **`$oc/devices/{device_id}/sys/events/up`**

### Parameters

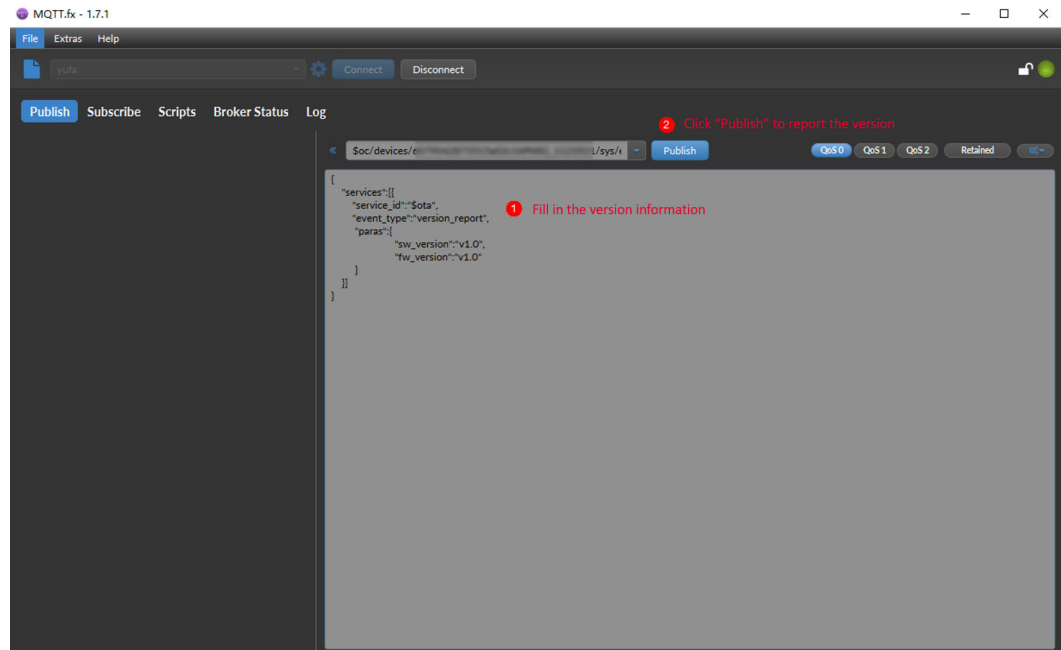
For details, see [Device Reporting the Software and Firmware Versions](#).

### Example

```
Topic: $oc/devices/{device_id}/sys/events/up  
Data format: {  
  "object_device_id": "{object_device_id}",  
  "services": [{  
    "service_id": "$ota",  
    "event_type": "version_report",  
    "event_time": "20151212T121212Z",  
    "paras": {  
      "sw_version": "v1.0",  
      "fw_version": "v1.0"  
    }  
  }  
}
```

```
}  
  }  
}
```

**Figure 3-14** Reporting version numbers



**Step 4** After the version numbers are reported, the simulator receives an upgrade notification from the platform. The notification is as follows:

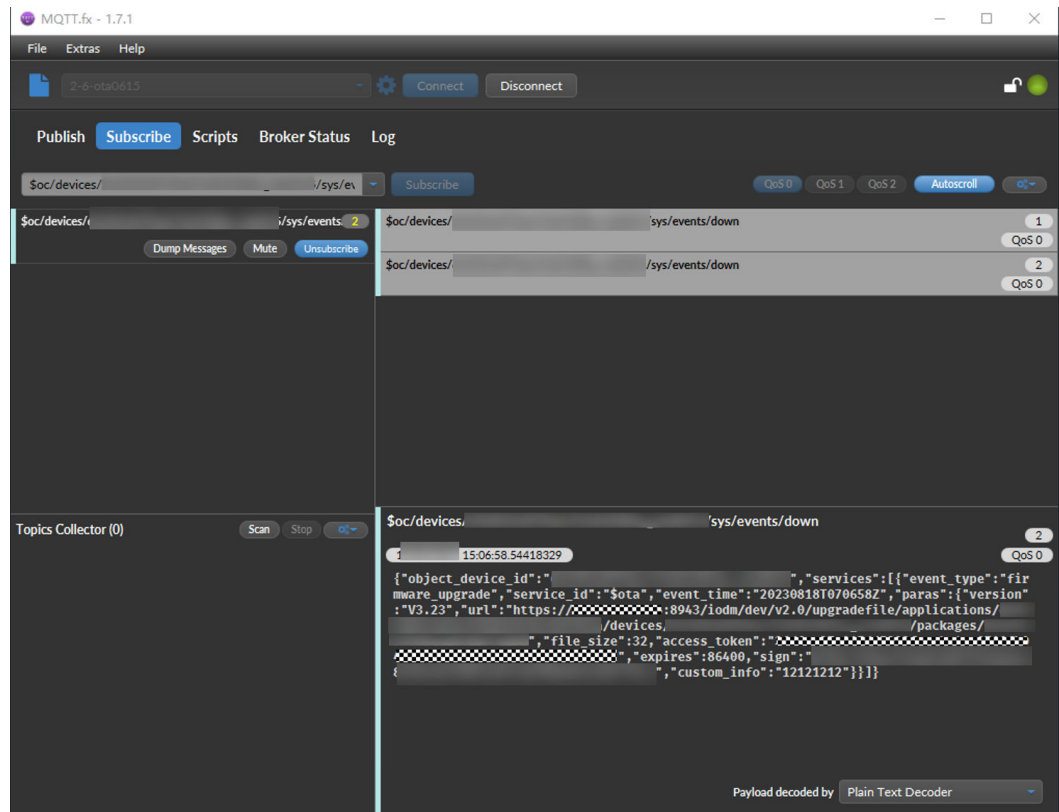
**Topic**

Upstream: **`$oc/devices/{device_id}/sys/events/down`**

**Parameters**

For details, see [Platform Delivering an Upgrade Event](#).

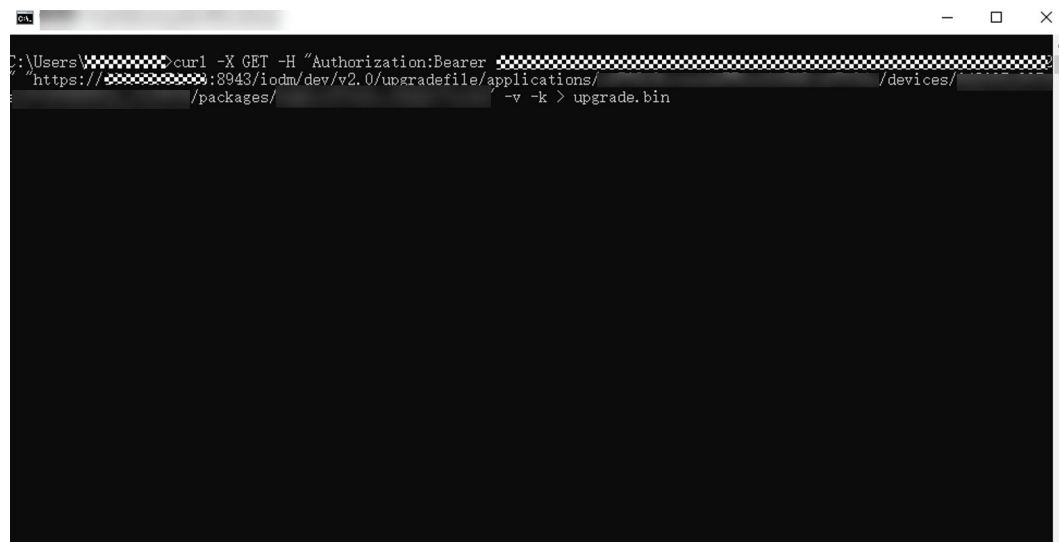
**Figure 3-15** Obtaining the upgrade notification



**Step 5** After the device receives the upgrade notification, send an HTTP request to download the upgrade package.

A cURL command is used in this demonstration.

**Figure 3-16** Downloading the upgrade package



**Example**

```

curl -X GET -H "Authorization:Bearer *****" "https://**.*.*:8943/iodm/dev/v2.0/upgradefile/applications/*****/devices/*****/packages/*****" -v -k
    
```

**CAUTION**

- Add **Authorization** to the HTTP request header. The value of **Authorization** is **Bearer** *{access\_token}*, where the value of *{access\_token}* is the token in the upgrade notification. Leave a space between **Bearer** and *{access\_token}*.
- If **event\_type** is set to **firmware\_upgrade\_v2** or **software\_upgrade\_v2**, the request header does not need to be carried in the request for downloading the software or firmware package. An example request is as follows:

```
GET https://*****.obs.cn-north-4.myhuaweicloud.com:443/test.bin?  
AccessKeyId=DX5G7W*****
```

**Step 6** Enable the device to report the upgrade status.

**Topic**

Upstream: **Soc/devices/{device\_id}/sys/events/up**

**Parameters**

For details, see [Device Reporting the Upgrade Status](#).

**Example**

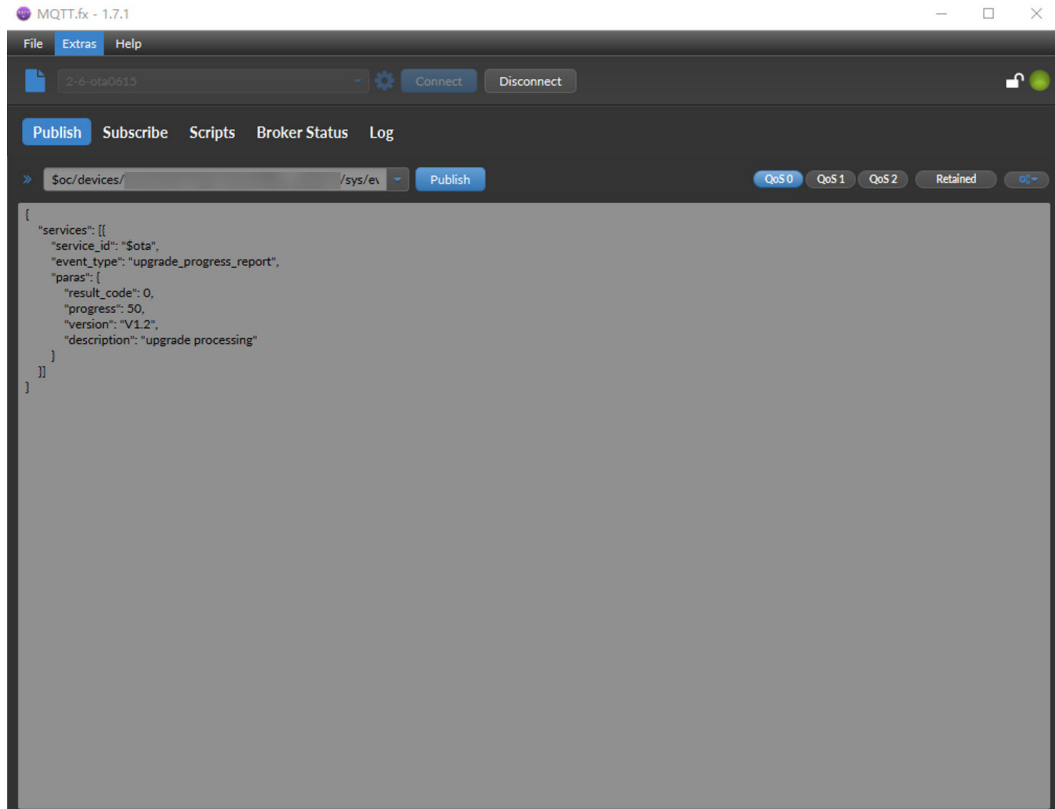
Topic: Soc/devices/{device\_id}/sys/events/up

Data format:

```
{  "object_device_id": "{object_device_id}",  
  "services": [{  
    "service_id": "$ota",  
    "event_type": "upgrade_progress_report",  
    "event_time": "20151212T121212Z",  
    "paras": {  
      "result_code": 0,  
      "progress": 50,  
      "version": "V1.0",  
      "description": "upgrade processing"  
    }  
  }  
}] }
```

The following figure shows that the upgrade progress is 50%, which is displayed on the platform.

**Figure 3-17** Reporting the upgrade progress (50%)



**Figure 3-18** Upgrade progress - Test\_upgrade\_1-

### Task Details ✕

Basic Information

Task Name	Test_upgrade_1	Status	Running
Execution Time	Now	Task Type	Firmware upgrade
Start Time	Jul 02, 2024 17:01:19 GMT+08:00	Retry Policy	Retry attempts: 0   Retry interval (min): 0
Single Device Upgrade Timeout <span>?</span>	30 minute	Single Device Task Timeout <span>?</span>	30day

Execution Details Selected Upgrade Package

Total Devices to Upgrade

1

● Successful

0

● Failed

0

Other

1

All Retry
Batch Retry
Batch Stop
All v
Enter the device ID. Q Q

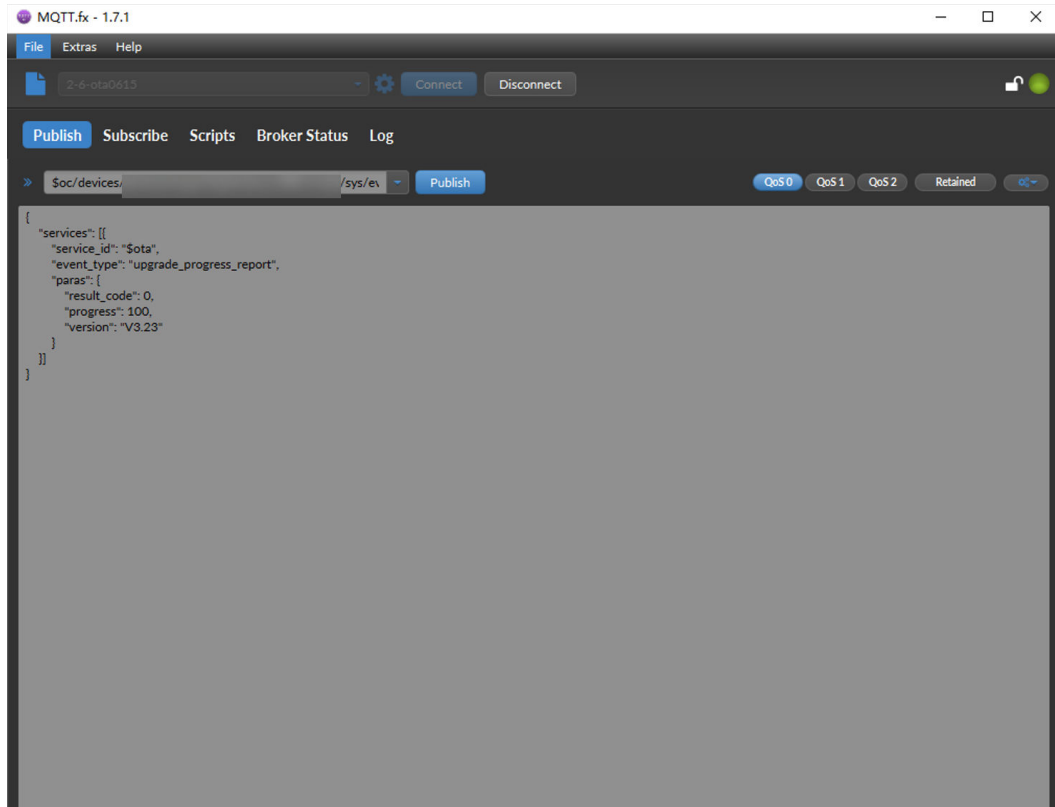
<input type="checkbox"/> Status	Device ID	Upgrade Description	Operation
<input checked="" type="checkbox"/> Running	...	downloading progress is 50%	<a href="#">Retry</a> <a href="#">Stop</a>

Total Records: 1 10 v < 1 >

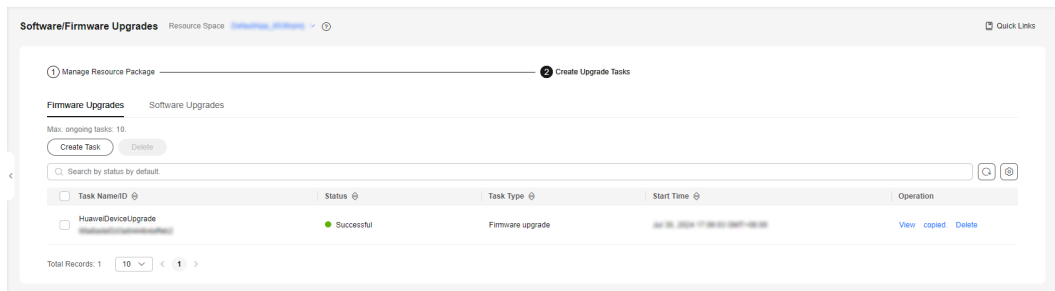
**Step 7** Complete the upgrade.

If the upgrade progress is 100% and the current version is the target version, the upgrade task success will be displayed on the platform.

**Figure 3-19** Reporting the upgrade progress (100%)



**Figure 3-20** Software/Firmware upgrade - Firmware upgrade



-----End

# 4 Data Forwarding

---

## 4.1 Pushing Metric Data to DMS for Kafka

### Introduction

IoTDA provides various data reports, which you can integrate into your O&M system. You can use the subscription function of Application Operations Management (AOM) to push metric data from IoTDA to DMS for Kafka, and then consume and display the data.

### Prerequisites

- You have [registered a Huawei account](#).
- You have subscribed to IoTDA. Method: Visit the [IoTDA](#) service page, and click the button for free trial or the price calculator to purchase and subscribe to the service.
- You have subscribed to AOM. Method: Visit the [AOM](#) service page, click **AOM 2.0 Console**, and click the button to enable the service for free and authorize.
- You have subscribed to DMS for Kafka. Method: Visit the [DMS for Kafka](#) service page, and click **Buy Now**.

### Example Scenario

In this example, the AOM subscription function is used to push metric data to DMS for Kafka. The number of online devices is used as an example. For details, see [IoTDA Metrics](#).

The procedure is as follows:

1. [Create a subscription](#) on the AOM console.
2. [Simulate device going online and verifying the result](#).

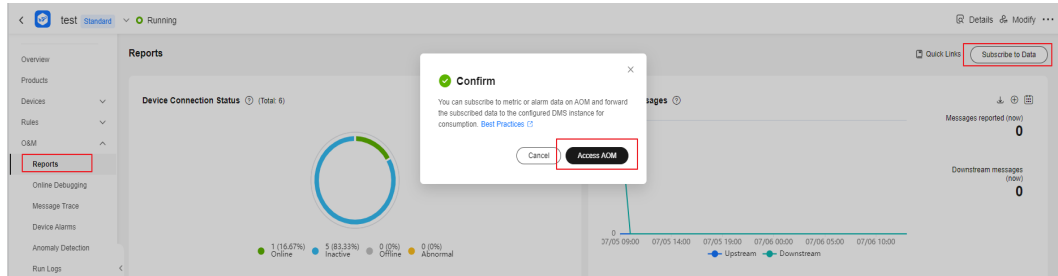
### Creating an AOM Subscription

**Step 1** Visit the [IoTDA](#) service page, and click **Access Console**.



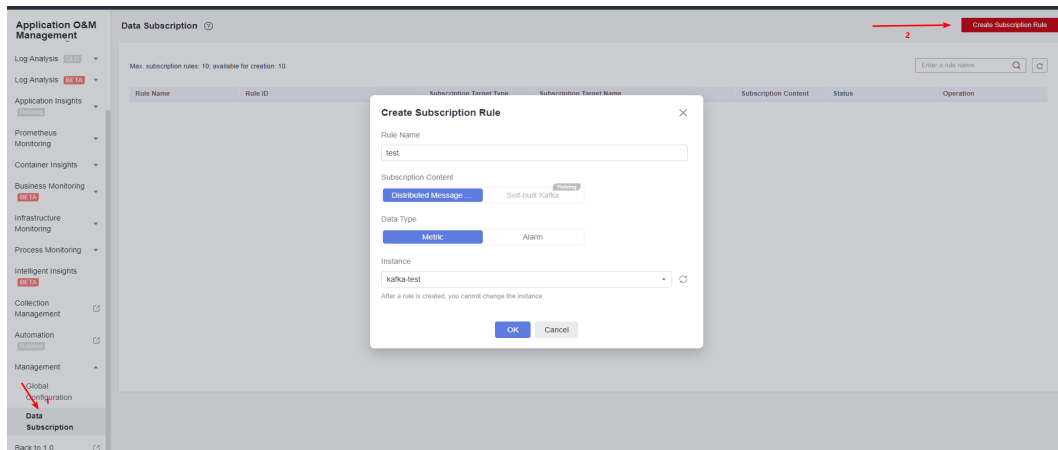
**Step 2** In the navigation pane, choose **O&M > Reports**, click **Subscribe to Data**, and click **Access AOM**.

**Figure 4-1** Creating a data subscription - going to AOM



**Step 3** In the navigation pane, choose **Management > Data Subscription**, click **Create Subscription Rule**, select the subscription content, data type, and DMS instance, and click **OK**.

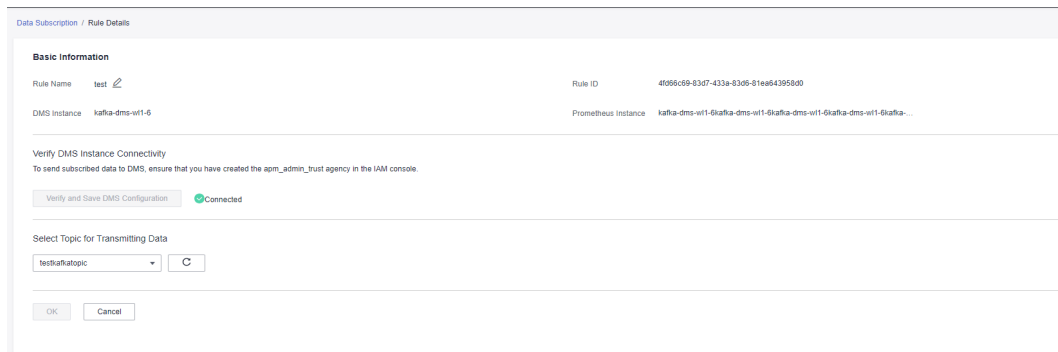
**Figure 4-2** Creating a data subscription - AOM



**Step 4** When configuring DMS subscription for the first time, you need to configure an IAM agency. For details, see [How Do I Create the apm\\_admin\\_trust Agency?](#)

**Step 5** Click **Verify and Save DMS Configuration** to verify the connectivity between AOM and DMS. After the connection is successful, select the data sending topic and click **OK**.

**Figure 4-3** Rule details - AOM



----End



# 5 Device Linkage

---

## 5.1 Triggering Alarms and Sending Email or SMS Notifications

### Introduction

Many IoT devices run 24 hours a day. Device administrators do not need to know the real-time device status. They only need to be notified of certain statuses.

IoT Device Access provides the rule engine function to meet this requirement. You can set rules to enable the platform to send a notification when the data reported by a device meets a certain condition.

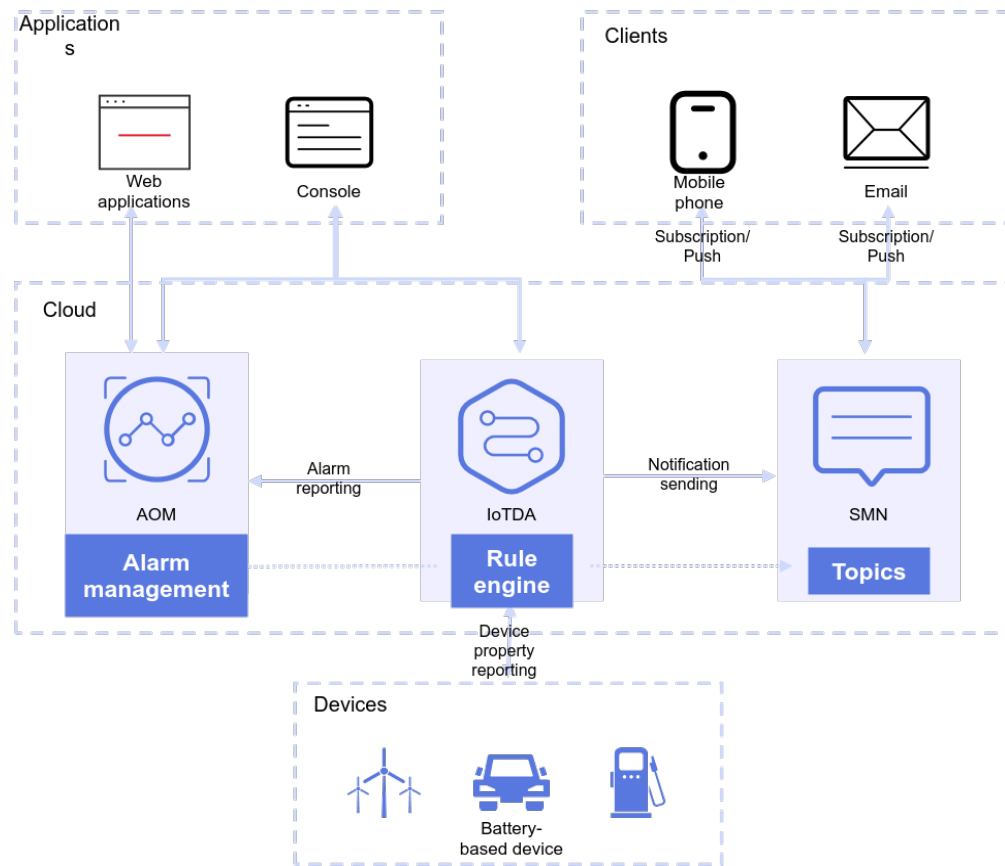
### Example Scenario

In this example, when the battery level reported by the device is lower than 20%, the IoT platform reports an alarm and sends an email or SMS notification to the specified mobile number.

The procedure is as follows:

1. **Simple Message Notification (SMN)** : Subscribe to notification channels.
2. **IoTDA**: Define product models, create devices, and configure linkage rules.
3. **Perform verification**.

**Figure 5-1 Example**



## Configuring SMN

On the Simple Message Notification (SMN) console, create a topic and add a subscription for the IoT Device Access service to invoke to send emails or SMS messages.

1. Log in to Huawei Cloud and access [Simple Message Notification \(SMN\)](#).
2. Click **Access Console**. If you have not subscribed to SMN, subscribe to it first.
3. Choose **Topic Management** > **Topics** page, and click **Create Topic**.
4. Enter the topic name, for example, **Test\_1**, and click **OK**.

Figure 5-2 Creating a topic - SMN

**Create Topic** ×

\* Topic Name  ?  
The name cannot be changed after the topic is created.

Display Name  ?

\* Enterprise Project   ? [Create Enterprise Project](#)

CTS Log

Tag  
It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. [View predefined tags](#)   
To add a tag, enter a tag key and a tag value below.

Tags you can still add: 20

5. Choose **Topic Management > Subscriptions**, and click **Add Subscription**.
6. Enter the subscription information and click **OK**.

Figure 5-3 Adding a subscription - SMN

**Add Subscription** ×

Topic Name Test\_1

\* Protocol

\* Endpoint ?

Endpoints	Description
<input type="text"/>	<input type="text"/>

[Batch Add Endpoints](#)

**Table 5-1** Parameters for adding a subscription

Parameter	Description
Topic Name	Select the topic created in <a href="#">4</a> .
Protocol	<ul style="list-style-type: none"><li>To send an email notification, select <b>Email</b>.</li><li>To send an SMS notification, select <b>SMS</b>.</li></ul>
Endpoint	<ul style="list-style-type: none"><li>If <b>Protocol</b> is set to <b>Email</b>, enter the email address for receiving notifications.</li><li>If <b>Protocol</b> is set to <b>SMS</b>, enter the mobile number for receiving notifications.</li></ul> To add multiple endpoints, place one endpoint in a line. A maximum of 10 lines can be entered.

## Configuring IoTDA

Using IoT Device Access, you can create a product model, register a device, and set a device linkage rule. When the device reports specific data, an alarm is triggered and an email or SMS notification is sent.

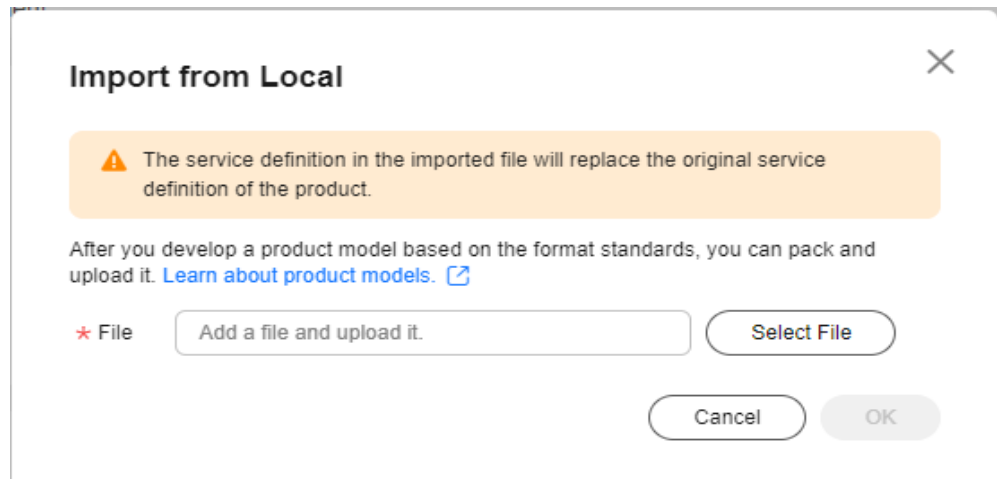
1. Visit the [IoTDA](#) product page and click **Access Console**. Click the target instance card.
2. In the navigation pane, choose **Products**.  
**Note:** The product model and device used in this document are only examples. You can use your own product model and device.
3. Click **Create Product** to create a product using MQTT. Set the parameters and click **OK**.

**Table 5-2** Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, <b>MQTT_Device</b> .
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Industry	Set the parameters as required.
Device Type	

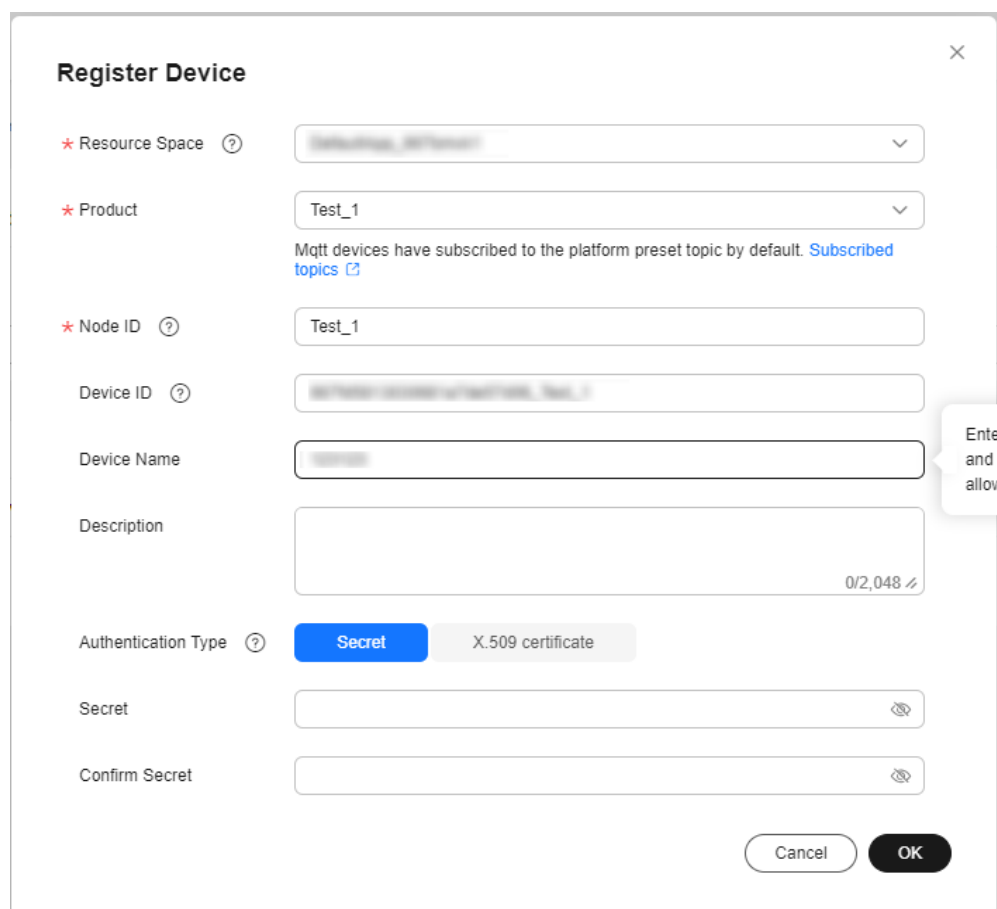
4. Click [here](#) to download a sample product model.
5. Click the created product. The product details page is displayed.
6. On the **Basic Information** tab page, click **Import from Local**. In the displayed dialog box, load the local product model and click **OK**.

Figure 5-4 Uploading a model file



7. In the navigation pane, choose **Devices > All Devices**. On the displayed page, click **Register Device**. On the displayed page, set device registration parameters. Click **OK**. Save the device ID and secret returned after the registration is successful.

Figure 5-5 Registering a device - MQTT

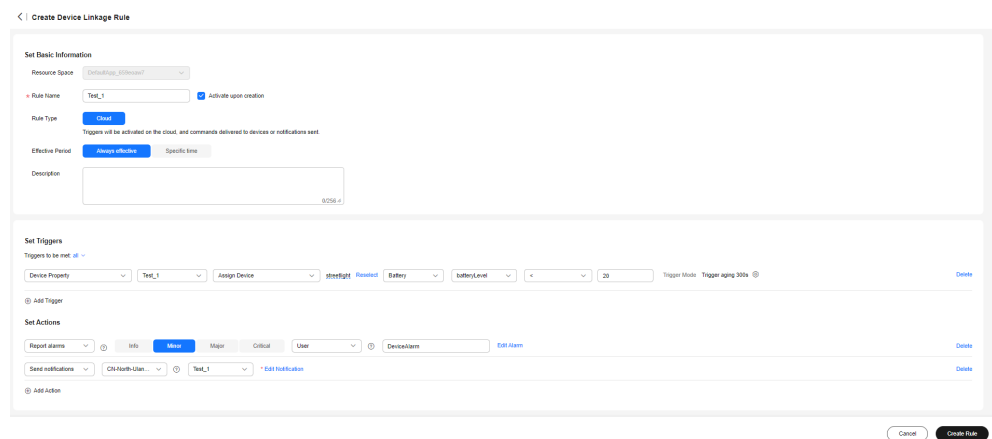


**Table 5-3** Parameters for registering a device

Parameter	Description
Product	Select the product created in <a href="#">3</a> .
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

- In the navigation pane, choose **Rules > Device Linkage**, and click **Create Rule**. (Before creating a rule, select the resource space to which the rule will belong.)
- Set the parameters based on the table below. The following parameter values are only examples. You can create your own rules by referring to [User Guide](#). After setting the parameters, click **Create Rule**.

**Figure 5-6** Creating a linkage rule - BatteryProperty



**Table 5-4** Parameters for creating a linkage rule

Parameter	Description
Rule Name	Enter a name, such as <b>Battery_Low_Mail</b> or <b>Battery_Low_SMS</b> .
Activate upon creation	Select <b>Activate upon creation</b> .



Parameter	Description
Effective Period	Select <b>Always effective</b> .
Description	Provide a description of the rule, for example, "When the battery level reported by a device is lower than 20%, an alarm is reported and a notification is sent."
Set Triggers	<ol style="list-style-type: none"><li>1. Click <b>Add Trigger</b>.</li><li>2. Select <b>Device Property</b>.</li><li>3. Select the product added in <b>3</b>, select <b>Assign Device</b>, and then select the device added in <b>7</b>.</li><li>4. Select <b>Battery</b> for <b>Select service</b>, <b>batteryLevel</b> for <b>Select property</b>, <b>&lt;</b> as the operation, and enter <b>20</b>. Click <b>Trigger Mode</b>. In the dialog box displayed, set <b>Trigger Strategy</b> to <b>Repetition suppression</b> and <b>Data Validity Period (s)</b> to <b>3600</b>, and click <b>OK</b>.</li></ol>

Parameter	Description
Set Actions	<p>Add an alarm.</p> <ol style="list-style-type: none"> <li>1. Click <b>Add Action</b>.</li> <li>2. Select <b>Report alarms</b>.</li> <li>3. Set the alarm severity to <b>Minor</b>, alarm isolation level to <b>Device</b>, <b>Alarm Name</b> to <b>Low battery level</b>, and <b>Description</b> to <b>The battery level is lower than 20%. Check and replace the battery in time</b>. Click <b>OK</b>.</li> </ol> <p>Add a notification.</p> <ol style="list-style-type: none"> <li>1. Click <b>Add Action</b>.</li> <li>2. Select <b>Send notifications</b>.</li> <li>3. Select the region where SMN is available, for example, cn-north-4. When you create a rule for connecting to SMN for the first time, a cloud service access authorization window will be displayed based on the cloud service to connect and region. Configure cloud service access authorization as prompted. (You can log in to the SMN console and view the information in the upper left corner.)</li> <li>4. Select the topic created when <b>configuring SMN</b> for <b>Topic Name</b>. <ul style="list-style-type: none"> <li>• If <b>Protocol</b> corresponding to the topic is <b>Email</b>, set <b>Message Title</b> to an email title, for example, <b>[Huawei IoT Platform] Low Battery Warning</b>, and set <b>Message Content</b> to information similar to <b>You have a device with less than 20% charge, please log in to the Huawei IoT Platform for details</b>.</li> <li>• If <b>Protocol</b> corresponding to the topic is <b>SMS</b>, left <b>Message Title</b> unspecified, and set <b>Message Content</b> to information similar to <b>You have a device with less than 20% charge, please log in to the Huawei IoT Platform for details</b>.</li> </ul> </li> </ol>

## Verifying the Configurations

- You can use a registered physical device to access the platform and enable the device to report the battery level less than 20.
- You can also use a simulator to simulate a device to report the battery level less than 20. For details, see [Developing an MQTT-based Simulated Smart Street Light Online](#).
- You can also use a virtual device for online debugging and enable the device to report the battery level less than 20.

Expected result:

- In the navigation pane on the left, choose **O&M > Device Alarms**. Click **Application Operations Management (AOM)** to go to the AOM console. A minor alarm is generated indicating that the device battery is low.

- If you have subscribed to email notification, the mailbox receives an email indicating that the device battery is low.
- If you have subscribed to SMS notification, the mobile phone receives an SMS notification indicating that the battery level is low.

## 5.2 Automatic Device Shutdown Upon High Temperature

### Introduction

The IoT platform supports device data reporting and command delivery. To associate the two, an application needs to provide corresponding logic.

However, with the rule engine function provided by IoT Device Access, the platform can automatically deliver specified commands when specific data is reported, reducing the application development workload.

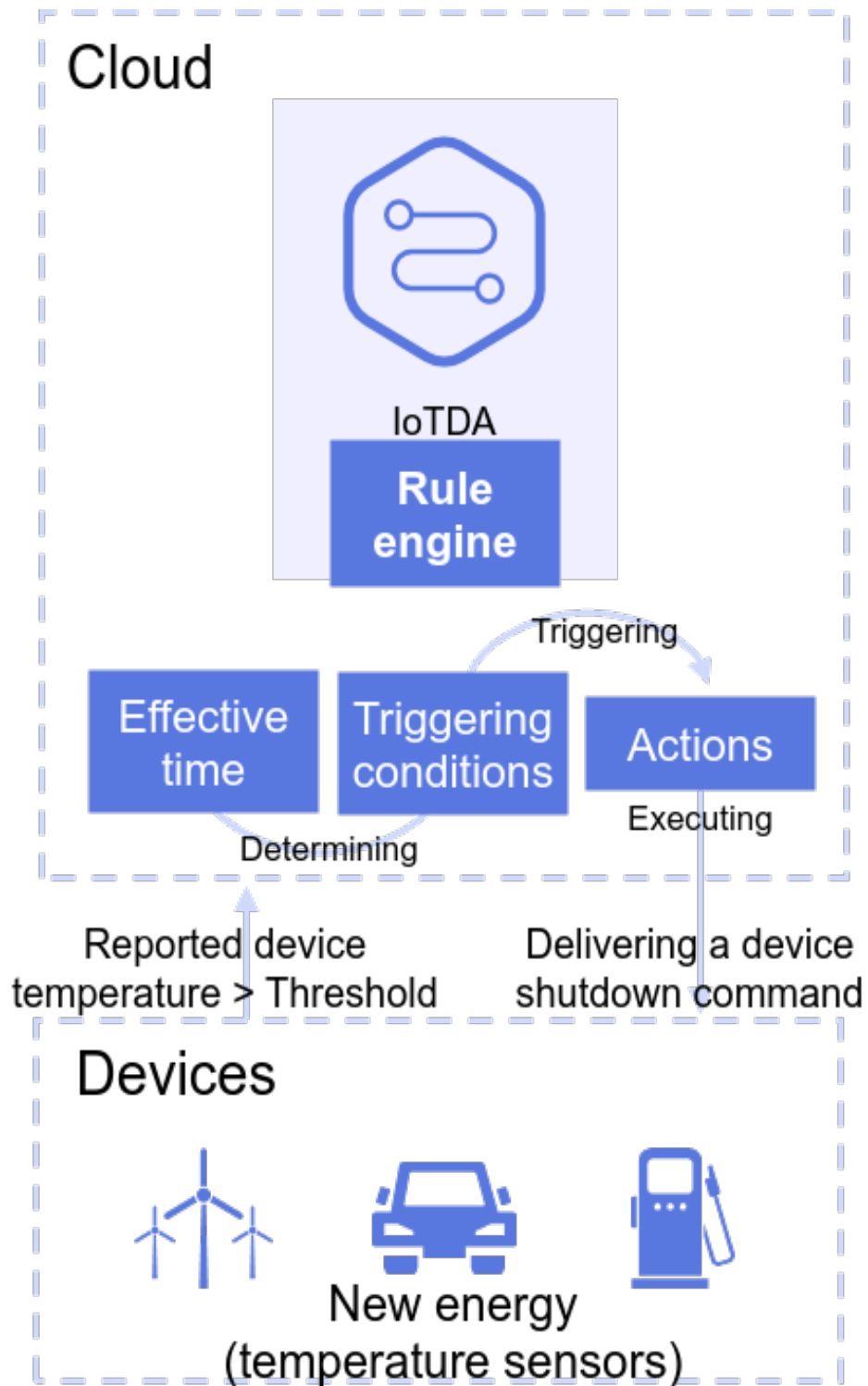
### Example Scenario

In this example, when the temperature reported by the temperature sensor of a device is higher than 80°C, the IoT platform automatically delivers a command to shut down the device.

The procedure is as follows:

1. **IoTDA**: Define product models, create devices, and configure linkage rules.
2. **Verify the access**.

Figure 5-7 Example Description



## Configuring IoTDA

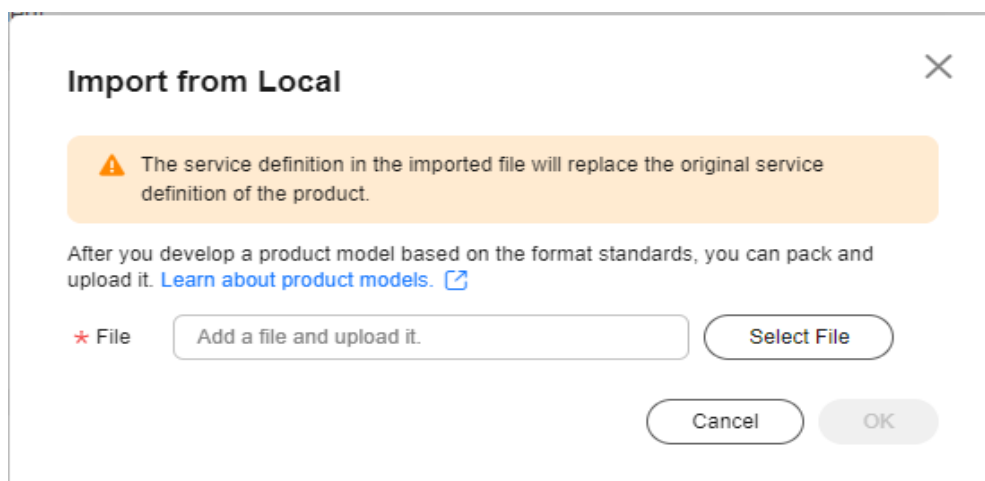
Using IoT Device Access, you can create a product model, register a device, and set a device linkage rule to enable the IoT platform to send a command when receiving specific data from the device.

1. Visit the [IoTDA](#) product page and click **Access Console**. Click the target instance card.
2. In the navigation pane, choose **Products**.  
**Note:** The product model and device used in this document are only examples. You can use your own product model and device.
3. Click **Create Product** to create a product using MQTT. Set the parameters and click **OK**.

**Table 5-5** Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, <b>MQTT_Device</b> .
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Manufacturer	Customize the value.
Industry	Set the parameters as required.
Device Type	

4. Click [Profile\\_tempSensor.zip](#) to download a sample product model.
5. On the **Basic Information** tab page, click **Import from Local**. In the displayed dialog box, load the local product model and click **OK**.

**Figure 5-8** Uploading a model file

6. In the navigation pane, choose **Devices > All Devices**. On the displayed page, click **Register Device**. On the displayed page, set device registration parameters.

**Figure 5-9** Registering a device - MQTT

The screenshot shows a 'Register Device' dialog box with the following fields and options:

- Resource Space:** A dropdown menu.
- Product:** A dropdown menu with 'Test\_1' selected. Below it, a note states: 'Mqtt devices have subscribed to the platform preset topic by default. [Subscribed topics](#)'.
- Node ID:** A text input field containing 'Test\_1'.
- Device ID:** A text input field containing a long alphanumeric string.
- Device Name:** A text input field.
- Description:** A text area with a character count '0/2,048'.
- Authentication Type:** Two radio buttons: 'Secret' (selected) and 'X.509 certificate'.
- Secret:** A text input field with a visibility toggle.
- Confirm Secret:** A text input field with a visibility toggle.
- Buttons:** 'Cancel' and 'OK' buttons at the bottom right.

**Table 5-6** Parameters for registering a device

Parameter	Description
Product	Select the product created in <a href="#">3</a> .
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

Click **OK**. Save the device ID and secret returned after the registration is successful.

7. In the navigation pane, choose **Rules > Device Linkage**, and click **Create Rule**. (Before creating a rule, select the resource space to which the rule will belong.)
8. Set the parameters based on the table below. The following parameter values are only examples. You can create your own rules by referring to [User Guide](#). After setting the parameters, click **Create Rule**.

**Figure 5-10** Creating a linkage rule - TemperatureProperty

**Table 5-7** Parameters for creating a linkage rule

Parameter	Description
Rule Name	Specify the name of the rule to be created, for example, <b>Overheated</b> .
Activate upon creation	Select <b>Activate upon creation</b> .
Effective Period	Select <b>Always effective</b> .
Description	Provide a description of the rule, for example, "The device is automatically shut down when the device temperature is higher than 80°C."
Set Triggers	<ol style="list-style-type: none"> <li>1. Click <b>Add Trigger</b>.</li> <li>2. Select <b>Device Property</b>.</li> <li>3. Select the product added in <b>3</b>, select <b>Assign Device</b>, and then select the device added in <b>6</b>.</li> <li>4. Select <b>tempSensor</b> for <b>Select service</b>, <b>temperature</b> for <b>Select property</b>, <b>&gt;</b> as the operation, and enter <b>80</b>. Click <b>Trigger Mode</b>. In the dialog box displayed, set <b>Trigger Strategy</b> to <b>Repetition suppression</b> and <b>Data Validity Period (s)</b> to <b>300</b>, and click <b>OK</b>.</li> </ol>

Parameter	Description
Set Actions	<ol style="list-style-type: none"><li>1. Click <b>Add Action</b>.</li><li>2. Select <b>Deliver commands</b>, and select the device created in <a href="#">6</a>.</li><li>3. Select <b>deviceSwitch</b> for <b>Select service</b>, and <b>ON_OFF</b> for <b>Select command</b>. Click <b>Configure Parameter</b>. In the dialog box displayed, set <b>power</b> to <b>OFF</b>, and click <b>OK</b>.</li></ol>

## Verifying the Configurations

- You can use a registered physical device to access the platform and enable the device to report the temperature greater than 80.
- You can also use a simulator to simulate a device to subscribe to Topic: **\$oc/devices/{device\_id}/sys/properties/report** (replace **{device\_id}** with the actual device ID) and report data whose temperature is greater than 80. For details, see [Developing an MQTT-based Simulated Smart Street Light Online](#).
- You can also use a virtual device for online debugging and enable the device to report the temperature greater than 80.

Expected result:

- If you use a physical device to report data, the device receives an ON\_OFF command in which **power** is **OFF**.
- If you use a simulator to report data, you can view the ON\_OFF command in which **power** is **OFF** on the **Subscribe** tab page.

## 5.3 Automatically Opening the Window upon High Gas Concentration

### Introduction

IoTDA can instruct a wireless window opener to open the window through a device linkage rule.

### Example Scenario

In this example, a gas detector reports the gas concentration value to the IoT platform. When the gas concentration exceeds a specific threshold, the preset device linkage rule is triggered. The platform delivers a window opening command to the wireless window opener, which then opens the window as instructed.

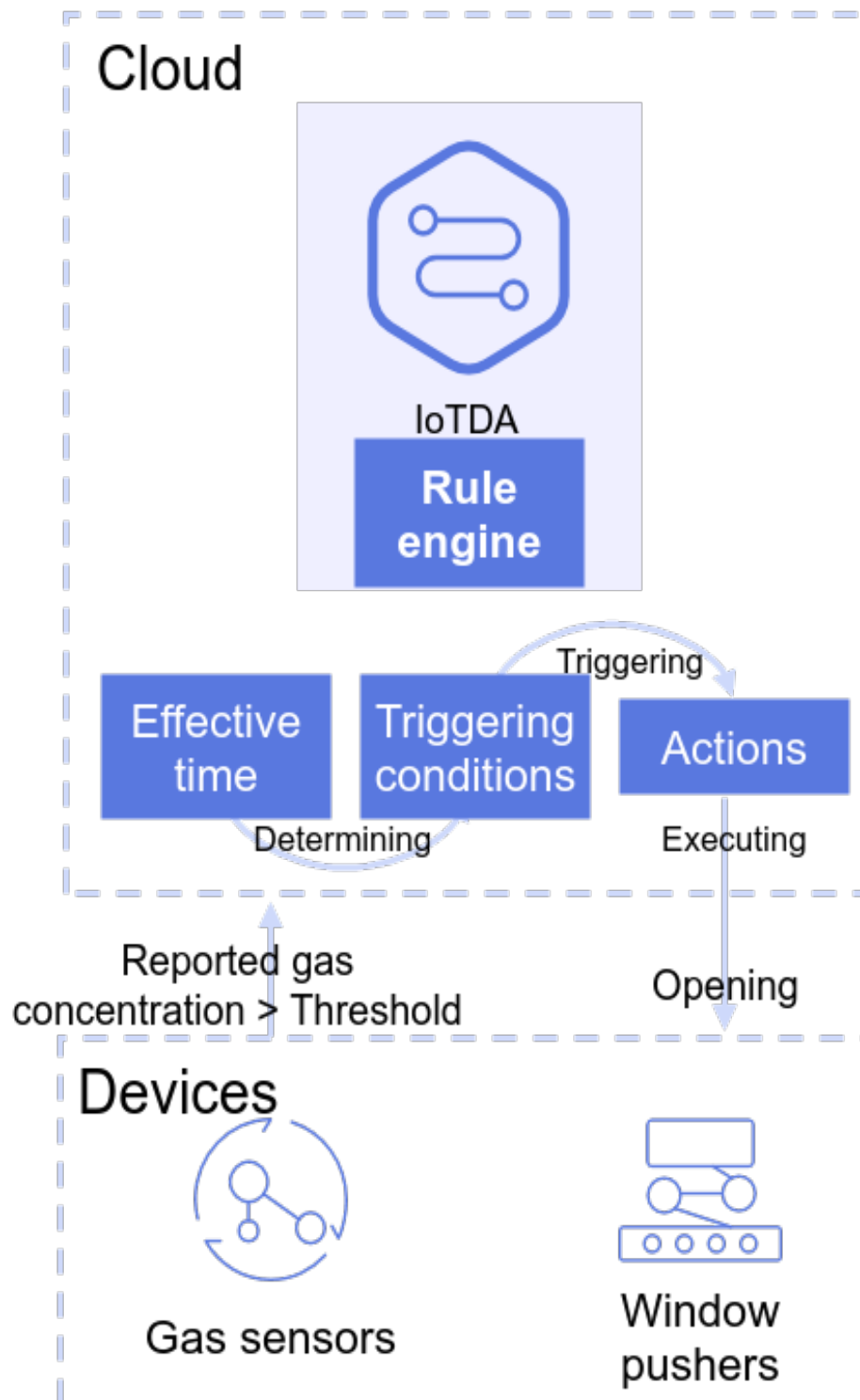
The procedure is as follows:

1. [Create a gas monitoring product](#).
2. [Register the device](#) based on the gas monitoring product.
3. [Configure a device linkage rule](#).



4. Perform verification.

Figure 5-11 Example



### Creating a Gas Monitoring Product

**Step 1** Visit the [IoTDA](#) product page and click **Access Console**. Click the target instance card.

**Step 2** In the navigation pane, choose **Products**.

**Step 3** Click **Create Product** on the left to create a gas monitoring product, set the parameters, and click **OK**.

**Table 5-8** Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, <b>gasdevice</b> .
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Industry	Customize the values.
Device Type	

**Step 4** On the **Basic Information** tab page, click **Customize Model** and configure the product model based on the table below. The **gaslevel** service monitors the gas concentration. The **windowswitch** service executes commands for opening and closing windows.

**Table 5-9** Parameters of the gas concentration monitoring product model

Service ID	Type	Description
gaslevel	Properties	Property Name: gaslevel Data Type: int Access Permissions: Read Value range: 0-100
windowswitch	Commands	Command Name: switch Parameter Name: switch Data Type: enum Enumerated Values: on,off

----End

## Registering a Device

**Step 1** In the navigation pane, choose **Devices > All Devices**. On the displayed page, click **Register Device**. On the displayed page, set device registration parameters. Register the gas monitoring device and record the device ID and secret.

**Figure 5-12** Registering a device - gasdevice

The screenshot shows a 'Register Device' dialog box with the following fields and values:

- Resource Space:** [Dropdown menu]
- Product:** gasdevice
- Node ID:** gasdevice
- Device ID:** [Auto-generated ID]
- Device Name:** gasdevice
- Description:** [Empty text area]
- Authentication Type:** Secret (selected), X.509 certificate
- Secret:** [Empty password field]
- Confirm Secret:** [Empty password field]

Buttons: Cancel, OK

**Table 5-10** Parameters for registering a device

Parameter	Description
Product	Select the product created in step 3.
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

**Step 2** In the navigation pane, choose **Devices > All Devices**. On the displayed page, click **Register Device**. On the displayed page, set device registration parameters. Register a window opener device and record the device ID and secret.

**Figure 5-13** Registering a device - windowswitch

The screenshot shows a 'Register Device' dialog box with the following fields and values:

- Resource Space:** [Dropdown menu]
- Product:** gasdevice
- Node ID:** windowswitch
- Device ID:** [Auto-generated ID]
- Device Name:** windowswitch
- Description:** [Empty text area]
- Authentication Type:** Secret (selected), X.509 certificate
- Secret:** [Empty text field]
- Confirm Secret:** [Empty text field]

Buttons: Cancel, OK

**Table 5-11** Parameters for registering a device

Parameter	Description
Product	Select the product created in <b>3</b> .
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

----End

## Configuring a Device Linkage Rule

- Step 1** In the navigation pane, choose **Rules > Device Linkage**, and click **Create Rule**. (Before creating a rule, select the resource space to which the rule will belong.)
- Step 2** Set the parameters based on the table below. The following parameter values are only examples. You can create your own rules by referring to [User Guide](#). After setting the parameters, click **Create Rule**.

**Figure 5-14** Creating a linkage rule - GasLevel

**Table 5-12** Parameters for creating a linkage rule

Parameter	Description
Rule Name	Specify the name of the rule to create, for example, <b>windowswitch</b> .
Activate upon creation	Select <b>Activate upon creation</b> .
Effective Period	Select <b>Always effective</b> .
Description	Enter a description of the rule, for example, "Automatically opens the window when the gas concentration is higher than 6".
Set Triggers	<ol style="list-style-type: none"> <li>1. Click <b>Add Trigger</b>.</li> <li>2. Select <b>Device Property</b>.</li> <li>3. Select the product added in <b>3</b>, select <b>Assign Device</b>, and then select the device added in <b>1</b>.</li> <li>4. Select <b>gaslevel</b> for <b>Select service</b>, <b>gaslevel</b> for <b>Select property</b>, &gt; as the operation, and enter <b>6</b>. Click <b>Trigger Mode</b>. In the dialog box displayed, set <b>Trigger Strategy</b> to <b>Repetition suppression</b> and <b>Data Validity Period (s)</b> to <b>300</b>, and click <b>OK</b>.</li> </ol>

Parameter	Description
Set Actions	<ol style="list-style-type: none"> <li>1. Click <b>Add Action</b>.</li> <li>2. Select <b>Deliver Commands</b>, and select the device created in <a href="#">2</a>.</li> <li>3. Select <b>windowswitch</b> for <b>Select service</b>, and <b>on_off</b> for <b>Select command</b>. Click <b>Configure Parameter</b>. In the dialog box displayed, set <b>switch</b> to <b>on</b>, click <b>OK</b>.</li> </ol>

----End

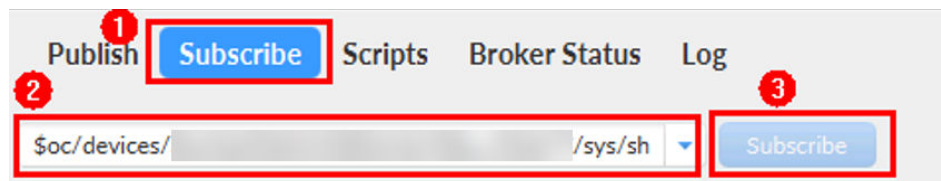
## Verifying the Configurations

### Method 1:

You can use MQTT.fx to simulate device verification.

1. Use MQTT.fx to simulate a gas detector and a window opener, and connect them to the platform. For details, see [Developing an MQTT-based Simulated Smart Street Light Online](#).
2. Open MQTT.fx that simulates the window opener to subscribe to commands delivered by the platform.
  - a. Click the **Subscribe** tab.
  - b. Enter **Topic=\$oc/devices/{device\_id}/sys/commands/#** of the command delivered by the subscription platform. (Replace *{device\_id}* with the device ID obtained in [2](#).)
  - c. Click **Subscribe** to deliver the subscription.

**Figure 5-15** Creating an MQTT subscription



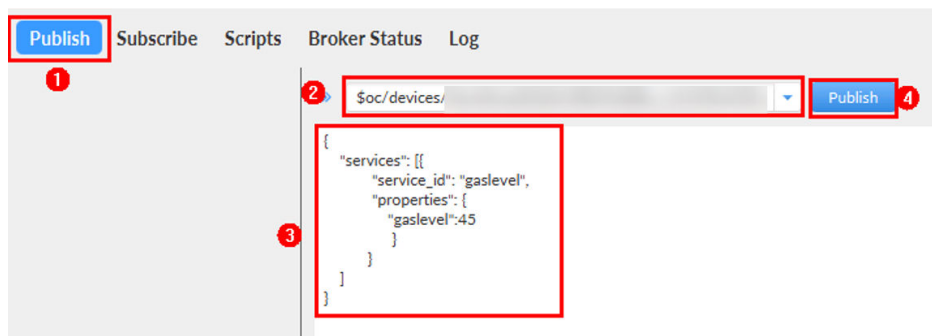
3. Switch to MQTT.fx that simulates the gas detector to report properties.
  - a. Click the **Publish** tab.
  - b. Enter topic **\$oc/devices/{device\_id}/sys/properties/report** for property reporting. (Replace *{device\_id}* with the device ID obtained in [1](#).)
  - c. Report the property **gaslevel** with a value greater than 6.

Example:

```
{
  "services": [{
    "service_id": "gaslevel",
    "properties": {
      "gaslevel": 45
    }
  }
]
```

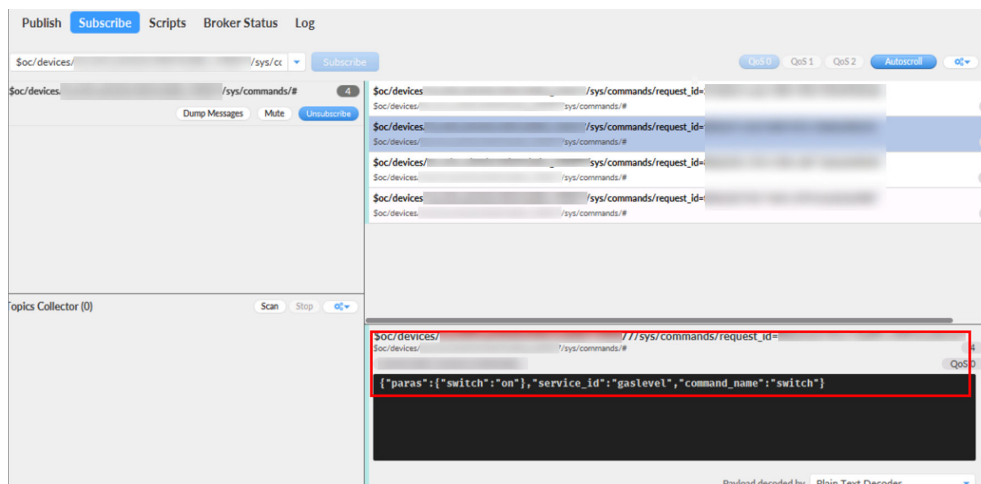
- d. Click **Publish** to report the property value.

**Figure 5-16** MQTT property reporting



- 4. Switch to MQTT.fx that simulates the window opener and click the **Subscribe** tab. The command carrying **switch** with the value set to **on** delivered by the platform is received.

**Figure 5-17** Viewing delivered commands



**Method 2:**

You can use a registered physical device to access the platform and enable the device to report the **gaslevel** greater than 6. The device receives a command carrying **switch** with the value set to **on** and automatically opens the window.

## 5.4 Monitoring Device Status Changes and Sending Notifications

### Introduction

Device administrators need to know connection statuses of IoT devices, such as IoT gateways.

IoTDA provides the rule engine function to meet this requirement. You can easily enable the platform to send a notification when the device status meets a certain condition.

## Example Scenario

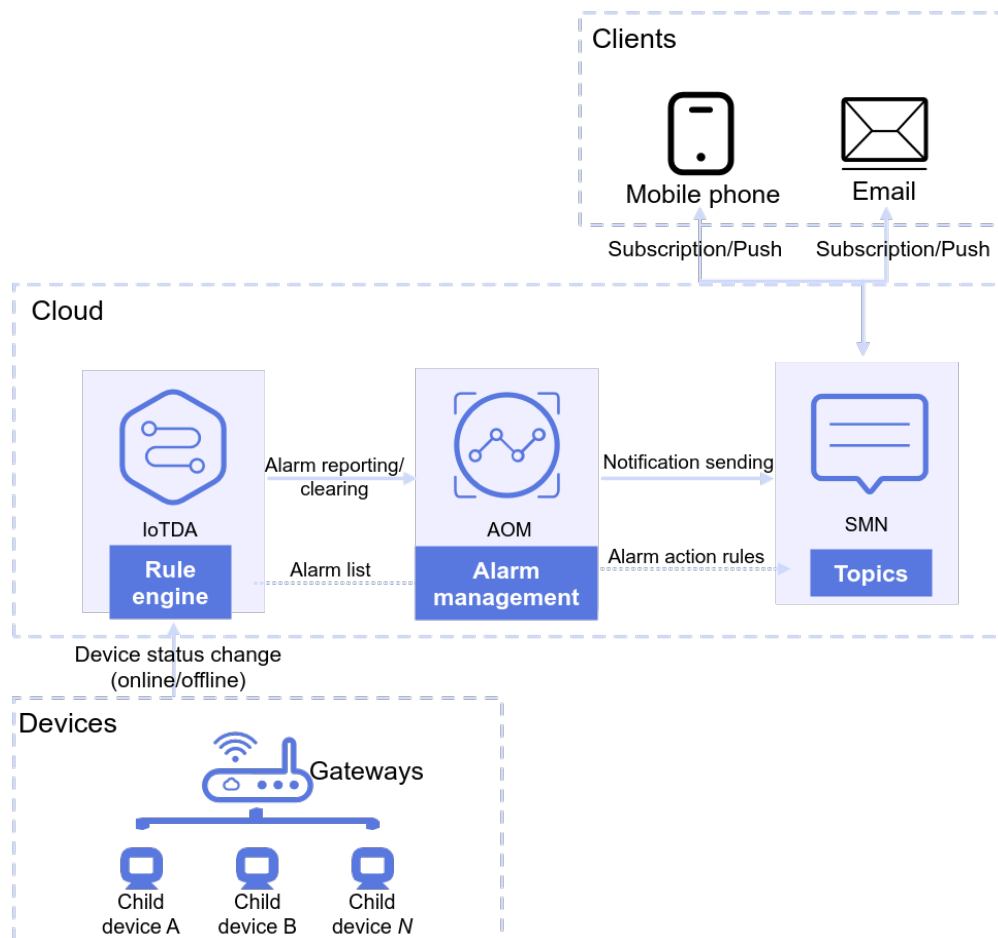
An enterprise has a batch of gateways under a gateway product. About 400 child devices are mounted to a gateway. Administrators need to check gateway statuses in real time to ensure that child devices report data properly. In addition, the gateways are connected to the IoT platform over the 4G network, frequent alarms are generated due to network jitter. Administrators consider that the scenario where devices go offline and quickly go online is normal and do not want to be notified of this scenario.

The following example shows how to monitor all gateways. When a gateway is offline for 5 minutes, an alarm is reported by IoTDA. When the gateway is back to online for 1 minute, the alarm is cleared and an email or SMS is sent to a specified address or mobile number.

The procedure is as follows:

1. **Configure IoTDA.** Create an IoT product and device and create a linkage rule so that alarms can be sent to Application Operations Management (AOM) when device status conditions are met.
2. **Configure Simple Message Notification (SMN).** Create an SMS or email subscription.
3. **Configure AOM.** Create alarm rules to process alarms reported by IoTDA and send SMS or email notifications using SMN.

Figure 5-18 Example





## Configuring IoTDA

Create a product model, register a device, and configure a device linkage rule on IoTDA. When a device is offline for 5 minutes, an alarm is reported to AOM. This alarm is cleared 1 minute after the device goes online.

### NOTE

The product model and device used are only examples. You can use your own product model and device.

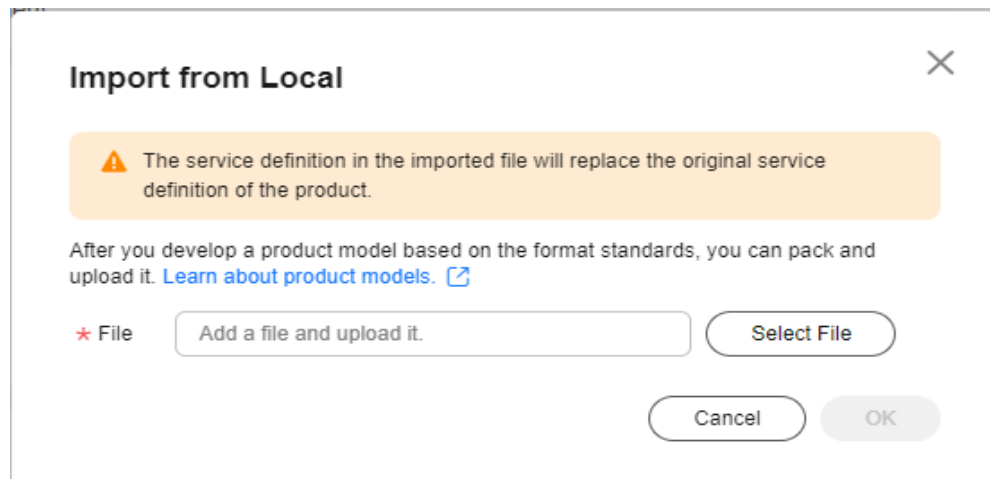
- Step 1** Visit the [IoTDA](#) service page and click **Access Console**. Click the target instance card.
- Step 2** Choose **Products** in the navigation pane and click **Create Product** on the left to create an MQTT product. Set the parameters and click **OK**.

**Table 5-13** Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, <b>MQTT_Device</b> .
Protocol	Select <b>MQTT</b> .
Data Type	Select <b>JSON</b> .
Industry	Set the parameters as required.
Device Type	

- Step 3** Click [here](#) to download a sample product model.
- Step 4** Click the created product. The product details page is displayed.
- Step 5** On the **Basic Information** tab page, click **Import from Local**. In the dialog box displayed, load the local product model and click **OK**.

**Figure 5-19** Uploading a model file



**Step 6** In the navigation pane, choose **Devices > All Devices**. On the displayed page, click **Register Device**. On the displayed page, set device registration parameters. Click **OK**. Save the device ID and secret returned after the registration is successful.

**Figure 5-20** Registering a device - gateway

**Table 5-14** Parameters for registering a device

Parameter	Description
Product	Select the product created in 4.
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select <b>Secret</b> .
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

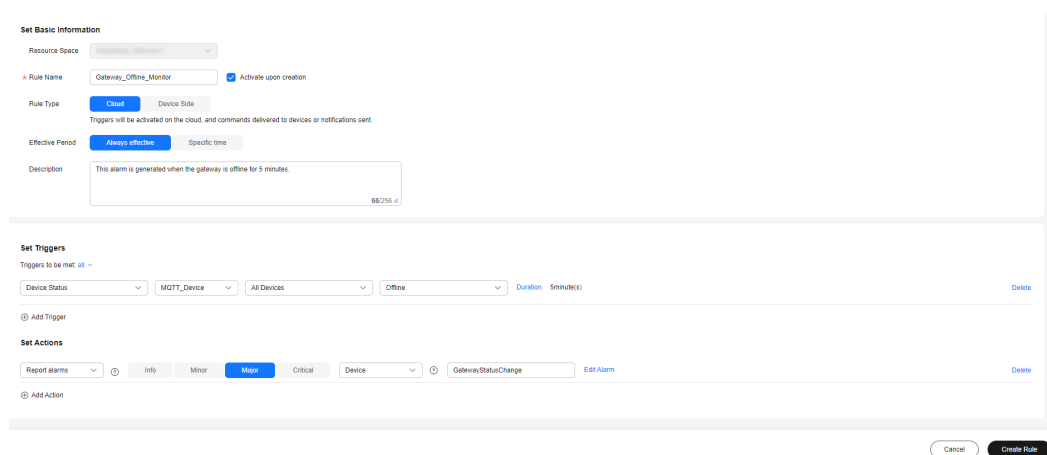
**Step 7** In the navigation pane, choose **Rules > Device Linkage**, and click **Create Rule**. (Before creating a rule, select the resource space to which the rule will belong.)

**Step 8** Configure rule parameters based on the table below to create a rule for reporting offline gateway alarms. The following parameter values are only examples. You can create your own rules by referring to [User Guide](#). After setting the parameters, click **Create Rule**.

**Table 5-15** Parameters for creating a linkage rule

Parameter	Description
Rule Name	Customize a value, for example, <b>Gateway_Offline_Monitor</b> .
Activate upon creation	Select <b>Activate upon creation</b> .
Effective Period	Select <b>Always effective</b> .
Description	Describe the rule, for example, <b>This alarm is generated when the gateway is offline for 5 minutes</b> .
Set Triggers	<ol style="list-style-type: none"> <li>1. Click <b>Add Trigger</b>.</li> <li>2. Select <b>Device Property</b>.</li> <li>3. Select the <b>MQTT_Device</b> product created in <a href="#">4</a>, select <b>All Devices</b>, and select <b>Offline</b> as the trigger status.</li> <li>4. Set <b>Duration</b> to 5 minutes.</li> </ol>
Set Actions	<ol style="list-style-type: none"> <li>1. Click <b>Add Action</b>.</li> <li>2. Select <b>Report alarms</b>.</li> <li>3. Set the alarm severity to <b>Major</b>, alarm isolation level to <b>Device</b>, <b>Alarm Name</b> to <b>GatewayStatusChange</b>, and <b>Description</b> to <b>The gateway is offline. Check the issue and arrange personnel for maintenance in a timely manner.</b>, and click <b>OK</b>.</li> </ol>

**Figure 5-21** Creating a linkage rule - GatewayOffline



**Step 9** Create a linkage rule for alarm clearance.**Figure 5-22** Creating a linkage rule - GatewayOnline

The screenshot displays the configuration interface for a linkage rule. It is divided into two main sections: 'Set Basic Information' and 'Set Triggers'.

**Set Basic Information:**

- Resource Space:** GatewayOnline
- Rule Name:** GatewayOnline\_Monitor
- Activate upon creation:** Checked
- Rule Type:** Cloud (selected), Device Side
- Effective Period:** Always effective (selected), Specific time
- Description:** This alarm is cleared when the gateway is online for 1 minute.

**Set Triggers:**

- Triggers to be met:** all
- Device Status:** Device Status
- MQTT\_Device:** MQTT\_Device
- All Devices:** All Devices
- Online:** Online
- Duration:** 1minute(s)

**Set Actions:**

- Clear alarms:** Clear alarms
- Severity:** Info, Minor, Major (selected), Critical
- Device:** Device
- Action:** GatewayStatusChange

Buttons for 'Cancel' and 'Create Rule' are located at the bottom right of the form.

**NOTICE**

- The alarm name, alarm severity, and alarm isolation dimension together identify an AOM alarm. The three attributes of the alarm to clear must be the same as those specified during alarm reporting. Otherwise, the alarm clearance will fail.
- There is a flow control for device status monitoring. If a large number of devices are monitored, flow control is triggered. As a result, alarms cannot be reported. For details, see [Limitations](#).

----End

## Configuring SMN

On the Simple Message Notification (SMN) console, create a topic and add a subscription for AOM to invoke to send emails or SMS messages.

- Step 1** Log in to Huawei Cloud and access [Simple Message Notification \(SMN\)](#).
- Step 2** Click **Access Console**. If you have not subscribed to SMN, subscribe to it first.
- Step 3** Choose **Topic Management > Topics**, and click **Create Topic**.
- Step 4** Enter a topic name, for example, **Test\_1**, and click **OK**.

Figure 5-23 Creating a topic - SMN

**Create Topic** ×

\* Topic Name  ?  
The name cannot be changed after the topic is created.

Display Name  ?

\* Enterprise Project  ↻ ? [Create Enterprise Project](#)

CTS Log

Tag It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. [View predefined tags](#) ↻  
To add a tag, enter a tag key and a tag value below.

Tags you can still add: 20

**Step 5** Choose **Topic Management > Subscriptions**, and click **Add Subscription**.

**Step 6** Enter the subscription information. Click **OK**.

Figure 5-24 Adding a subscription - SMN

**Add Subscription** ×

---

Topic Name Test\_1

\* Protocol

\* Endpoint ?

Endpoints	Description
<input type="text"/>	<input type="text"/>

[+ Add Endpoint](#)  
[Batch Add Endpoints](#)

---

**Table 5-16** Parameters for adding a subscription

Parameter	Description
Topic Name	Select the topic created in <a href="#">Step 4</a> .
Protocol	<ul style="list-style-type: none"> <li>To send an email notification, select <b>Email</b>.</li> <li>To send an SMS notification, select <b>SMS</b>.</li> </ul>
Endpoint	<ul style="list-style-type: none"> <li>If <b>Protocol</b> is set to <b>Email</b>, enter the email address for receiving notifications.</li> <li>If <b>Protocol</b> is set to <b>SMS</b>, enter the mobile number for receiving notifications.</li> </ul> <p>To add multiple endpoints, place one endpoint in a line. A maximum of 10 lines can be entered.</p>

----End

## Configuring AOM

Create alarm rules and alarm action rules on AOM. When IoTDA alarms are reported, AOM handles the alarm and sends email or SMS notifications.

- Step 1** Log in to Huawei Cloud and visit [AOM](#).
- Step 2** Click **Try Free** to access the AOM console. If you have not subscribed to AOM, subscribe to it first.
- Step 3** Choose **Alarm Center > Alarm Action Rules** and click **Create**.
- Step 4** Enter an alarm action rule name, for example, **Test\_1**, select the **Test\_1** topic created in [Configuring SMN](#), and click the button to confirm the setting.

**Figure 5-25** Creating an action rule - AOM

**Create Alarm Action Rule**

\* Rule Name ?

\* Enterprise Project

Description ? --

\* Action Type  Metric/Event  Log

\* Action

\* Topic  C

If you do not see a topic you like, create one on the SMN console.

\* Message Template  C [Create Template](#) | [View Template](#)

**Step 5** Choose **Alarm Center > Alarm Rules** and click **Create Alarm Rule**.

**Step 6** Enter a rule name, for example, **Gateway\_Status\_Change\_Alarm\_Rule**, select the event alarm and custom event options, enter **IoTDA** in **Alarm Source**, set **Select Object** to **event\_name=GatewayStatusChange** (**GatewayStatusChange** is the alarm name), set **Triggering Policy** to **Immediate Triggering**, set **Alarm Mode** to **Direct Alarm Reporting**, set **Action Rule** to the action rule created in **4**, and click **Create Now** in the lower right corner.

**Figure 5-26** Creating an alarm rule - AOM

The screenshot shows the 'Alarm Rule Settings' and 'Alarm Rule Details' sections of the AOM console. In the 'Alarm Rule Settings' section, 'Rule Type' is set to 'Event alarm rule', 'Event Type' is 'Custom', and 'Event Source' is 'IoTDA'. The 'Alarm Rule Details' section includes a 'Monitored Object' field with the value 'CustomAttributes: event\_name...', a table with columns for 'Event Name', 'Trigger Mode', and 'Alarm Severity', and an 'Alarm Notification' section where 'Alarm Mode' is 'Direct alarm reporting' and 'Action Rule' is 'Test\_1'. A 'Confirm' button is visible at the bottom right.

----End

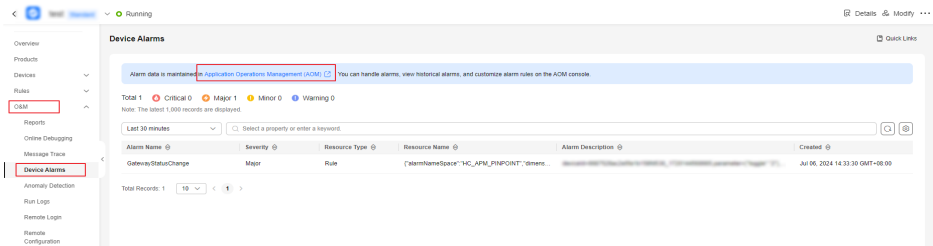
## Verifying Configurations

- You can use a registered physical device to access the platform.
- You can also use a simulator to simulate a device going online and offline. For details, see [Developing an MQTT-based Simulated Smart Street Light Online](#).

Expected result:

1. After the device is offline for 5 minutes:
  - In the navigation pane, choose **O&M > Device Alarms**. Click **Application Operations Management (AOM)** to go to the AOM console. A major **GatewayStatusChange** alarm is generated.

**Figure 5-27** Current alarms - Going to AOM



- If the alarm rule and email notification action are configured in AOM, the target email address will receive an email notifying that the gateway is offline.
  - If the alarm rule and SMS notification action are configured in AOM, the target number will receive an SMS notifying that the gateway is offline.
2. After the device is back online for 1 minute:
- The major alarm is cleared. You can view the alarm in the historical alarm list.
  - If the alarm rule and email notification action are configured in AOM, the target email address will receive an email notifying that the gateway goes online.
  - If the alarm rule and SMS notification action are configured in AOM, the target number will receive an SMS notifying that the gateway goes online.