IoT Device Access

Best Practices

Issue	1.0
Date	2022-06-30

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2023. 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

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.

For details about service scenarios, see **Platform Overview**.

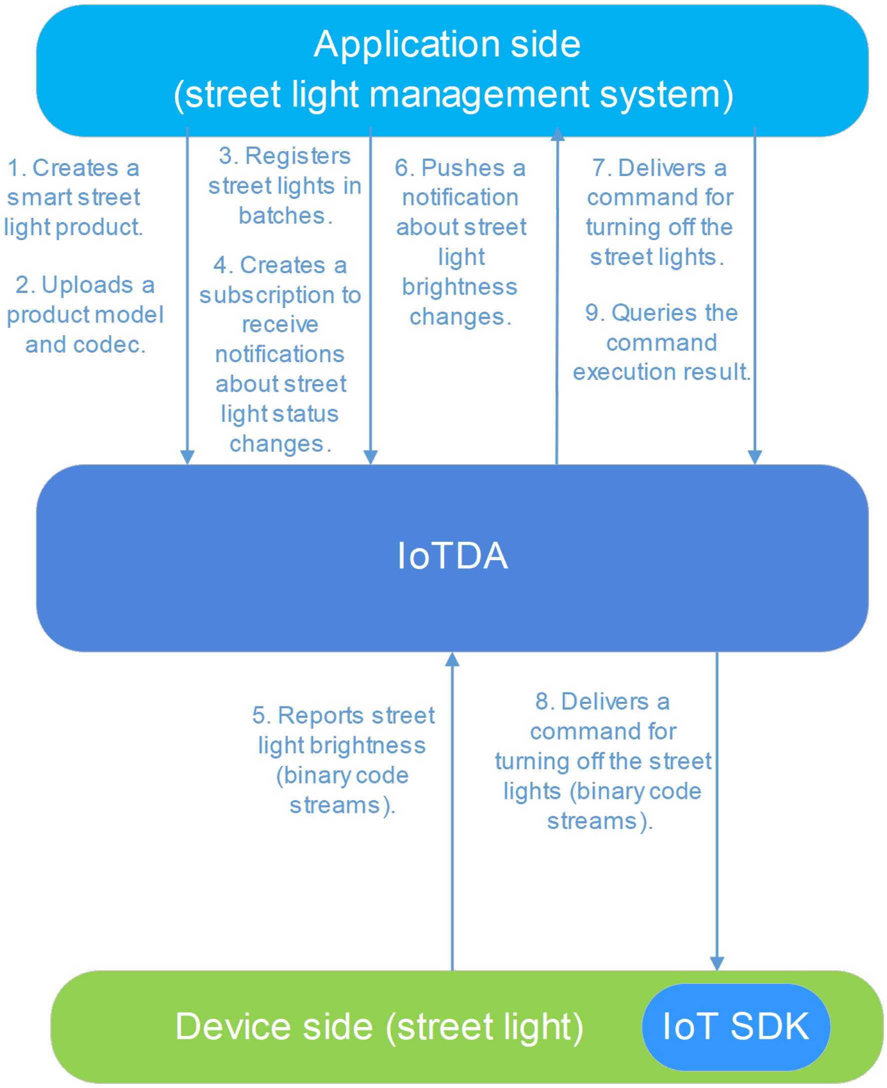
Scenario Example: Smart Street Lamp

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

In this scenario, devices interact with the platform using LwM2M. The application side can subscribe to device change notifications on the platform and deliver commands to devices.

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

For details about this scenario, see **Developing a Smart Street Light Using NB-IoT BearPi**.


Application side
(street light management system)
1. Creates a smart street light product.
2. Uploads a product model and codec.
3. Registers street lights in batches.
4. Creates a subscription to receive notifications about street light status changes.
6. Pushes a notification about street light brightness changes.
7. Delivers a command for turning off the street lights.
9. Queries the command execution result.
IoTDA
5. Reports street light brightness (binary code streams).
8. Delivers a command for turning off the street lights (binary code streams).
Device side (street light)
IoT SDK

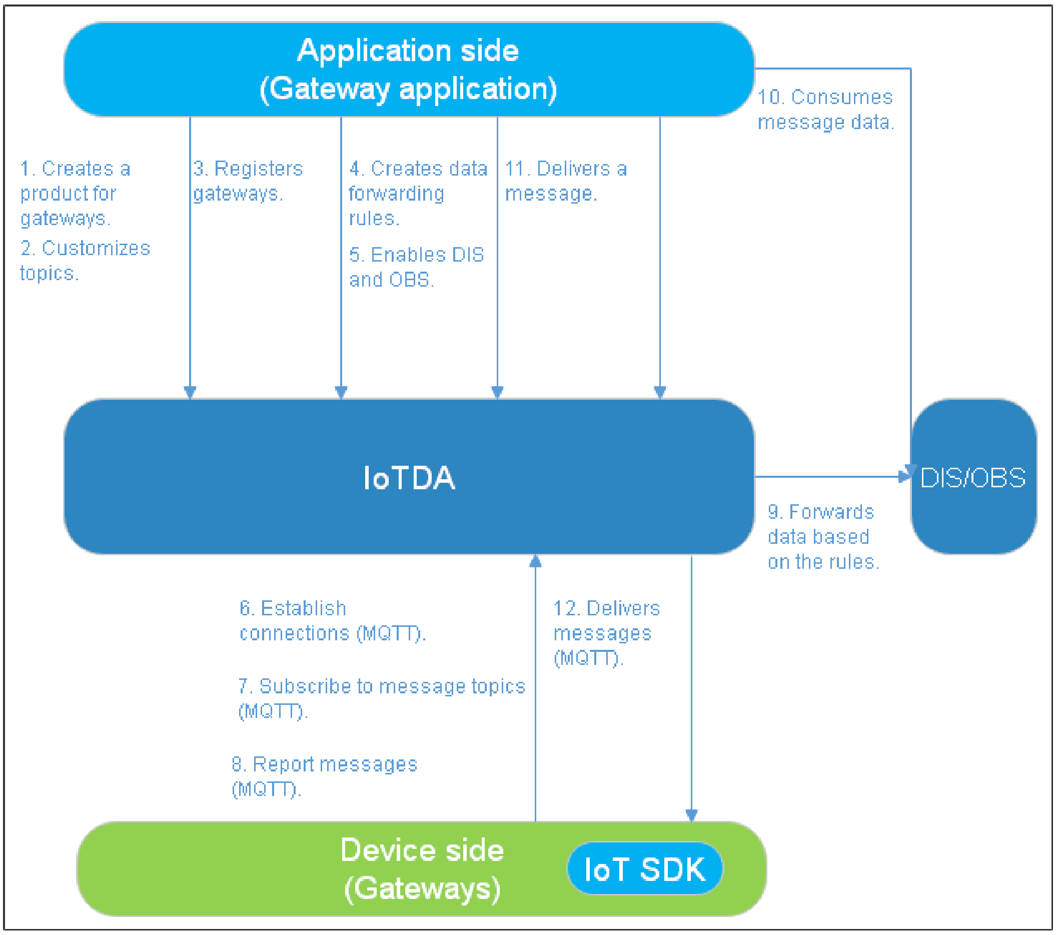
Scenario Example: Smart Gateway

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

For details about this scenario, see **Using a Custom Topic for Communication**.


Application side
(Gateway application)
1. Creates a product for gateways.
2. Customizes topics.
3. Registers gateways.
4. Creates data forwarding rules.
5. Enables DIS and OBS.
11. Delivers a message.
10. Consumes message data.
IoTDA
DIS/OBS
9. Forwards data based on the rules.
6. Establish connections (MQTT).
7. Subscribe to message topics (MQTT).
8. Report messages (MQTT).
12. Delivers messages (MQTT).
Device side
(Gateways)
IoT SDK

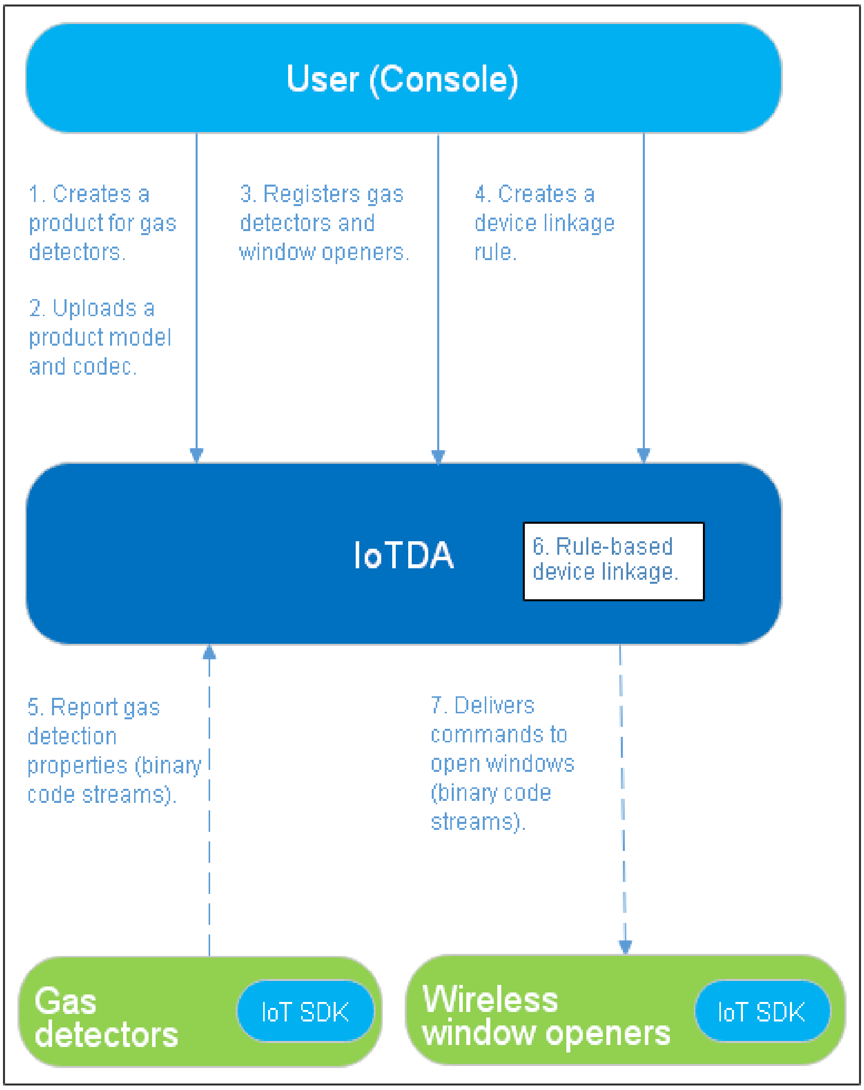
Scenario Example: Smart Home Gas Detection

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**.


User (Console)
1. Creates a product for gas detectors.
2. Uploads a product model and codec.
3. Registers gas detectors and window openers.
4. Creates a device linkage rule.
IoTDA
6. Rule-based device linkage.
5. Report gas detection properties (binary code streams).
7. Delivers commands to open windows (binary code streams).
Gas detectors
IoT SDK
Wireless window openers
IoT SDK

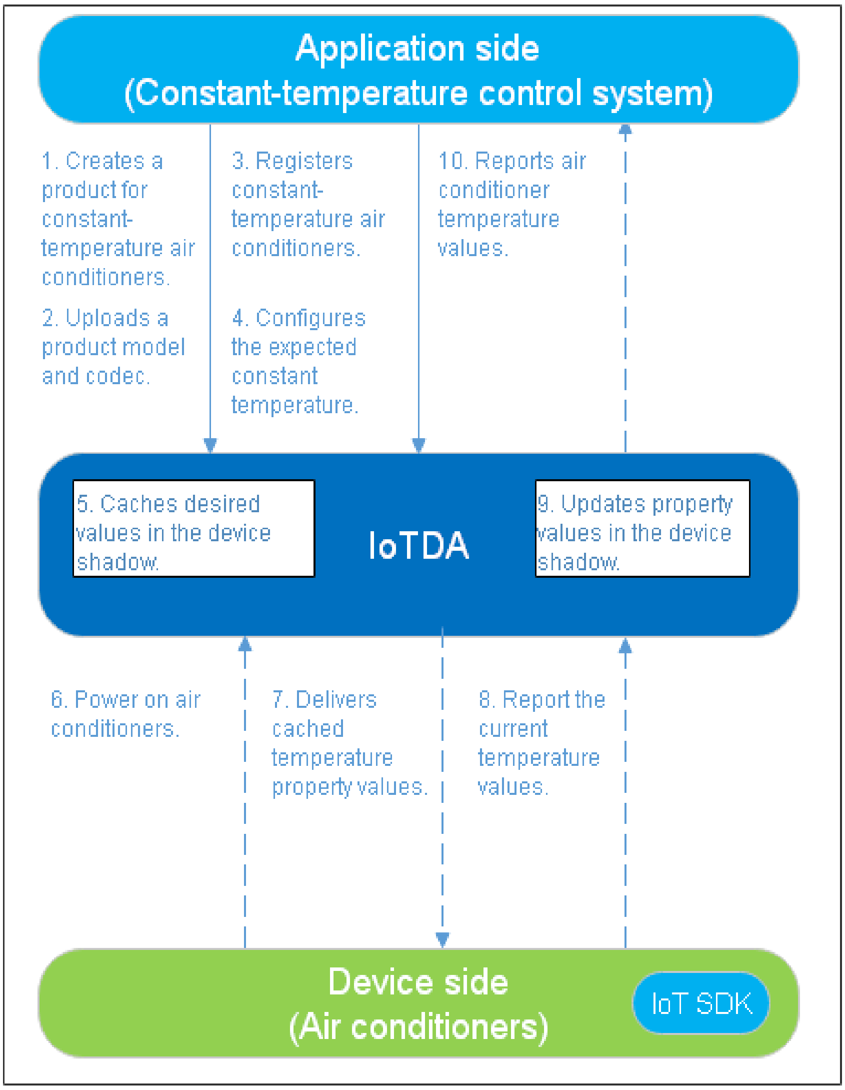
Scenario Example: Constant-Temperature Air Conditioner

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

For details about this scenario, see **Constant-Temperature Air Conditioner**.


Application side
(Constant-temperature control system)
1. Creates a product for constant-temperature air conditioners.
2. Uploads a product model and codec.
3. Registers constant-temperature air conditioners.
4. Configures the expected constant temperature.
10. Reports air conditioner temperature values.
5. Caches desired values in the device shadow.
IoTDA
9. Updates property values in the device shadow.
6. Power on air conditioners.
7. Delivers cached temperature property values.
8. Report the current temperature values.
Device side
(Air conditioners)
IoT SDK

2 Device Access

2.1 Developing an MQTT-based Smart Street Light Online

Scenarios

This topic uses a smart street lamp as an example to describe how to use MQTT.fx device simulators to experience data reporting and command delivery.

Assume that:

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

Service 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** in the upper right corner.

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. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create it first.
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Industry	Select the industry to which the product model belongs.
Device Type	If the product model preset on the platform is used, the device type is automatically matched.
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.
Description	Provide a description for the product. Set this parameter based on the site requirements.

----End

Developing a Product Model

Step 1 Click the product created in **Creating a Product**. The product details page is displayed.

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

Step 3 Add the **BasicData** service.

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

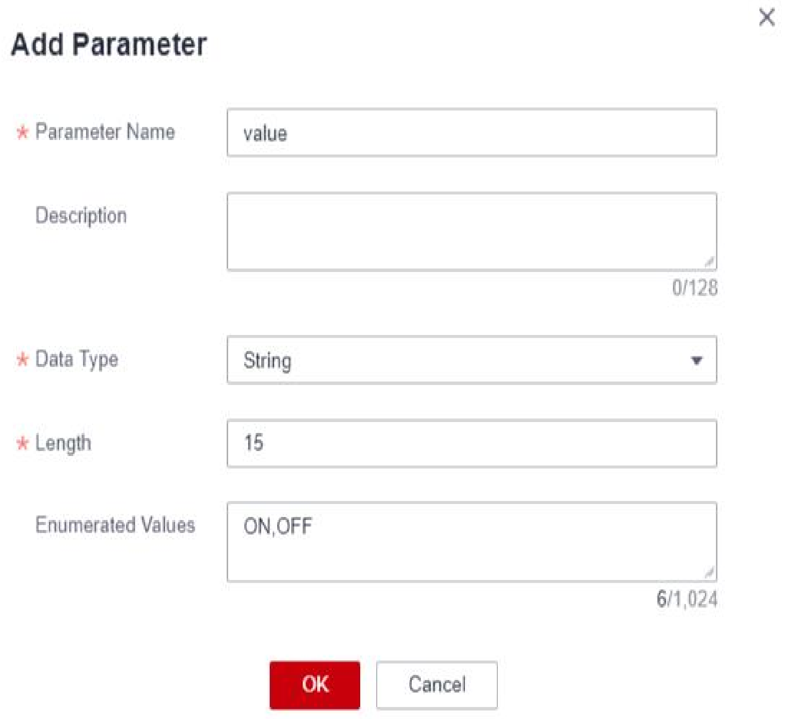
- **Service ID**: Enter **BasicData**.
- **Service Type**: You are advised to set this parameter to the same value as **Service ID**.
- **Description**: Enter **Reports street light data**.

2. In the **BasicData** service list on the right, click **Add Property**, enter related information, and click **OK**.
   - **Property Name**: Enter **luminance**.
   - **Description**: Enter **light intensity**.
   - **Data Type**: Select **Integer**.
   - **Access Permissions**: Select **Read** and **Write**.
   - **Value Range**: Set it to 0–65535.
   - **Step**: Enter **0**.
   - **Unit**: Leave it blank.

Step 4 Add the **LightControl** service.

1. On the **Model Definition** tab page, click **Add Service**, configure **Service ID**, **Service Type**, and **Description**, 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**.
3. On the **Add Command** page, click **Add Command Parameter**, enter related information, and click **OK**.

Figure 2-1 Adding a parameter



Add Parameter

* Parameter Name: value

Description:

0/128

* Data Type: String

* Length: 15

Enumerated Values: ON,OFF

6/1,024

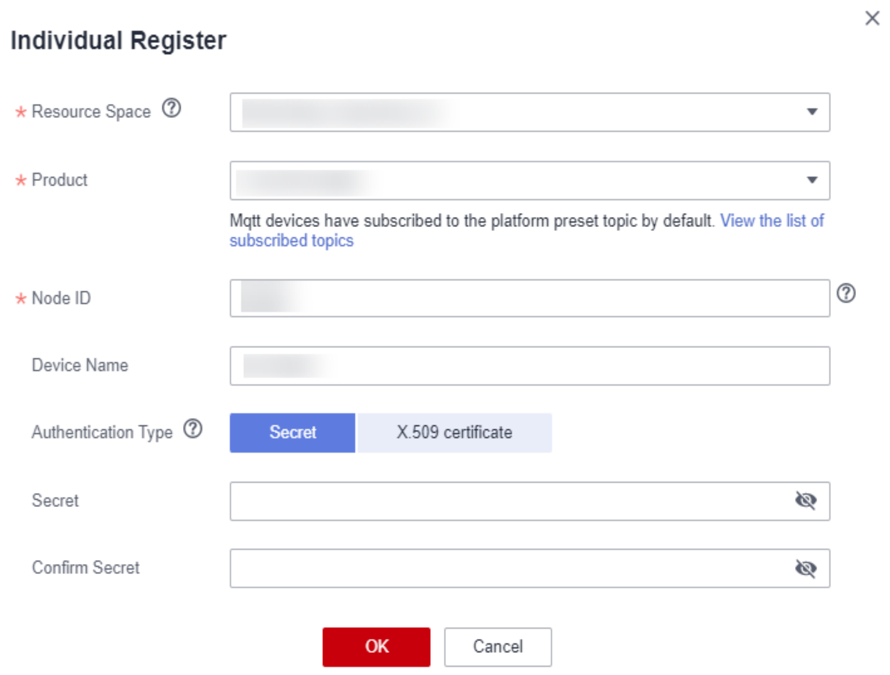
----End

Registering a Device

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

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

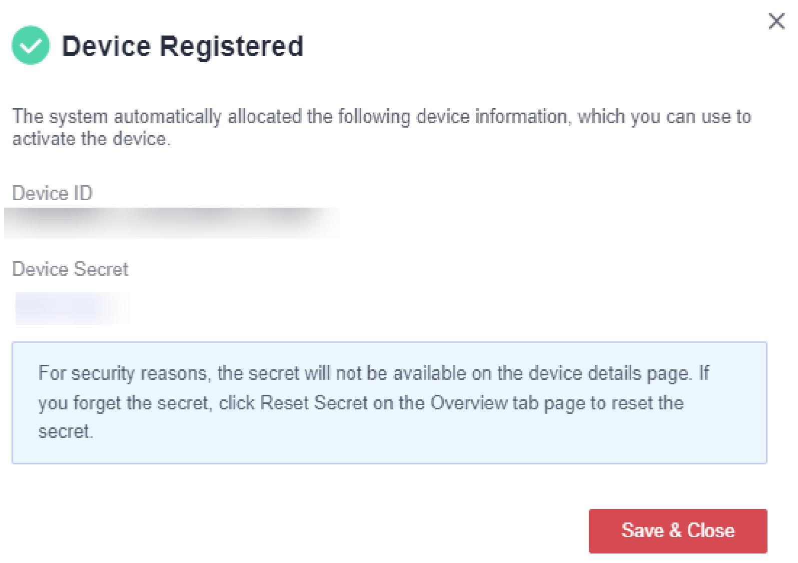
Parameter	Description
Resource Space	Ensure that the device and the product created in 1 belong to the same resource space.
Product	Select the product created in 2.
Node ID	Customize a unique physical identifier for the device. The value consists of letters and numbers.
Device Name	Customize a device name, for example, **streetlight**.
Authentication Type	Select **Secret**.
Secret	If you do not set this parameter, IoTDA automatically generates a value.



Individual Register

* Resource Space	[redacted]
* Product	[redacted] Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics
* Node ID	[redacted]
Device Name	[redacted]
Authentication Type	Secret / X.509 certificate
Secret	
Confirm Secret	

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



Device Registered

The system automatically allocated the following device information, which you can use to activate the device.

Device ID

[redacted]

Device Secret

[redacted]

For security reasons, the secret will not be available on the device details page. If you forget the secret, click Reset Secret on the Overview tab page to reset the secret.

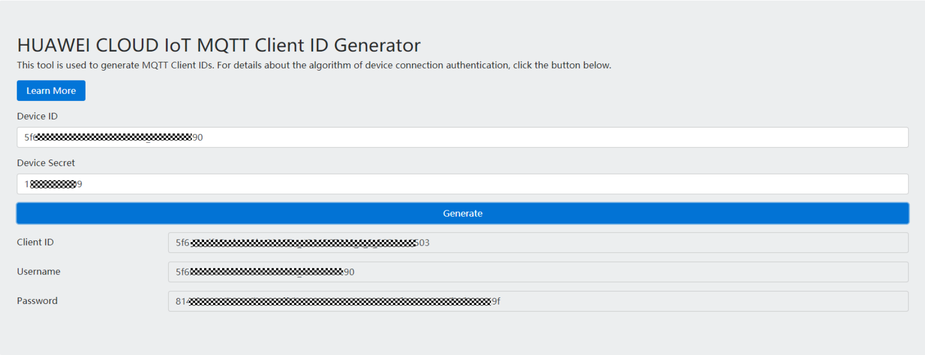
----End

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**).



HUAWEI CLOUD IoT MQTT Client ID Generator

This tool is used to generate MQTT Client IDs. For details about the algorithm of device connection authentication, click the button below.

Device ID: 5f[redacted]90

Device Secret: 1[redacted]9

Client ID: 5f6[redacted]503

Username: 5f6[redacted]90

Password: 814[redacted]9f

Parameter	Mandatory	Type	Description
ClientId	Yes	String(256)	The value of this parameter consists of a device ID, device type, password signature type, and timestamp. They are separated by underscores (_). ● 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 (_). ● Device type: The value is fixed at **0**, indicating a device ID. ● Password signature type: The length is 1 byte, and the value can be **0** or **1**. – **0**: The timestamp is not verified using the HMAC-SHA256 algorithm. – **1**: The timestamp is verified using the HMAC-SHA256 algorithm. ● 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 **2018072417**.
Username	Yes	String(256)	Device ID.
Password	Yes	String(256)	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. The device secret is returned by IoTDA upon successful device registration.

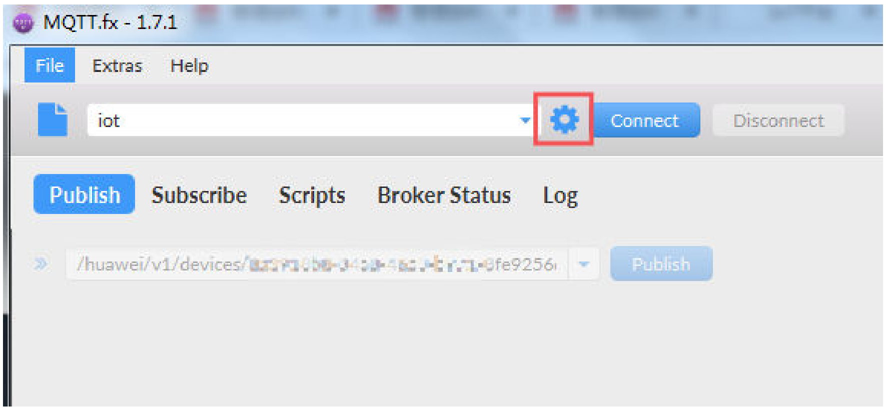
Each device performs authentication using the MQTT CONNECT message, which must contain all information of the client ID. After receiving a CONNECT message, IoTDA checks the authentication type and password digest algorithm of the device.

The generated client ID is in the format ***Device ID*_0_0_*Timestamp***. By default, the timestamp is not verified.

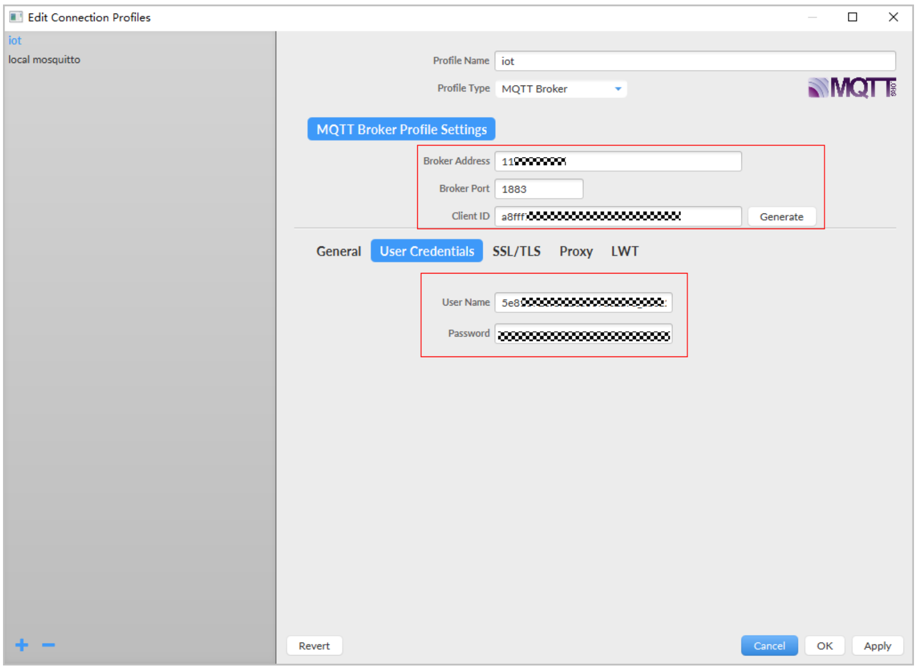
- If the timestamp needs to be verified using the HMAC-SHA256 algorithm, the platform checks whether the message timestamp is consistent with the platform time and then checks whether the password is correct.
- If the timestamp does not need to be verified using the HMAC-SHA256 algorithm, the timestamp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



Profile Name: iot

Profile Type: MQTT Broker

Broker Address: 11[redacted]

Broker Port: 1883

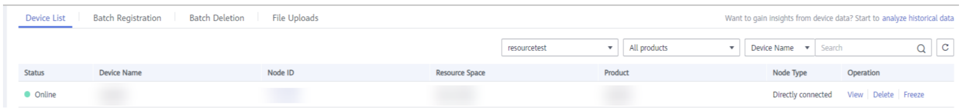
Client ID: a8fff[redacted]

User Name: 5e8[redacted]

Password: [redacted]

Parameter	Description
Broker Address	Enter the **device access address** (domain name) obtained from the IoTDA console. For devices that cannot be connected to the platform using a domain name, run the **ping** *Domain name* command in the CLI to obtain the IP address. The IP address is variable and needs to be set using a configuration item.
Broker Port	The default value is **1883**.
Cliend ID	Enter the device ClientId obtained in **2**.
User Name	Enter the DeviceId 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.



Device List | Batch Registration | Batch Deletion | File Uploads

Want to gain insights from device data? Start to analyze historical data

resourcetest ▾ | All products ▾ | Device Name ▾ | Search

Status	Device Name	Node ID	Resource Space	Product	Node Type	Operation
● Online	[redacted]	[redacted]	[redacted]	[redacted]	Directly connected	View | Delete | Freeze

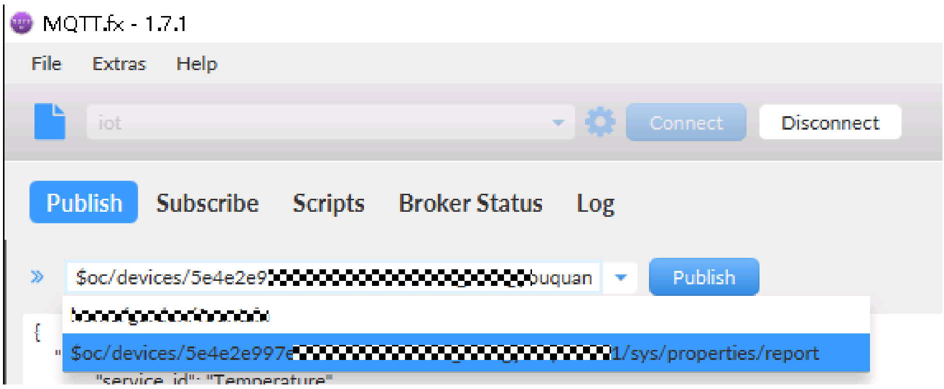
----End

Reporting Data

Use MQTT.fx to report data to IoTDA.

If the 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**.



$oc/devices/5e4e2e9[redacted]ouquan

[redacted]

$oc/devices/5e4e2e997e[redacted]1/sys/properties/report

{
"
"service_id": "Temperature"

Step 2 Enter the reported data in the blank in the middle of the tool interface.

Request parameters

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

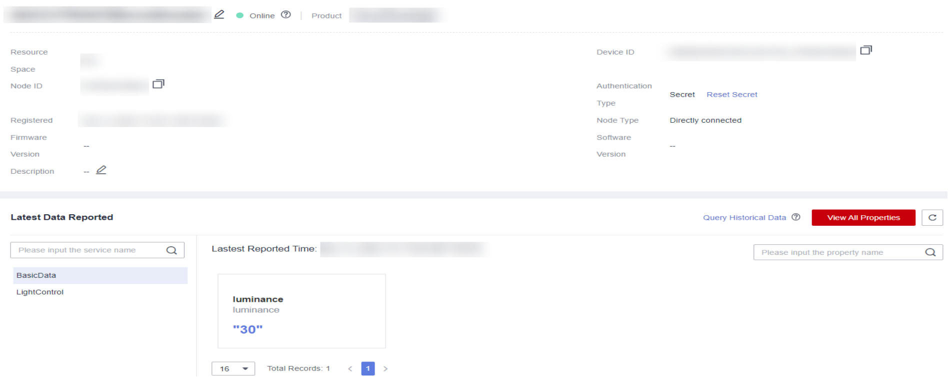
ServiceProperty structure

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.
eventTime	No	String	UTC time when the device reports data. The format is yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is not carried in the reported data or is in incorrect format, the time when IoTDA receives the data is used.

Example request

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

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



[redacted] Online | Product [redacted]

Resource Space: [redacted]
Node ID: [redacted]
Registered: [redacted]
Firmware Version: --
Description: --
Device ID: [redacted]
Authentication Type: Secret Reset Secret
Node Type: Directly connected
Software Version: --

Latest Data Reported

Lastest Reported Time: [redacted]

BasicData
LightControl

luminance
luminance
"30"

16 Total Records: 1 < 1 >

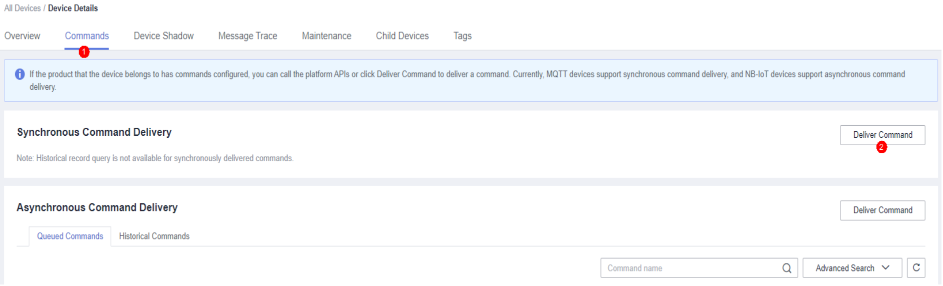
----End

Delivering a Command

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

Step 1 In the navigation pane, choose **Devices** > **All Devices**, locate the device created in **Registering a Device**, and click **View** to access its details.

Step 2 Click the **Commands** tab and deliver a synchronization command.



If the product that the device belongs to has commands configured, you can call the platform APIs or click Deliver Command to deliver a command. Currently, MQTT devices support synchronous command delivery, and NB-IoT devices support asynchronous command delivery.

Synchronous Command Delivery

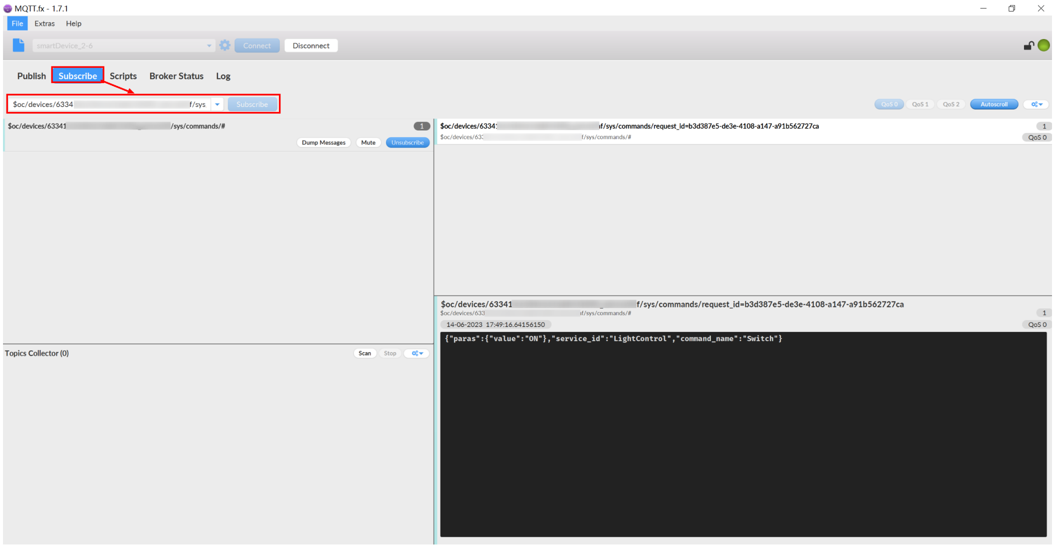
Note: Historical record query is not available for synchronously delivered commands.

Asynchronous Command Delivery

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.



$oc/devices/6334[redacted]f/sys

$oc/devices/63341[redacted]/sys/commands/#

$oc/devices/6334[redacted]f/sys/commands/request_id=b3d387e5-de3e-4108-a147-a91b562727ca
$oc/devices/633[redacted]f/sys/commands/#

$oc/devices/63341[redacted]f/sys/commands/request_id=b3d387e5-de3e-4108-a147-a91b562727ca
$oc/devices/633[redacted]f/sys/commands/#
14-06-2023 17:49:16.64156150

```
{"paras":{"value":"ON"},"service_id":"LightControl","command_name":"Switch"}
```

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 **deviceId** 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

Advanced Experience

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

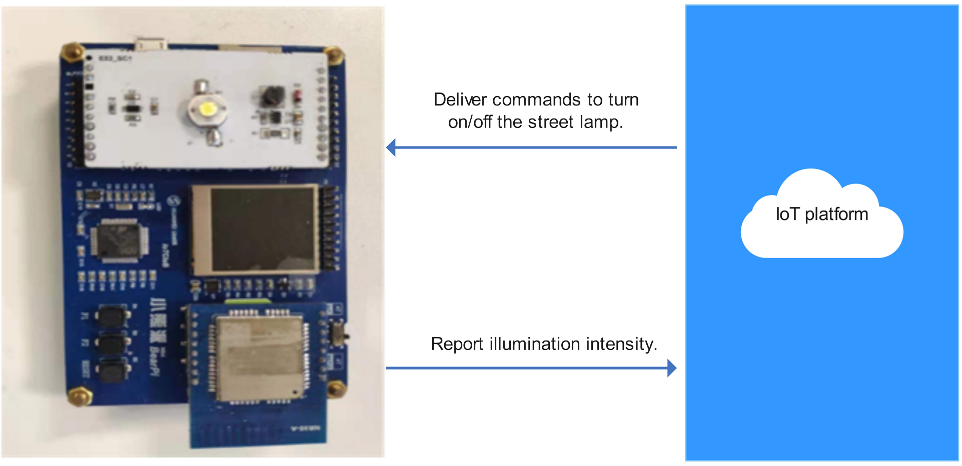
To better experience the IoTDA service, develop real-world applications and devices and connect them to the platform. For details, see **Developer Guide**.

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 topic 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.

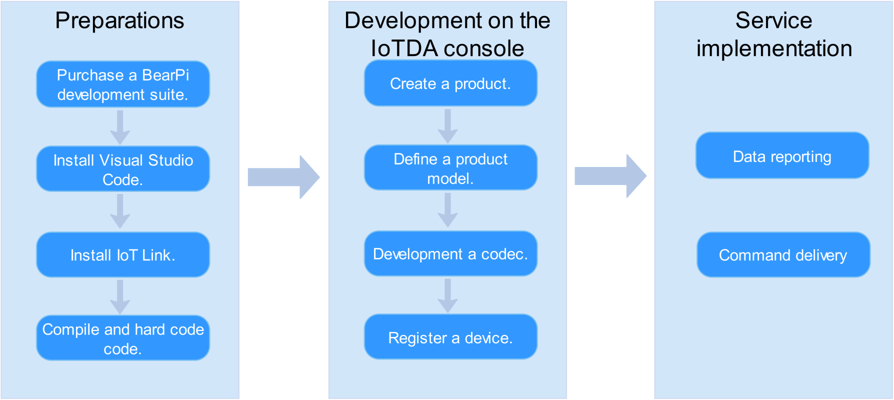

Deliver commands to turn on/off the street lamp.
IoT platform
Report illumination intensity.

Development Environment

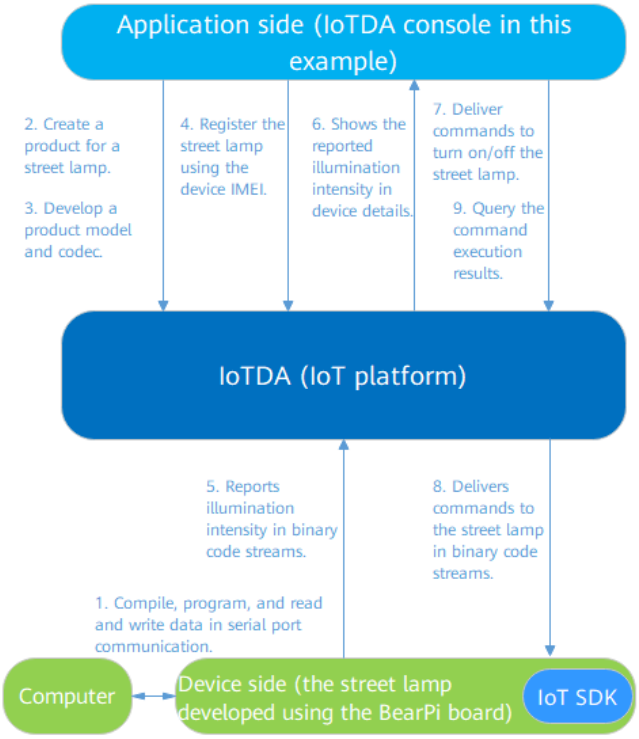
- Hardware: BearPi-IoT development suite (including NB-IoT cards, NB-IoT modules, smart street lamp function modules, and USB data cables)
- Software: Visual Studio Code, the IoT Link plug-in, Huawei Cloud **IoTDA** service, and 64-bit Windows 7 or later (64-bit Windows 10 is used in the following demonstration.)

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.


Application side (IoTDA console in this example)
2. Create a product for a street lamp.
3. Develop a product model and codec.
4. Register the street lamp using the device IMEI.
6. Shows the reported illumination intensity in device details.
7. Deliver commands to turn on/off the street lamp.
9. Query the command execution results.
IoTDA (IoT platform)
5. Reports illumination intensity in binary code streams.
8. Delivers commands to the street lamp in binary code streams.
1. Compile, program, and read and write data in serial port communication.
Computer
Device side (the street lamp developed using the BearPi board)
IoT SDK

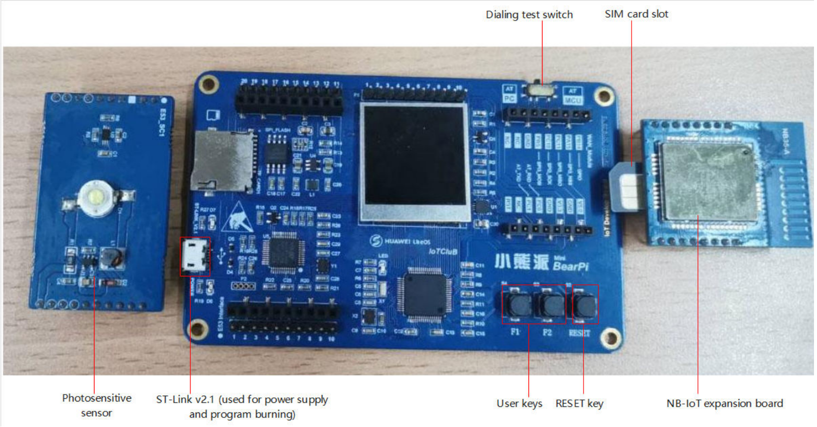
Introduction to the BearPi Development Board

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.

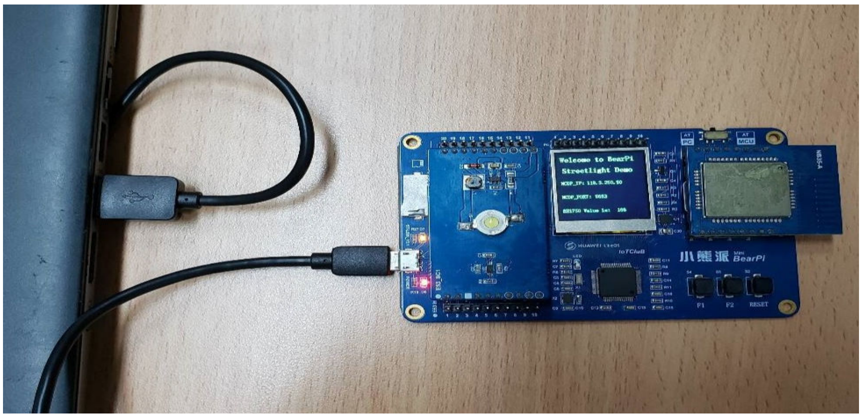
There is a DIP switch in the upper right corner of the development board. When you set the switch to the AT-PC mode, use the serial port assistant on the computer to send AT commands to debug the communication module. When you set it to the AT-MCU mode, use the MCU to send AT commands to interact with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

Figure 2-2 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 1**.
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.

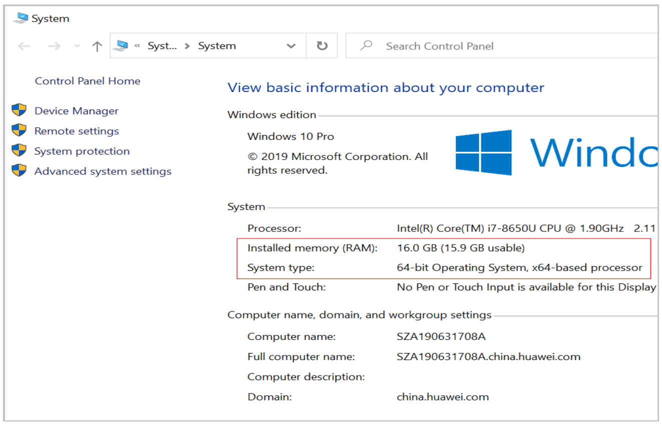


Installing the IoT Link Studio Plug-in

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 2-3 Obtaining the system configuration



View basic information about your computer

Windows edition

Windows 10 Pro

© 2019 Microsoft Corporation. All rights reserved.

System

Processor: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11

Installed memory (RAM): 16.0 GB (15.9 GB usable)

System type: 64-bit Operating System, x64-based processor

Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name: SZA190631708A

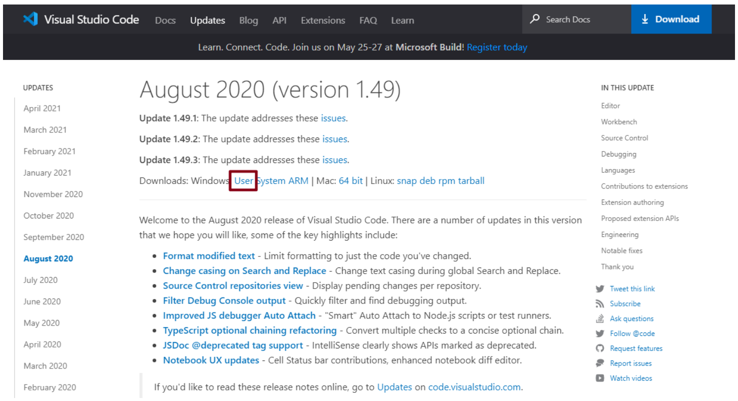
Full computer name: SZA190631708A.china.huawei.com

Computer description:

Domain: china.huawei.com

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 2-4 Downloading Visual Studio Code



Learn. Connect. Code. Join us on May 25-27 at **Microsoft Build!** Register today

August 2020 (version 1.49)

Update 1.49.1: The update addresses these issues.

Update 1.49.2: The update addresses these issues.

Update 1.49.3: The update addresses these issues.

Downloads: Windows: User System ARM | Mac: 64 bit | Linux: snap deb rpm tarball

Welcome to the August 2020 release of Visual Studio Code. There are a number of updates in this version that we hope you will like, some of the key highlights include:

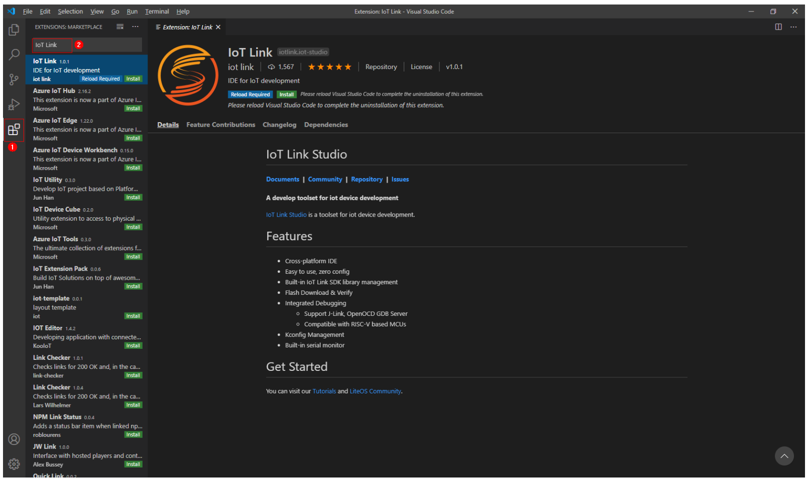
- **Format modified text** - Limit formatting to just the code you've changed.
- **Change casing on Search and Replace** - Change text casing during global Search and Replace.
- **Source Control repositories view** - Display pending changes per repository.
- **Filter Debug Console output** - Quickly filter and find debugging output.
- **Improved JS debugger Auto Attach** - "Smart" Auto Attach to Node.js scripts or test runners.
- **TypeScript optional chaining refactoring** - Convert multiple checks to a concise optional chain.
- **JSDoc @deprecated tag support** - IntelliSense clearly shows APIs marked as deprecated.
- **Notebook UX updates** - Cell Status bar contributions, enhanced notebook diff editor.

If you'd like to read these release notes online, go to Updates on code.visualstudio.com.

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.

Figure 2-5 Installation



IoT Link 1.0.1
IDE for IoT development
iot link Reload Required Install

Azure IoT Hub 2.16.2
This extension is now a part of Azure I...
Microsoft Install

Azure IoT Edge 1.22.0
This extension is now a part of Azure I...
Microsoft Install

Azure IoT Device Workbench 0.15.0
This extension is now a part of Azure I...
Microsoft Install

IoT Utility 0.3.0
Develop IoT project based on Platfor...
Jun Han Install

IoT Device Cube 0.2.0
Utility extension to access to physical ...
Microsoft Install

Azure IoT Tools 0.3.0
The ultimate collection of extensions f...
Microsoft Install

IoT Extension Pack 0.0.6
Build IoT Solutions on top of awesom...
Jun Han Install

iot-template 0.0.1
layout template
iot Install

IOT Editor 1.4.2
Developing application with connecte...
KooIoT Install

Link Checker 1.0.1
Checks links for 200 OK and, in the ca...
link-checker Install

Link Checker 1.0.4
Checks links for 200 OK and, in the ca...
Lars Wilhelmer Install

NPM Link Status 0.0.4
Adds a status bar item when linked np...
roblourens Install

JW Link 1.0.0
Interface with hosted players and cont...
Alex Bussey Install

Quick Link 0.0.2

IoT Link iotlink.iot-studio
iot link | 1,567 | ★★★★★ | Repository | License | v1.0.1
IDE for IoT development
Reload Required Install *Please reload Visual Studio Code to complete the uninstallation of this extension.*
Please reload Visual Studio Code to complete the uninstallation of this extension.

IoT Link Studio

Documents | Community | Repository | Issues

A develop toolset for iot device development

IoT Link Studio is a toolset for iot device development.

Features

- Cross-platform IDE
- Easy to use, zero config
- Built-in IoT Link SDK library management
- Flash Download & Verify
- Integrated Debugging
  - Support J-Link, OpenOCD GDB Server
  - Compatible with RISC-V based MCUs
- Kconfig Management
- Built-in serial monitor

Get Started

You can visit our Tutorials and LiteOS Community.

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

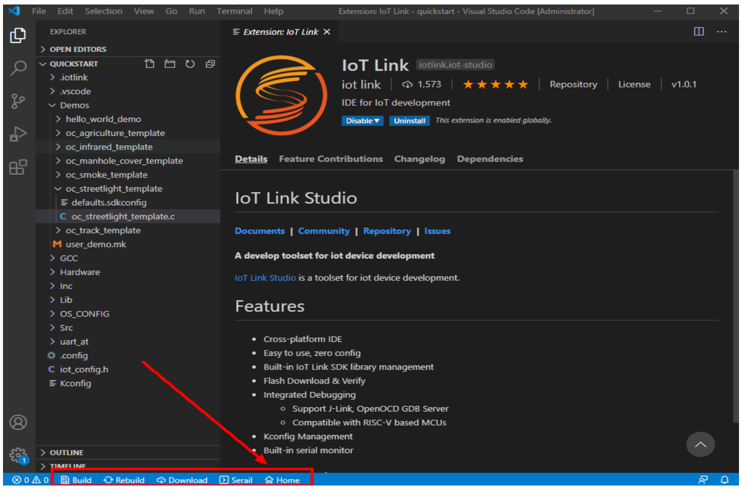
If a proxy is required, click ⚙ in the lower left corner of the Visual Studio Code home page and choose **Settings** > **Application** > **Proxy**, and set **Use the proxy support for extensions** to **on**.

----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 2**).
- **Download** is used to hard code to the MCU (displayed after **Step 2**).


File Edit Selection View Go Run Terminal Help
Extension: IoT Link - quickstart - Visual Studio Code [Administrator]
EXPLORER
OPEN EDITORS
QUICKSTART
.iotlink
.vscode
Demos
hello_world_demo
oc_agriculture_template
oc_infrared_template
oc_manhole_cover_template
oc_smoke_template
oc_streetlight_template
defaults.sdkconfig
oc_streetlight_template.c
oc_track_template
user_demo.mk
GCC
Hardware
Inc
Lib
OS_CONFIG
Src
uart_at
.config
iot_config.h
Kconfig
OUTLINE
TIMELINE
Extension: IoT Link
IoT Link iotlink.iot-studio
iot link | 1,573 | ★★★★★ | Repository | License | v1.0.1
IDE for IoT development
Disable Uninstall This extension is enabled globally.
Details Feature Contributions Changelog Dependencies
Build Rebuild Download Serail Home

IoT Link Studio

Documents | Community | Repository | Issues

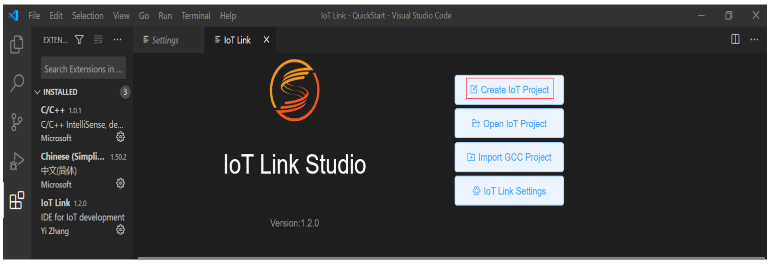
A develop toolset for iot device development

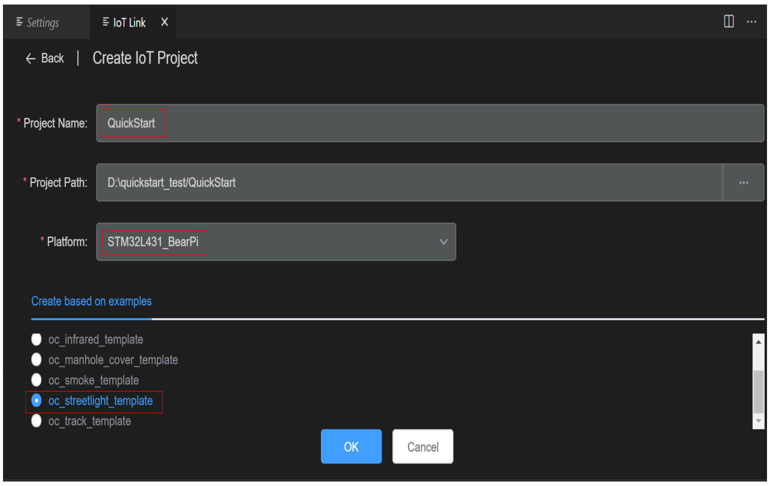
IoT Link Studio is a toolset for iot device development.

Features

- Cross-platform IDE
- Easy to use, zero config
- Built-in IoT Link SDK library management
- Flash Download & Verify
- Integrated Debugging
  - Support J-Link, OpenOCD GDB Server
  - Compatible with RISC-V based MCUs
- Kconfig Management
- Built-in serial monitor

Step 2 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: QuickStart

* Project Path: D:\quickstart_test/QuickStart

* Platform: STM32L431_BearPi

○ oc_infrared_template
○ oc_manhole_cover_template
○ oc_smoke_template
● oc_streetlight_template
○ oc_track_template

- **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.

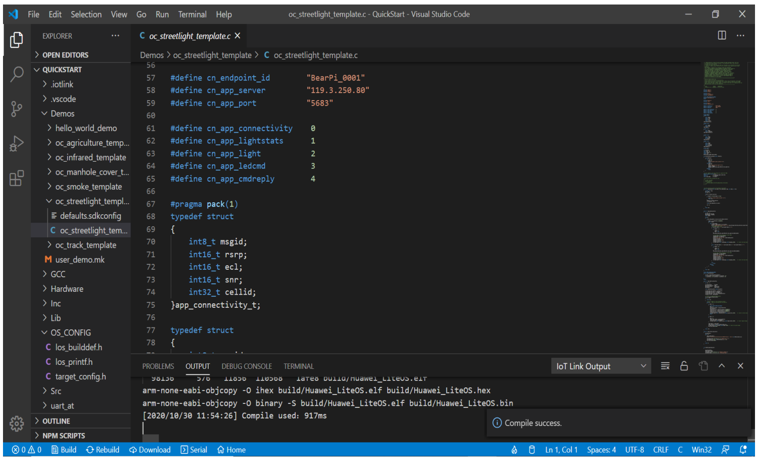
Step 3 Click **OK**.

----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.



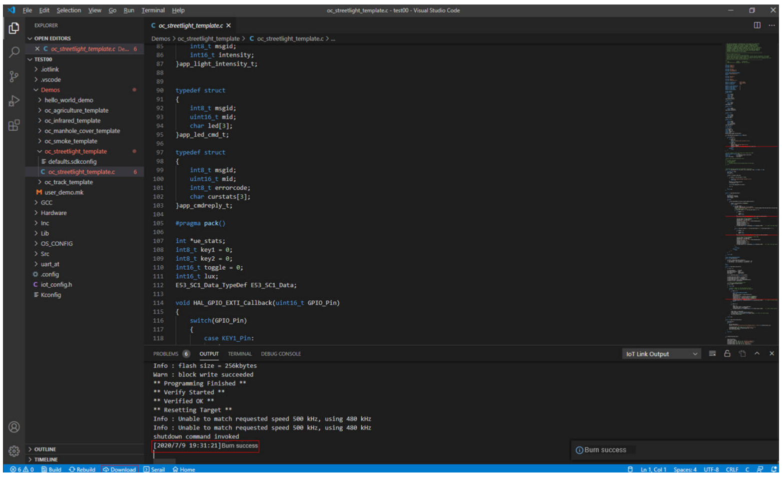
```
#define cn_endpoint_id        "BearPi_0001"
#define cn_app_server         "119.3.250.80"
#define cn_app_port           "5683"

#define cn_app_connectivity    0
#define cn_app_lightstats      1
#define cn_app_light           2
#define cn_app_ledcmd          3
#define cn_app_cmdreply        4

#pragma pack(1)
typedef struct
{
    int8_t msgid;
    int16_t rsrp;
    int16_t ecl;
    int16_t snr;
    int32_t cellid;
}app_connectivity_t;

typedef struct
{
```

```
arm-none-eabi-objcopy -O ihex build/Huawei_LiteOS.elf build/Huawei_LiteOS.hex
arm-none-eabi-objcopy -O binary -S build/Huawei_LiteOS.elf build/Huawei_LiteOS.bin
[2020/10/30 11:54:26] Compile used: 917ms
```

Compile success.

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.



```
    int8_t msgid;
    int16_t intensity;
}app_light_intensity_t;


typedef struct
{
    int8_t msgid;
    uint16_t mid;
    char led[3];
}app_led_cmd_t;

typedef struct
{
    int8_t msgid;
    uint16_t mid;
    int8_t errorcode;
    char curstats[3];
}app_cmdreply_t;

#pragma pack()

int *ue_stats;
int8_t key1 = 0;
int8_t key2 = 0;
int16_t toggle = 0;
int16_t lux;
E53_SC1_Data_TypeDef E53_SC1_Data;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch(GPIO_Pin)
    {
        case KEY1_Pin:
```

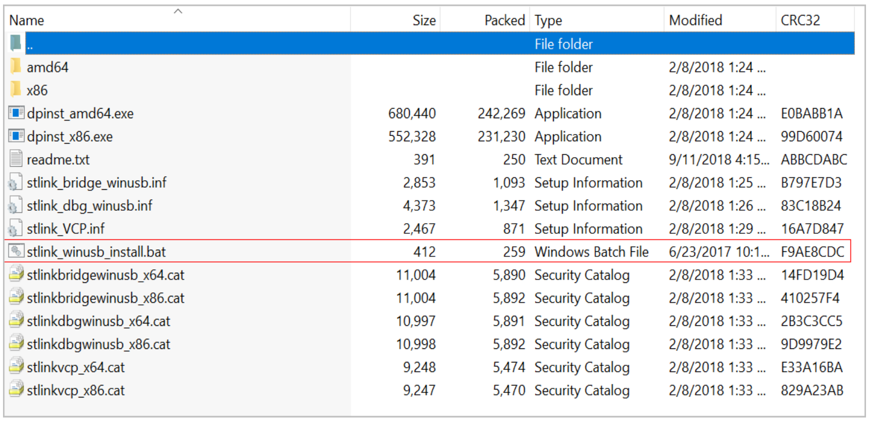
```
Info : flash size = 256kbytes
Warn : block write succeeded
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
shutdown command invoked
[2020/7/9 19:31:21]Burn success
```

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.

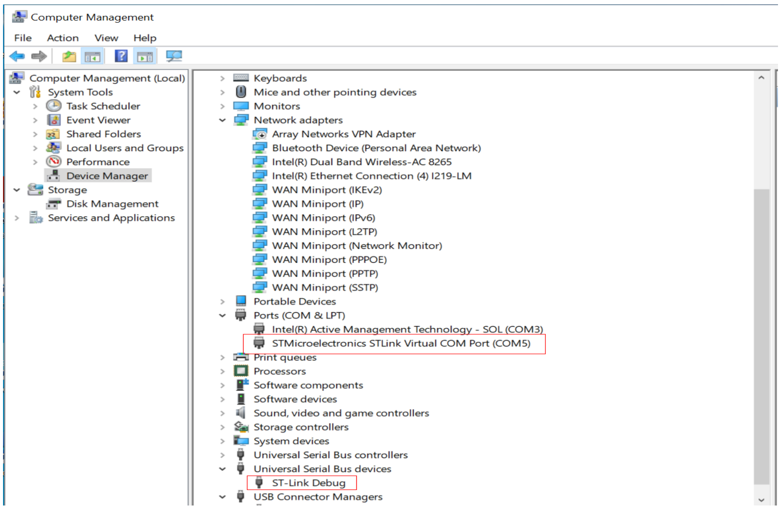
1. Visit the **ST website**, download the ST-Link driver, and double-click the **stlink_winusb_install.bat** file to start automatic installation. This section uses Windows 10 64-bit ST-Link 2.0.1 as an example.



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 ...	2B3C3CC5
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.

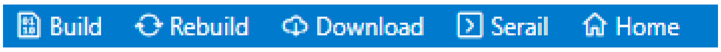


----End

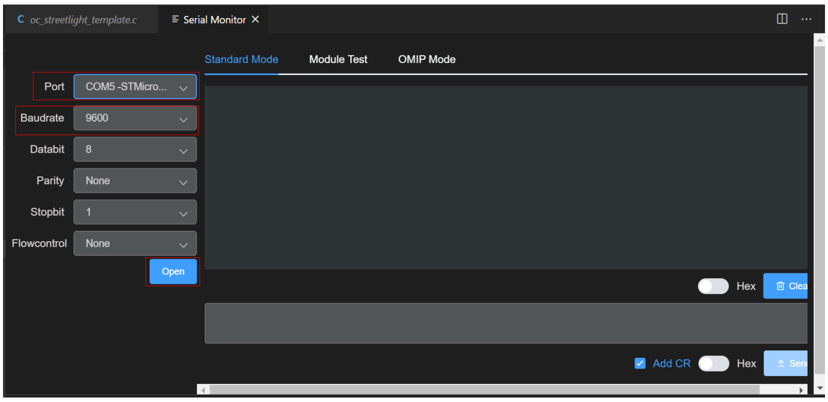
Locating Module Communication Problems Using AT Commands

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.

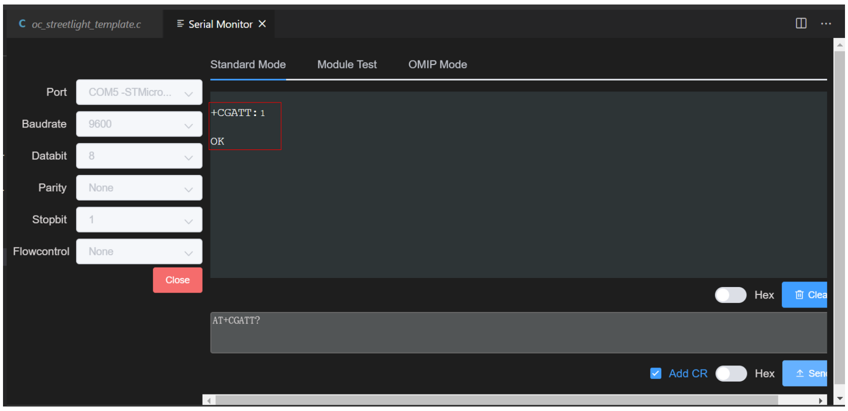


3. Select the port number obtained in 2, set **Baudrate** to **9600**, and click **Open**.



Port COM5 -STMicro...
Baudrate 9600
Databit 8
Parity None
Stopbit 1
Flowcontrol None

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.



Port: COM5 -STMicro...
Baudrate: 9600
Databit: 8
Parity: None
Stopbit: 1
Flowcontrol: None

```
+CGATT:1

OK
```

```
AT+CGATT?
```

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 (profiles), and codecs generated during IoT capability building. In this example, create a smart street light product on the IoTDA console.

Step 1 Visit the **IoTDA** product page and click **Access Console**.

Step 2 In the navigation pane, click **Products**. Click the drop-down list in the upper right corner, and select the resource space to which the new product belongs.

Step 3 Click **Create Product** in the upper right corner to create a product using CoAP. Set parameters and click **OK**.

Basic Information	
Resource Space	Select the resource space to which the product belongs.
Product Name	Enter a name, for example, **BearPi_StreetLight**.
Protocol	Select **LwM2M over CoAP**.
Data Type	Select **Binary**.
Manufacturer	Enter a name, for example, **BearPi**.
Industry	Select **Default**.
Device Type	Enter **StreetLight**.

Step 4 After the product is created, click the product to access its details and perform subsequent operations.

----End

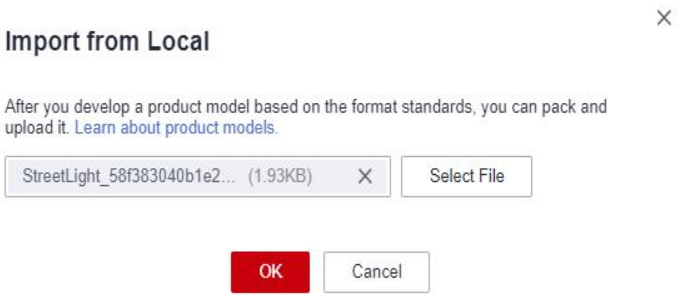
Uploading a Product Model

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

1. On the **Model Definition** tab page of the product details page, click **Import from Local**.
2. On the dialogue box displayed, upload the **product model provided** and click **OK**.

Figure 2-6 Uploading a model file



Import from Local

After you develop a product model based on the format standards, you can pack and upload it. Learn about product models.

StreetLight_58f383040b1e2... (1.93KB)

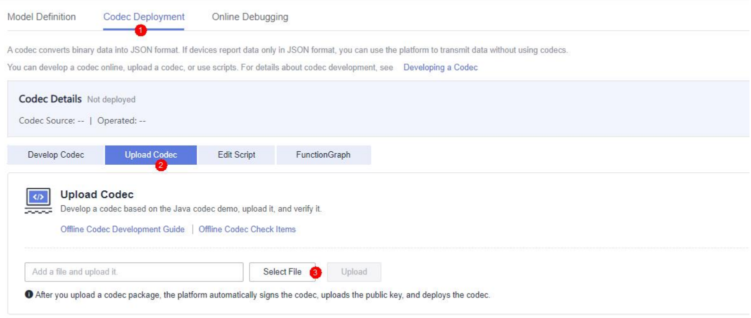
Developing a Codec

In the previous step, we have defined the functions of the product on the console, including the properties reported by the device and the commands that the device can understand. The data format of a smart street light is binary, and the data format of IoTDA is JSON. A codec is needed to enable IoTDA to understand the data reported by the smart street light and to enable the smart street light to understand the commands delivered by IoTDA.

This section describes how to import codec offline to help you quickly experience the cloud migration process. For details about how to develop a codec, see **Developing a Codec**.

Step 1 On the product details page, click the **Codec Development** tab page. Click **Upload Codec**, and upload the **codec provided**.

Figure 2-7 Uploading a codec



A codec converts binary data into JSON format. If devices report data only in JSON format, you can use the platform to transmit data without using codecs.

You can develop a codec online, upload a codec, or use scripts. For details about codec development, see Developing a Codec

Codec Details Not deployed

Codec Source: -- | Operated: --

Upload Codec

Develop a codec based on the Java codec demo, upload it, and verify it.

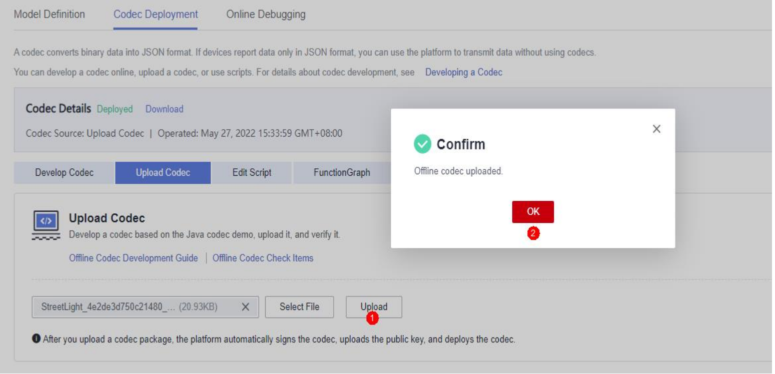
Offline Codec Development Guide | Offline Codec Check Items

Add a file and upload it.

After you upload a codec package, the platform automatically signs the codec, uploads the public key, and deploys the codec.

Step 2 Click **OK**.

Figure 2-8 Uploaded



A codec converts binary data into JSON format. If devices report data only in JSON format, you can use the platform to transmit data without using codecs.

You can develop a codec online, upload a codec, or use scripts. For details about codec development, see Developing a Codec

Codec Details Deployed

Codec Source: Upload Codec | Operated: May 27, 2022 15:33:59 GMT+08:00

Upload Codec

Develop a codec based on the Java codec demo, upload it, and verify it.

StreetLight_4e2de3d750c21480_... (20.93KB)

After you upload a codec package, the platform automatically signs the codec, uploads the public key, and deploys the codec.

Confirm

Offline codec uploaded.

----End

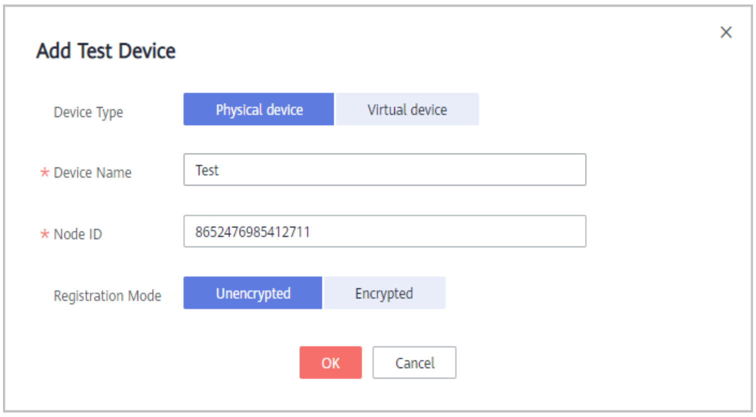
Registering a Device

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 **Add Test Device** on the **Online Debugging** tab page

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

Figure 2-9 Adding a test device



Add Test Device

Device Type: Physical device

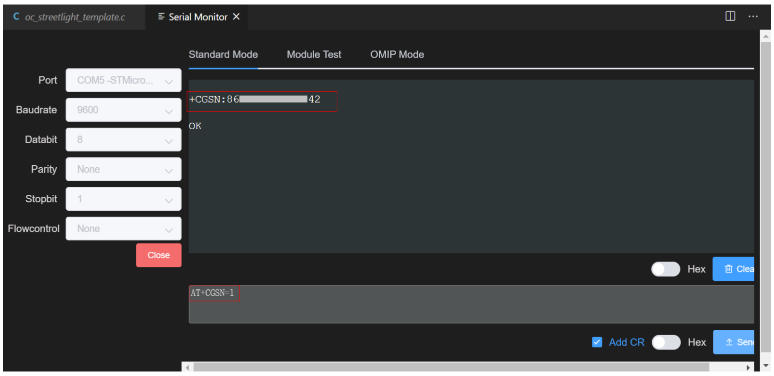
* Device Name: Test

* Node ID: 8652476985412711

Registration Mode: Unencrypted

- **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.



Port: COM5 -STMicro...
Baudrate: 9600
Databit: 8
Parity: None
Stopbit: 1
Flowcontrol: None

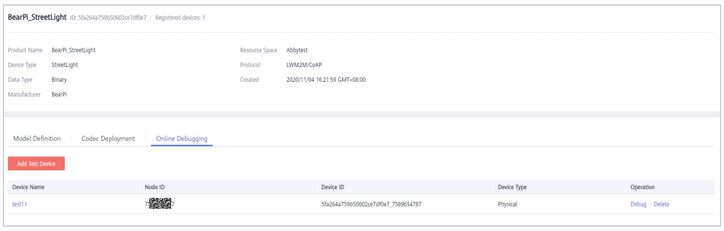
```
+CGSN:86[redacted]42
OK
```

```
AT+CGSN=1
```

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

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

Figure 2-10 Viewing the created device



BearPi_StreetLight ID: 5fa264a759b50602ce7df0e7 | Registered devices: 1

Product Name: BearPi_StreetLight
Device Type: StreetLight
Data Type: Binary
Manufacturer: BearPi
Resource Space: Abbytest
Protocol: LWM2M/CoAP
Created: 2020/11/04 16:21:59 GMT+08:00

Device Name	Node ID	Device ID	Device Type	Operation
test11	7[redacted]7	5fa264a759b50602ce7df0e7_7589654787	Physical	Debug | Delete

----End

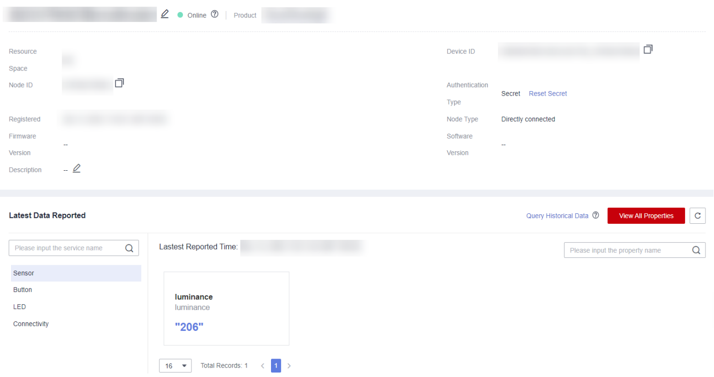
Data Reporting

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 choose **Devices** > **All Devices**.

Step 2 Select the device registered in **Registering a Device** and click **View** to view the data reported to the platform.



[redacted] Online | Product [redacted]

Resource Space: [redacted]
Node ID: [redacted]
Registered: [redacted]
Firmware Version: --
Description: --
Device ID: [redacted]
Authentication Type: Secret Reset Secret
Node Type: Directly connected
Software Version: --

Latest Data Reported

Sensor
Button
LED
Connectivity

Lastest Reported Time: [redacted]

luminance
luminance
"206"

16 Total Records: 1 1

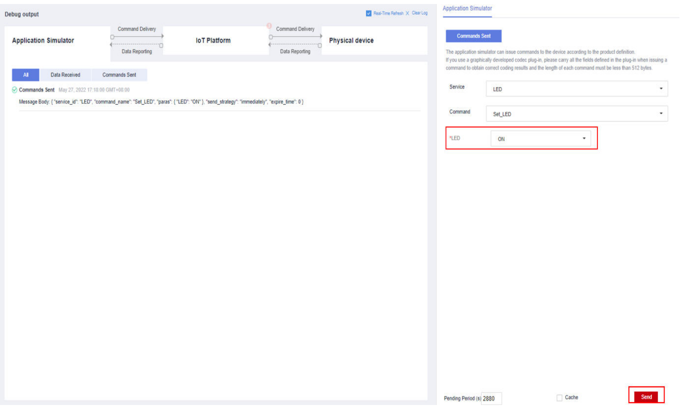
----End

Delivering a Command

Step 1 Log in to the **IoTDA console**. Click the product created in **Creating a Product** to go to the product details page.

Step 2 On the **Online Debugging** tab page, click the device registered in **Registering a Device** to access the debugging page.

Step 3 After setting the command parameters, click **Send**.



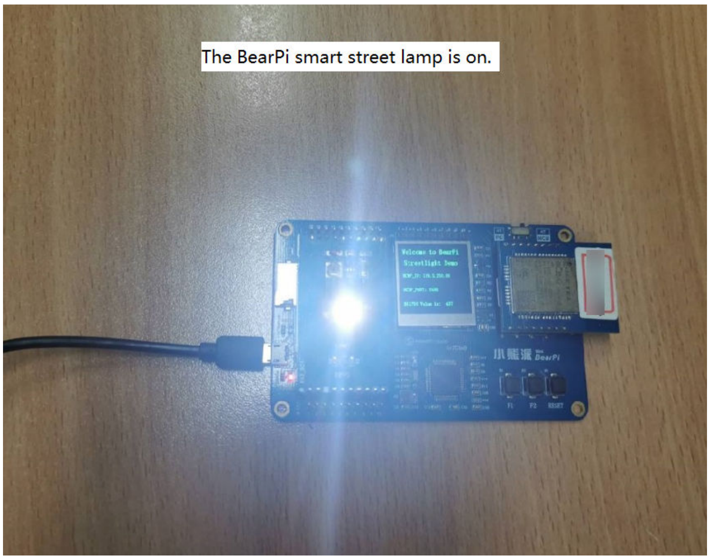
Commands Sent May 27, 2022 17:18:00 GMT+08:00

```
Message Body: { "service_id": "LED", "command_name": "Set_LED", "paras": { "LED": "ON" }, "send_strategy": "immediately", "expire_time": 0 }
```

The application simulator can issue commands to the device according to the product definition.
If you use a graphically developed codec plug-in, please carry all the fields defined in the plug-in when issuing a command to obtain correct coding results and the length of each command must be less than 512 bytes.

Service: LED
Command: Set_LED
*LED: ON
Pending Period (s): 2880

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 **Model Definition** tab page of the product details page, click **Custom Model** to add services of the product.

  Table 2-1 describes the service defined in the product model.

  Table 2-1 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 2-2 Button

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

Table 2-3 LED command list

Capability Description	Command Name	Command Parameter	Parameter Name	Data Type	Data Length	Enumeration
Commands	Set_LED	Command	**LED**	string	3	ON,OFF
		Response	**Light_state**	string	3	ON,OFF

Table 2-4 Sensor

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

Table 2-5 Connectivity

Capability Description	Property Name	Data Type	Data Range
Properties	SignalPower	int	-140 to -44
	ECL	int	0 to 2
	SNR	int	-20 to 30
	CellID	int	0 to 65535

– **Adding the Button Service**

i. On the **Add Service** page, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.

○ **Service ID**: Enter **Button**.

○ **Service Type**: You are advised to set this parameter to the same value as **Service ID**.

○ **Description**: Enter **Real-Time button detection**.

ii. Choose **Button**, click **Add Property**, enter related information, and click **OK**.

○ **Property Name**: Enter **toggle**.

○ **Description**: Leave it blank.

○ **Data Type**: Select **Integer**.

○ **Access Permissions**: Select **Read** and **Write**.

- **Value Range**: Set it to 0–65535.
- **Step**: Enter **0**.
- **Unit**: Leave it blank.

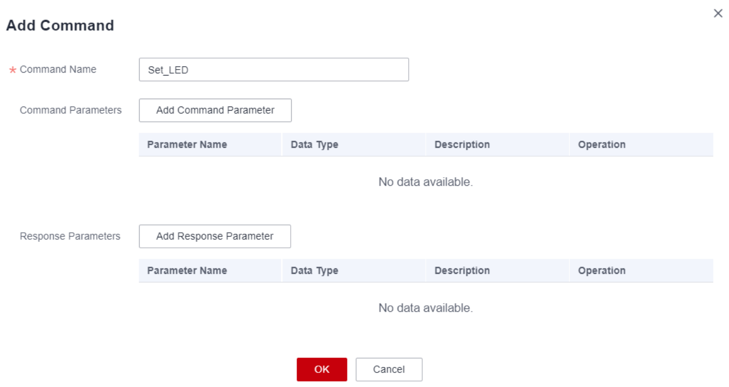
– **Adding the LED Service**

i. On the **Model Definition** tab page, click **Add Service**, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.

- **Service ID**: Enter **LED**.
- **Service Type**: You are advised to set this parameter to the same value as **Service ID**.
- **Description**: Enter **LED control**.

ii. Choose **LED**, click **Add Command**, and enter the command name **Set_LED**.

Figure 2-11 Adding a command



Add Command

* Command Name: Set_LED

Command Parameters

Parameter Name	Data Type	Description	Operation

No data available.

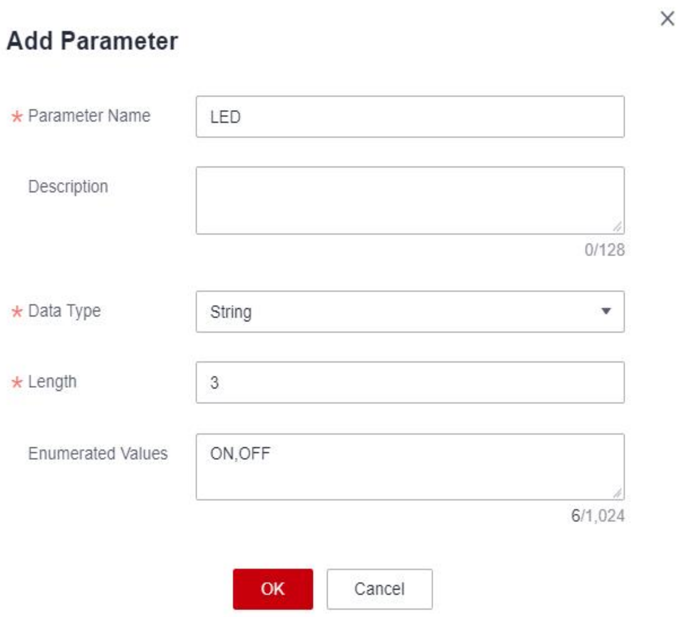
Response Parameters

Parameter Name	Data Type	Description	Operation

No data available.

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

Figure 2-12 Adding the input parameter LED



Add Parameter

* Parameter Name: LED

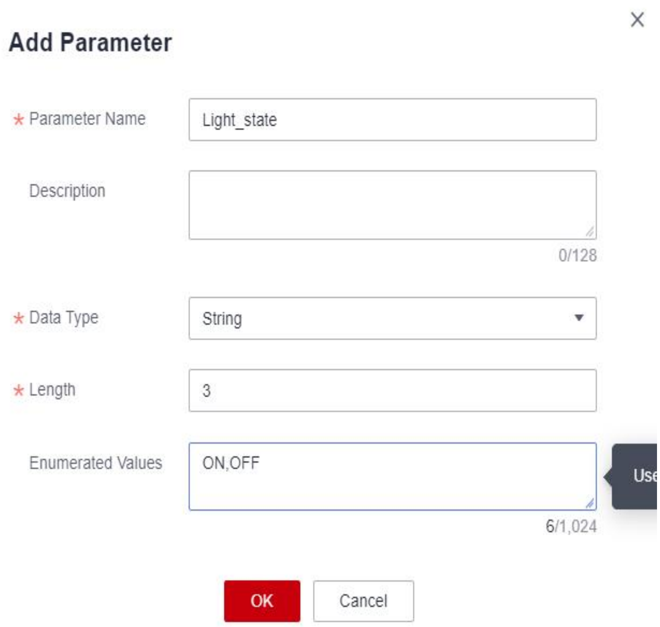
Description: 0/128

* Data Type: String

* Length: 3

Enumerated Values: ON,OFF 6/1,024

Figure 2-13 Adding the response parameter Light_state



Add Parameter

* Parameter Name: Light_state

Description:

0/128

* Data Type: String

* Length: 3

Enumerated Values: ON,OFF

6/1,024

- **Adding the Sensor Service**

  i. On the **Model Definition** tab page, click **Add Service**, configure **Service ID**, **Service Type**, and **Description**, and click **OK**.

     - **Service ID**: Enter **Sensor**.
     - **Service Type**: Set this parameter to the same value as **Service ID**.
     - **Description**: Enter **Real-time light intensity detection**.

  ii. Choose **Sensor**, click **Add Property**, enter related information, and click **OK**.

     - **Property Name**: Enter **luminance**.
     - **Description**: Leave it blank.
     - **Data Type**: Select **Integer**.
     - **Access Permissions**: Select **Read** and **Write**.
     - **Value Range**: Set it to 0–65535.
     - **Step**: Leave it blank.
     - **Unit**: Enter **lux**.

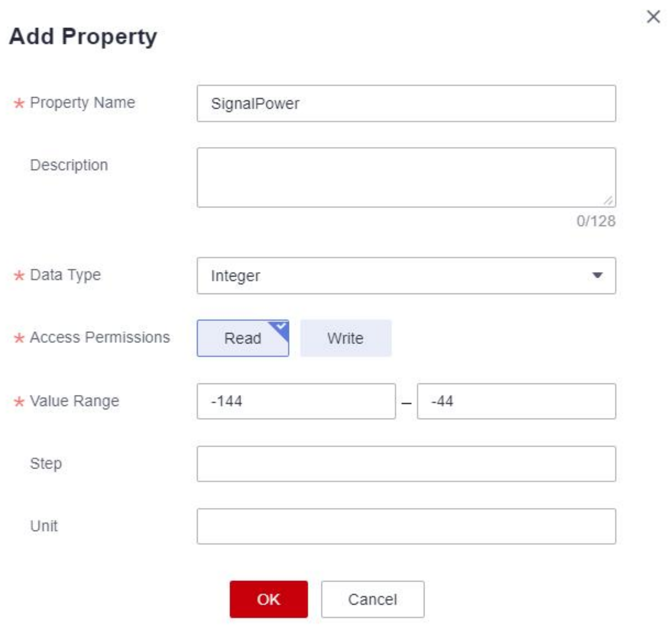
- **Adding the Connectivity Service**

  i. On the **Model Definition** tab page, click **Add Service**, 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 **Real-time signal quality detection**.

ii. Choose **Connectivity**, click **Add Property** to add **SignalPower**, **ECL**, **SNR**, and **CellID**, respectively. Enter related information, and click **OK**.

Figure 2-14 Adding the property SignalPower



Add Property

* Property Name: SignalPower

Description: 0/128

* Data Type: Integer

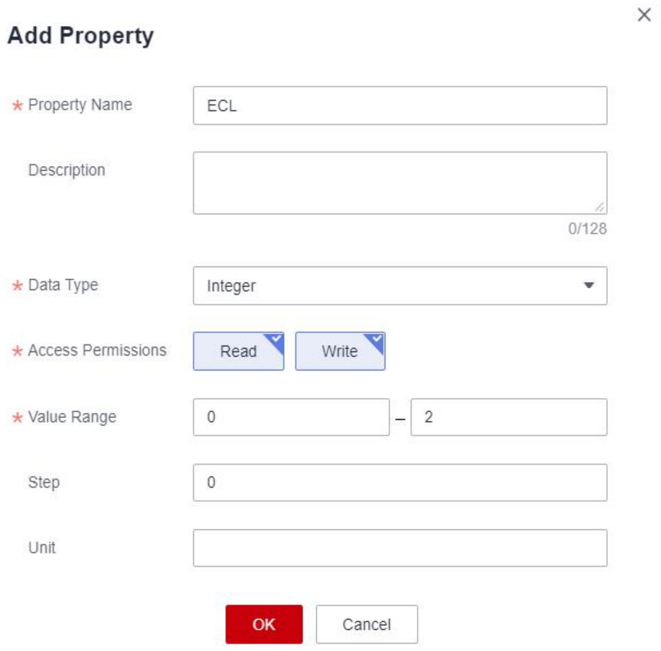
* Access Permissions: Read

* Value Range: -144 – -44

Step:

Unit:

Figure 2-15 Adding the property ECL



Add Property

* Property Name: ECL

Description: 0/128

* Data Type: Integer

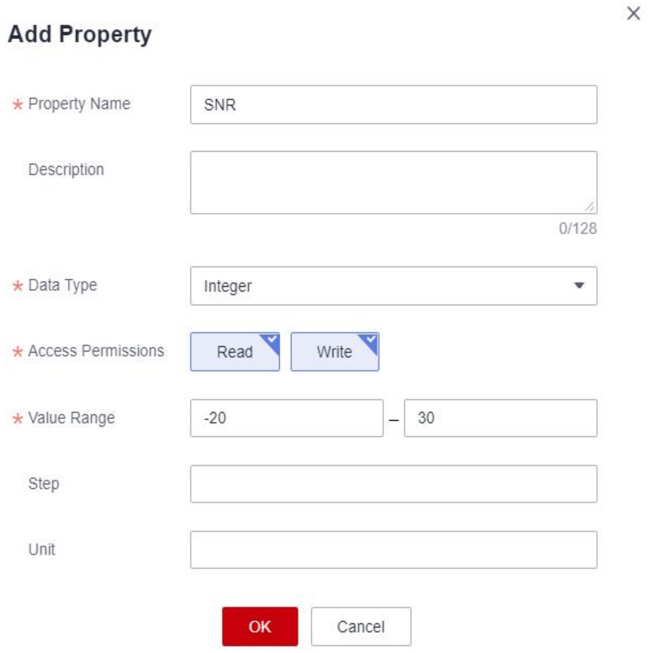
* Access Permissions: Read Write

* Value Range: 0 – 2

Step: 0

Unit:

Figure 2-16 Adding the property SNR



Add Property

* Property Name: SNR

Description:

0/128

* Data Type: Integer

* Access Permissions: Read Write

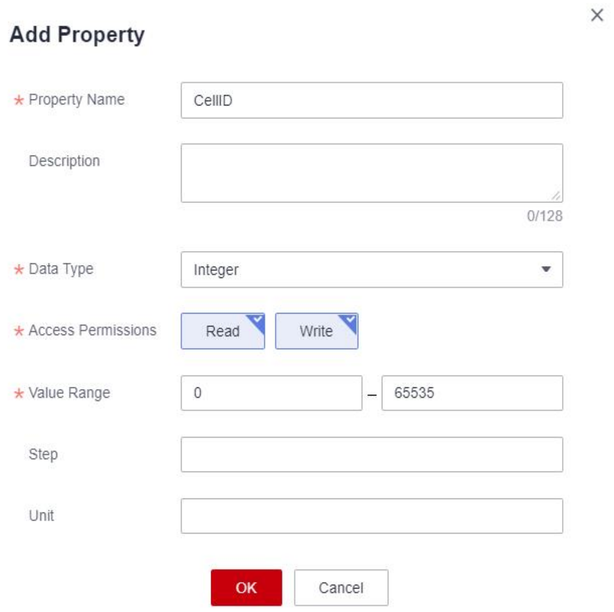
* Value Range: -20 – 30

Step:

Unit:

OK Cancel

Figure 2-17 Adding the property CellID



Add Property

* Property Name: CellID

Description:

0/128

* Data Type: Integer

* Access Permissions: Read Write

* Value Range: 0 – 65535

Step:

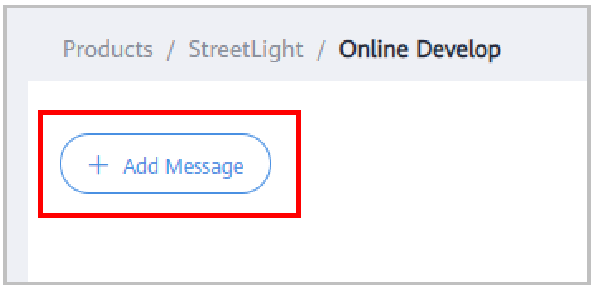
Unit:

OK Cancel

- **Developing a Codec**
  1. On the product details page, click **Develop Codec** on the **Develop Codec** tab page.
  2. Click **Add Message**.

     NOTE

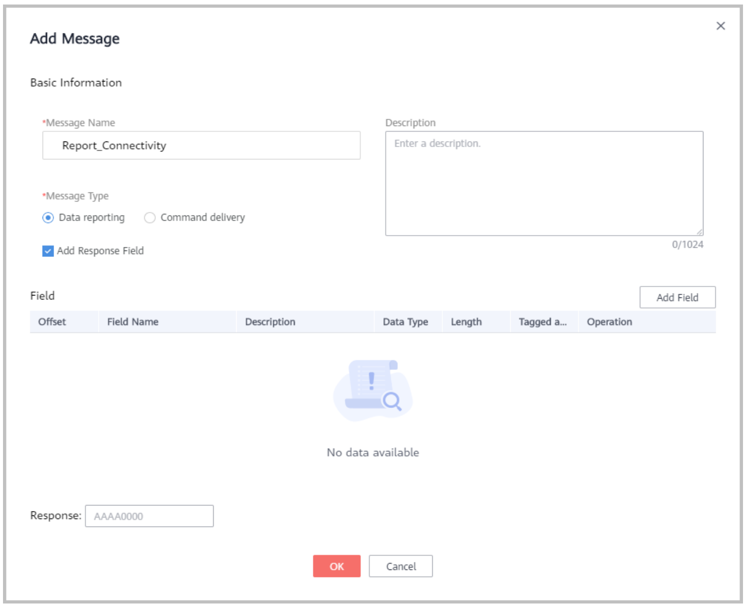
     Develop the codec based on the operations provided in this section and ensure that the fields are added in the order specified in this section.



c. Add the **Report_Connectivity** message. A configuration example is as follows:

- **Message Name**: Enter **Report_Connectivity**.
- **Message Type**: Select **Data reporting**.
- **Add Response Field**: Select this option.
- **Response**: Retain the default value **AAAA0000**.

Figure 2-18 Configuration example



*Message Name: Report_Connectivity

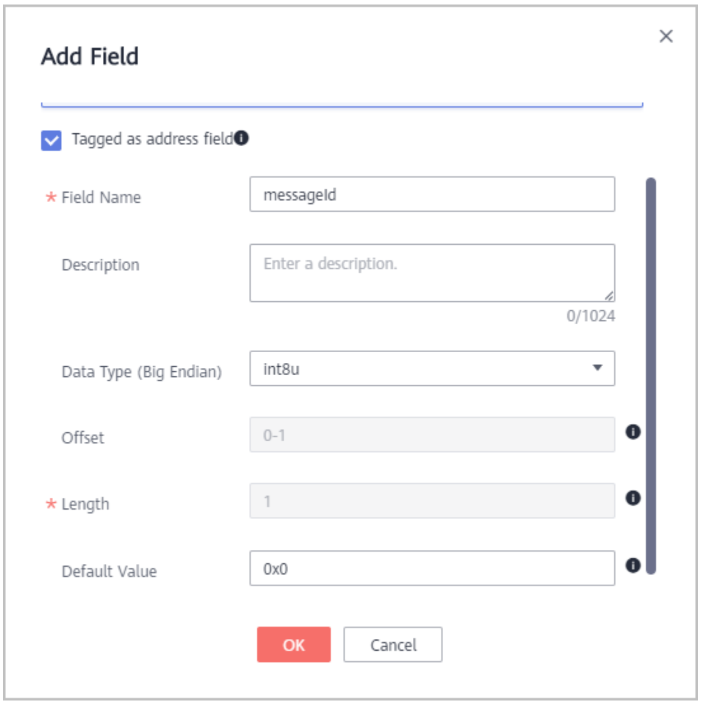
*Message Type: ● Data reporting ○ Command delivery

☑ Add Response Field

Response: AAAA0000

i. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.

Figure 2-19 Adding an address field



Add Field

☑ Tagged as address field

* Field Name: messageId

Description: Enter a description. 0/1024

Data Type (Big Endian): int8u

Offset: 0-1

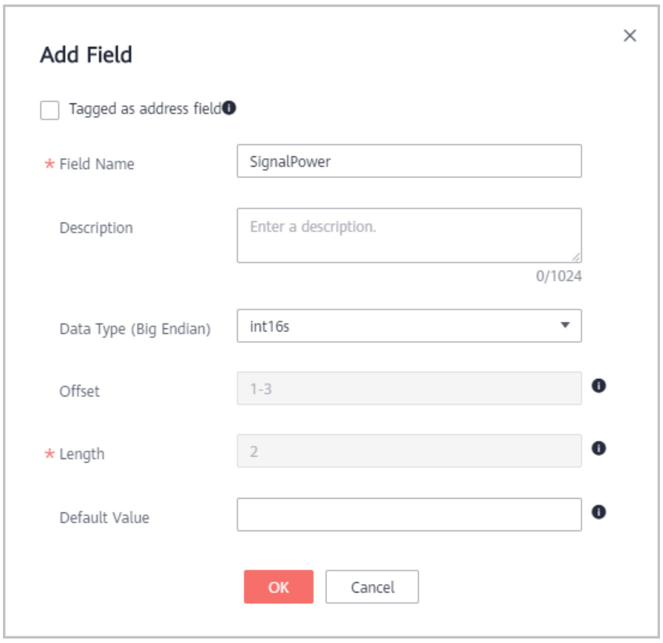
* Length: 1

Default Value: 0x0

ii. Click **Add Field**, add the **SignalPower** field, enter related information, and click **OK**.

- **Field Name**: Enter **SignalPower**.
- **Data Type**: Select **int16s**, which means a 16-bit signed integer.

Figure 2-20 Adding the SignalPower field



Add Field

□ Tagged as address field

* Field Name: SignalPower

Description: Enter a description. 0/1024

Data Type (Big Endian): int16s

Offset: 1-3

* Length: 2

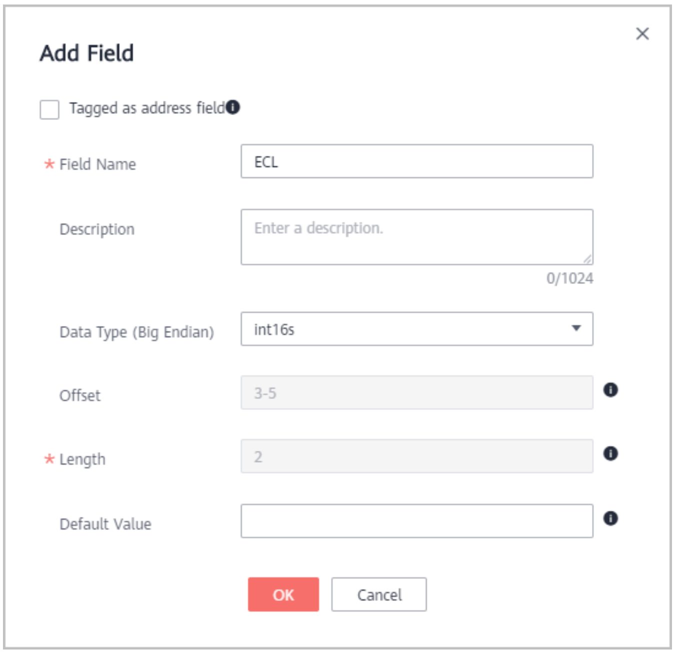
Default Value:

iii. Click **Add Field**, add the **ECL** field, enter related information, and click **OK**.

- **Field Name**: Enter **ECL**.

- ○ **Data Type**: Select **int16s**, which means a 16-bit signed integer.

Figure 2-21 Adding the ECL field



Add Field

□ Tagged as address field

* Field Name: ECL

Description: Enter a description. 0/1024

Data Type (Big Endian): int16s

Offset: 3-5

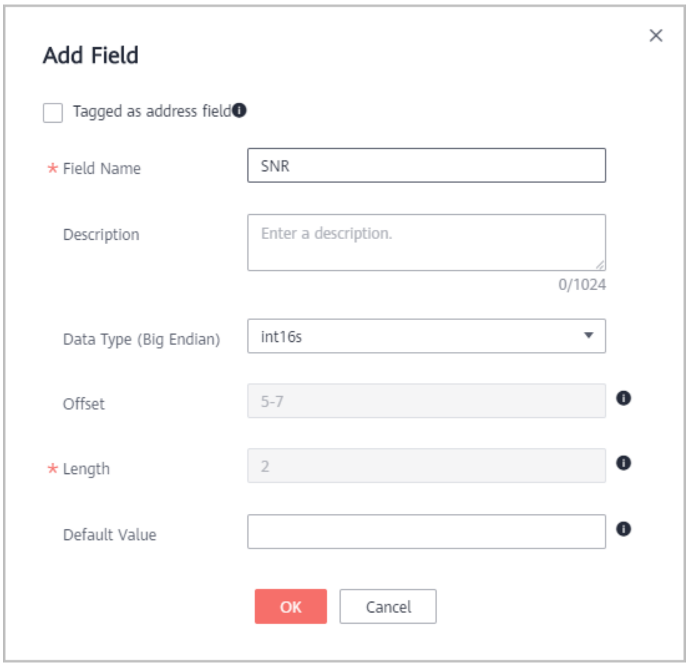
* Length: 2

Default Value:

iv. Click **Add Field**, add the **SNR** field, enter related information, and click **OK**.

- ○ **Field Name**: Enter **SNR**.
- ○ **Data Type**: Select **int16s**, which means a 16-bit signed integer.

Figure 2-22 Adding the SNR field



Add Field

□ Tagged as address field

* Field Name: SNR

Description: Enter a description. 0/1024

Data Type (Big Endian): int16s

Offset: 5-7

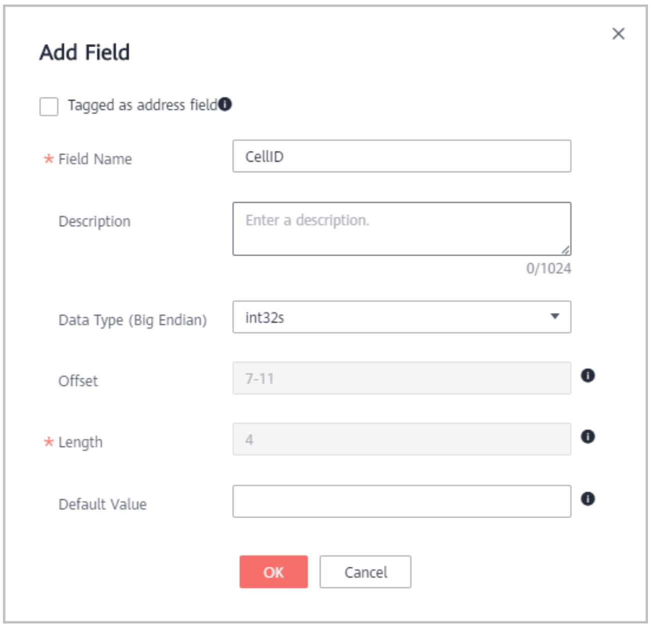
* Length: 2

Default Value:

v. Click **Add Field**, add the **CellID** field, enter related information, and click **OK**.

- ○ **Field Name**: Enter **CellID**.
- ○ **Data Type**: Select **int32s**, which means a 32-bit signed integer.

Figure 2-23 Adding the CellID field



Add Field

□ Tagged as address field

* Field Name: CellID

Description: Enter a description. 0/1024

Data Type (Big Endian): int32s

Offset: 7-11

* Length: 4

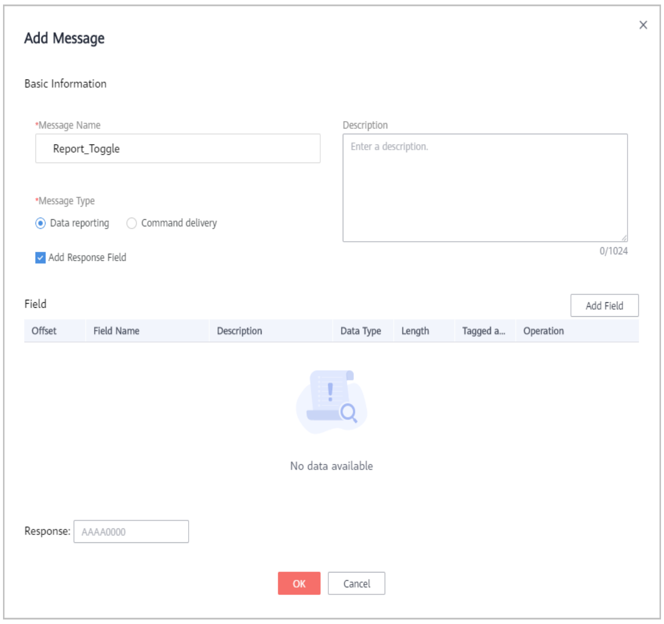
Default Value:

In the **Add Message** dialog box, click **OK** to complete the configuration of **Report_Connectivity**.

d. Add the **Report_Toggle** message. A configuration example is as follows:

- ■ **Message Name**: Enter **Report_Toggle**.
- ■ **Message Type**: Select **Data reporting**.
- ■ **Add Response Field**: Select this option.
- ■ **Response**: Retain the default value **AAAA0000**.

Figure 2-24 Adding a message



Basic Information

*Message Name: Report_Toggle

Description: Enter a description. 0/1024

*Message Type: ● Data reporting ○ Command delivery

☑ Add Response Field

Field

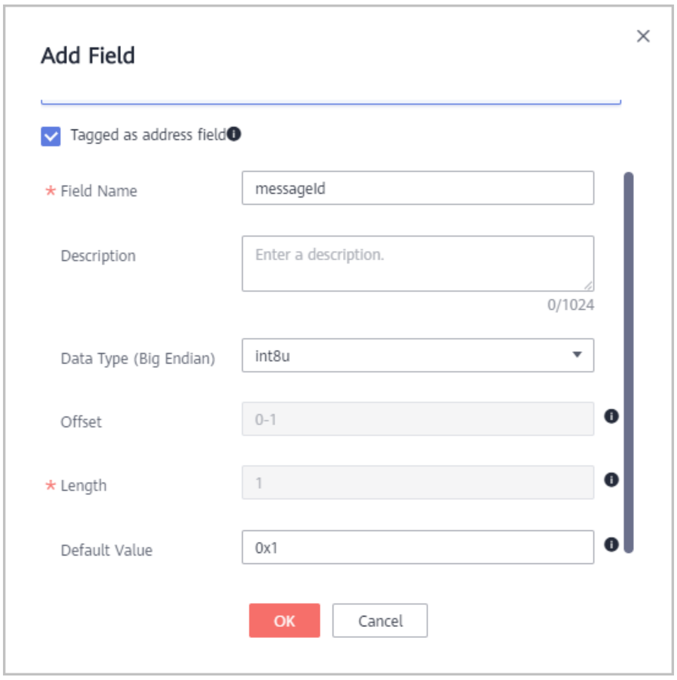
Offset	Field Name	Description	Data Type	Length	Tagged a...	Operation

No data available

Response: AAAA0000

i. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.

Figure 2-25 Adding a field



☑ Tagged as address field

* Field Name: messageId

Description: Enter a description. 0/1024

Data Type (Big Endian): int8u

Offset: 0-1

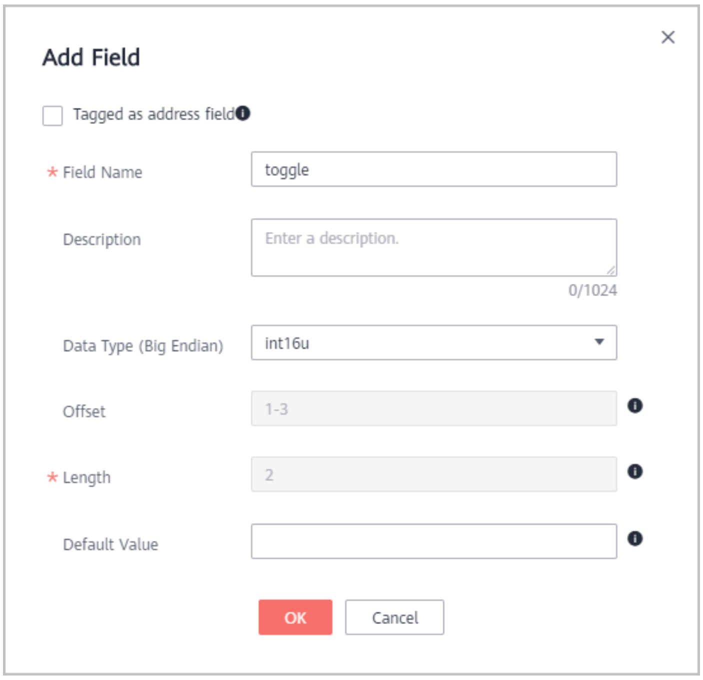
* Length: 1

Default Value: 0x1

ii. Click **Add Field**, add the **toggle** field, enter related information, and click **OK**.

- **Field Name**: Enter **toggle**.
- **Data Type**: Select **int16u**, which means a 16-bit unsigned integer.

Figure 2-26 Adding the toggle field



Add Field

□ Tagged as address field

* Field Name: toggle

Description: Enter a description. 0/1024

Data Type (Big Endian): int16u

Offset: 1-3

* Length: 2

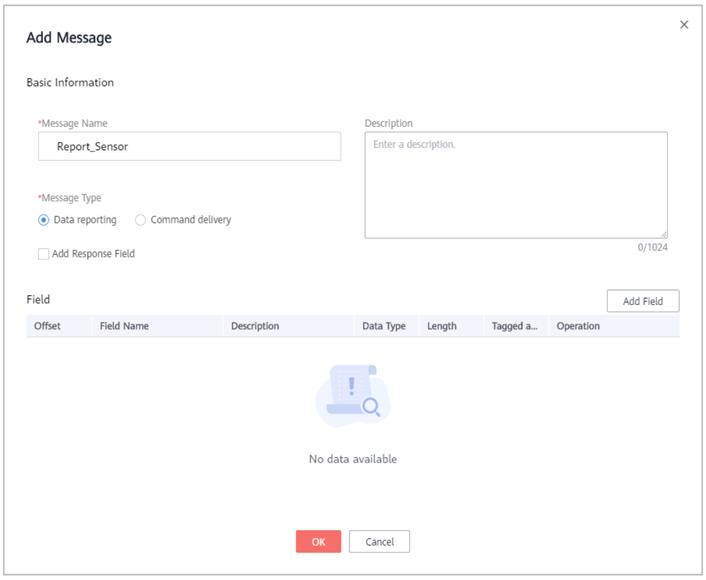
Default Value:

In the **Add Message** dialog box, click **OK** to complete the configuration of **Report_Toggle**.

e. Add the **Report_Sensor** message. A configuration example is as follows:

- **Message Name**: Enter **Report_Sensor**.
- **Message Type**: Select **Data reporting**.

Figure 2-27 Adding a message



Add Message

Basic Information

*Message Name: Report_Sensor

Description: Enter a description. 0/1024

*Message Type: ● Data reporting ○ Command delivery

□ Add Response Field

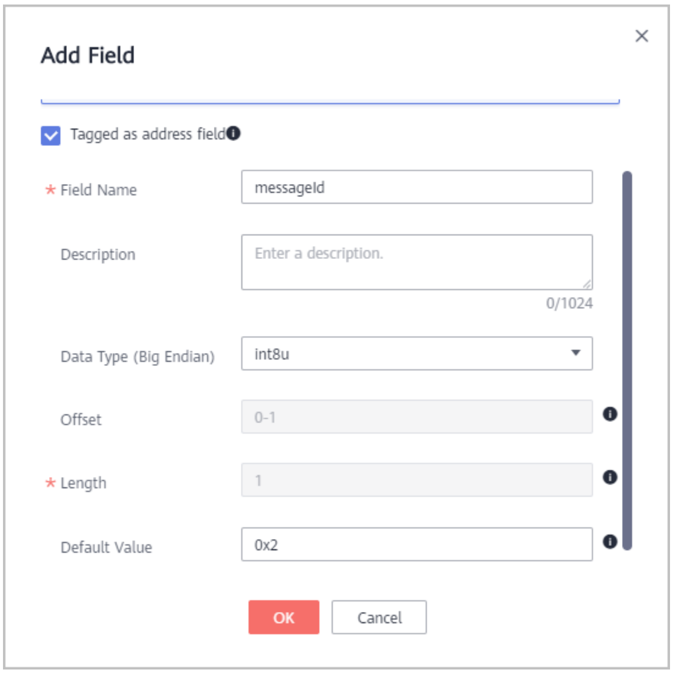
Field

Offset	Field Name	Description	Data Type	Length	Tagged a...	Operation

No data available

i. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.

Figure 2-28 Adding a field



Add Field

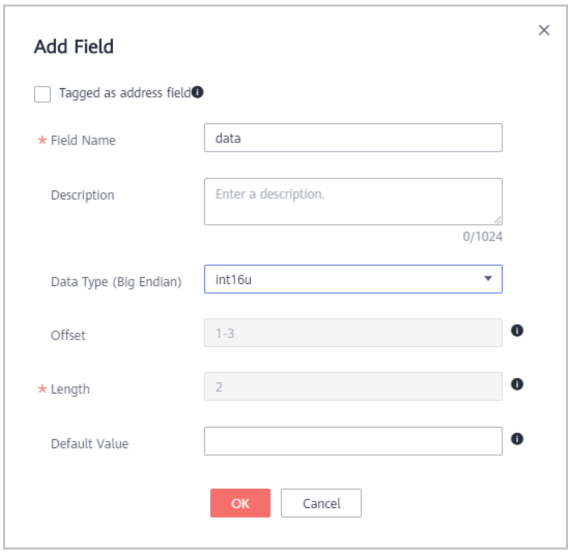
☑ Tagged as address field

* Field Name	messageId
Description	Enter a description. 0/1024
Data Type (Big Endian)	int8u
Offset	0-1
* Length	1
Default Value	0x2

ii. Click **Add Field**, add the **data** field, enter related information, and click **OK**.

- **Field Name**: Enter **data**.
- **Data Type**: Select **int16u**, which means a 16-bit unsigned integer.

Figure 2-29 Adding the data field



Add Field

□ Tagged as address field

* Field Name	data
Description	Enter a description. 0/1024
Data Type (Big Endian)	int16u
Offset	1-3
* Length	2
Default Value	

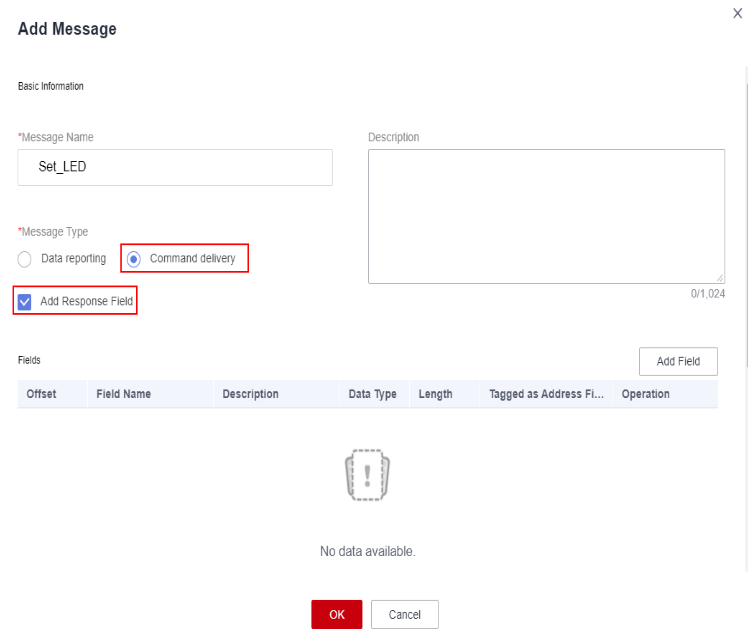
In the **Add Message** dialog box, click **OK** to complete the configuration of **Report_Sensor**.

f. Add the **Set_LED** message. A configuration example is as follows:

- **Message Name**: Enter **Set_LED**.

- **Message Type**: Select **Command delivery**.
- **Add Response Field**: Select this option.

Figure 2-30 Adding a message



Add Message

Basic Information

*Message Name: Set_LED

Description: 0/1,024

*Message Type: ○ Data reporting ● Command delivery

☑ Add Response Field

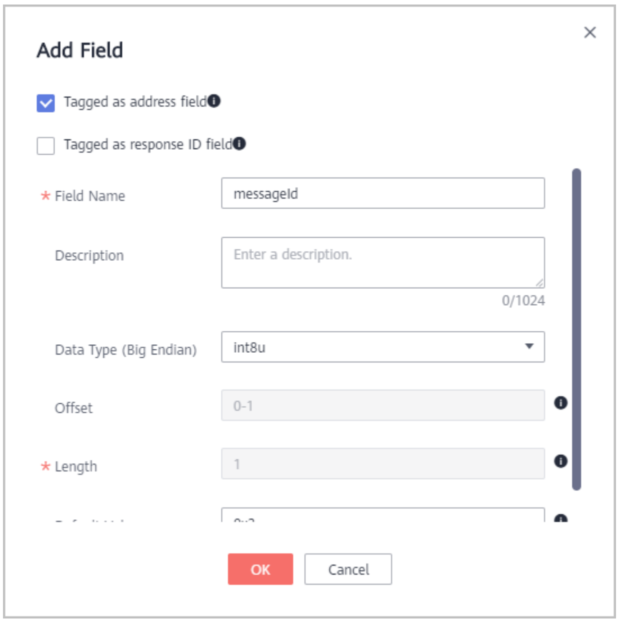
Fields

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

No data available.

i. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.

Figure 2-31 Adding a field



Add Field

☑ Tagged as address field

□ Tagged as response ID field

* Field Name: messageId

Description: Enter a description. 0/1024

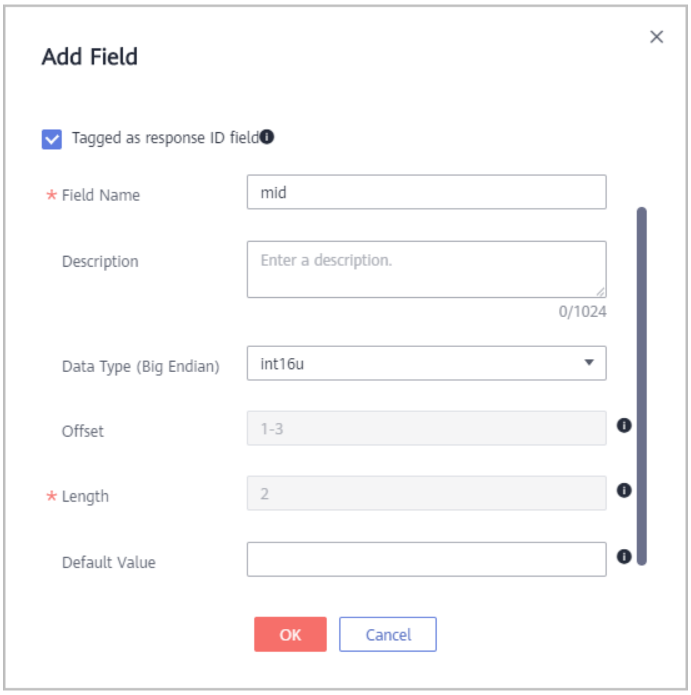
Data Type (Big Endian): int8u

Offset: 0-1

* Length: 1

ii. Click **Add Field**. In the dialog box displayed, select **Tagged as response ID field** to add the address field **mid**, and click **OK**.

Figure 2-32 Adding a field



Add Field

☑ Tagged as response ID field

* Field Name: mid

Description: Enter a description. 0/1024

Data Type (Big Endian): int16u

Offset: 1-3

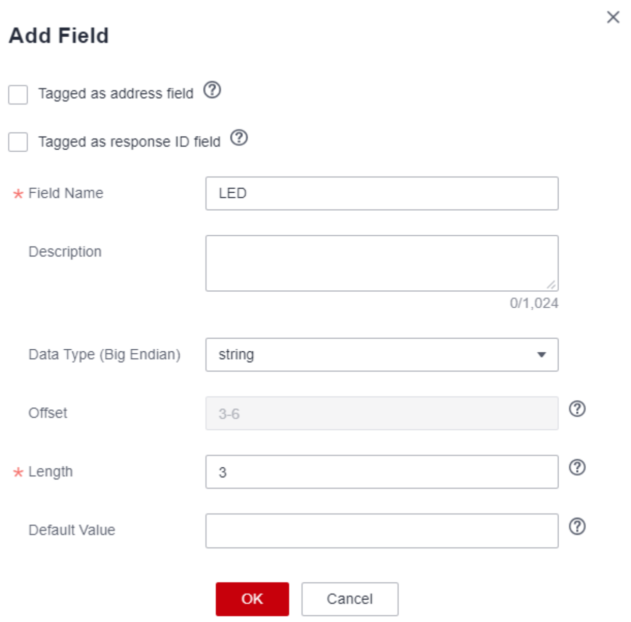
* Length: 2

Default Value:

iii. Click **Add Field**. In the dialog box displayed, add the **LED** field, and click **OK**.

- **Field Name**: Enter **LED**.
- **Data Type**: Select **string**.
- **Length**: Enter **3**.

Figure 2-33 Adding a field



Add Field

□ Tagged as address field

□ Tagged as response ID field

* Field Name: LED

Description: 0/1,024

Data Type (Big Endian): string

Offset: 3-6

* Length: 3

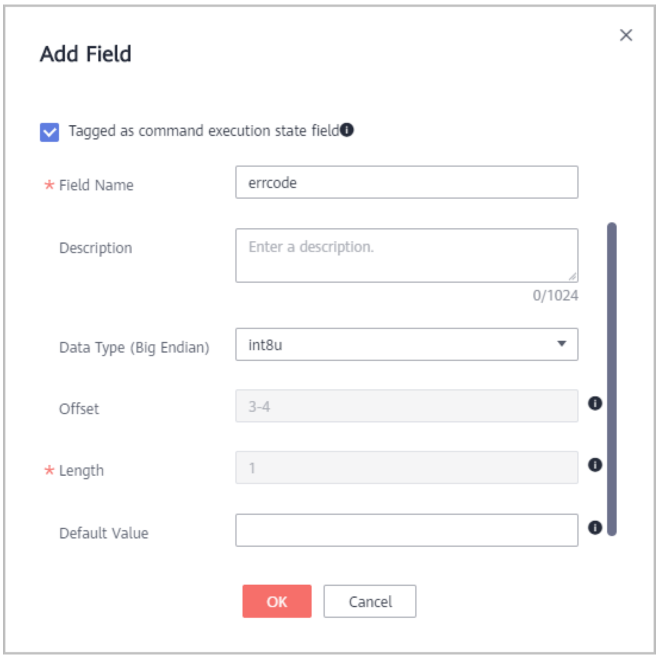
Default Value:

iv. In the **Add Message** dialog box, click **Add Response Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.

v. Click **Add Response Field**. In the dialog box displayed, select **Tagged as response ID field**, and click **OK**.

vi. Click **Add Response Field**. In the dialog box displayed, select **Tagged as command execution state field** to add the address field **errcode**, and click **OK**.

Figure 2-34 Adding a field



Add Field

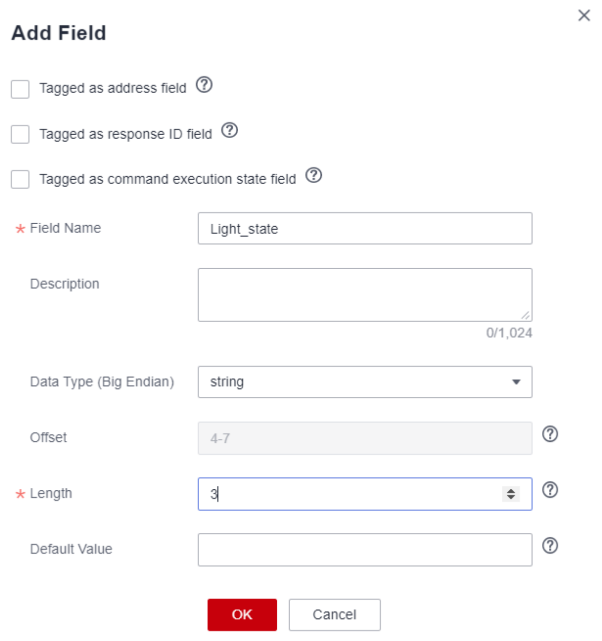
☑ Tagged as command execution state field

* Field Name	errcode
Description	Enter a description. 0/1024
Data Type (Big Endian)	int8u
Offset	3-4
* Length	1
Default Value	

vii. Click **Add Response Field**. In the dialog box displayed, add the **Light_state** field, and click **OK**.

- **Field Name**: Enter **Light_state**.
- **Data Type**: Select **string**.
- **Length**: Enter **3**.

Figure 2-35 Adding a field



Add Field

□ Tagged as address field

□ Tagged as response ID field

□ Tagged as command execution state field

* Field Name: Light_state

Description: 0/1,024

Data Type (Big Endian): string

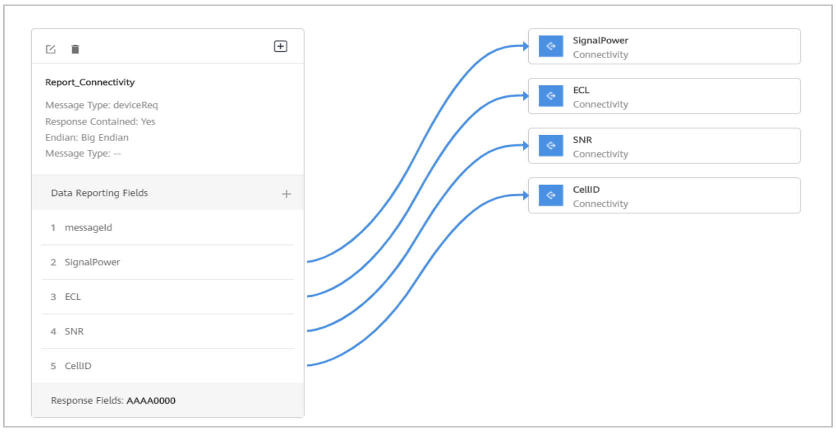
Offset: 4-7

* Length: 3

Default Value:

In the **Add Message** dialog box, click **OK** to complete the configuration of **Set_LED**.

g. Map the property fields, command fields, and response fields in **Product Model** on the right with the fields in the data reporting message, command delivery message, and command response.



Report_Connectivity

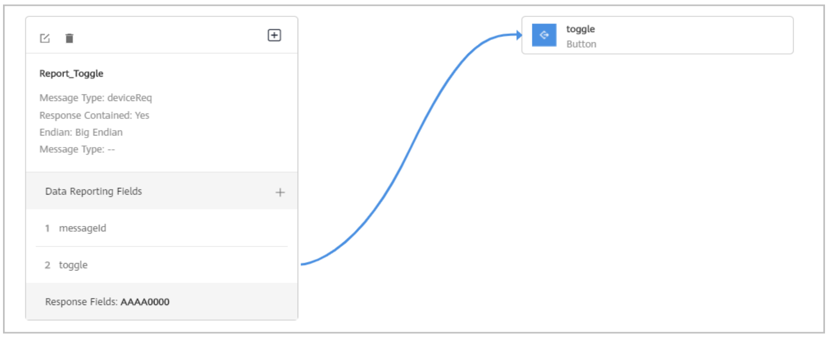
Message Type: deviceReq
Response Contained: Yes
Endian: Big Endian
Message Type: --

Data Reporting Fields

1 messageId
2 SignalPower
3 ECL
4 SNR
5 CellID

Response Fields: AAAA0000

SignalPower Connectivity
ECL Connectivity
SNR Connectivity
CellID Connectivity



Report_Toggle

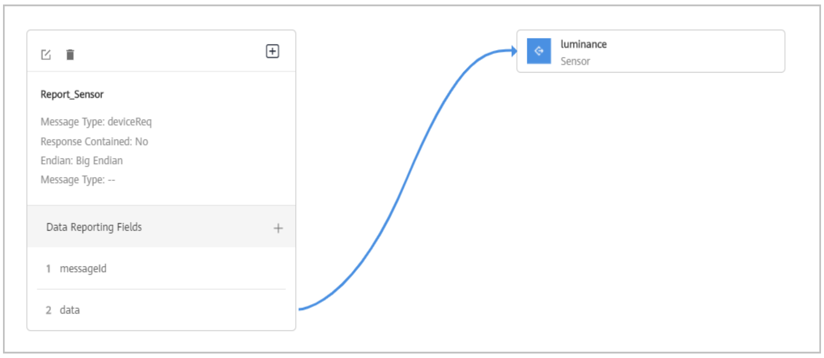
Message Type: deviceReq
Response Contained: Yes
Endian: Big Endian
Message Type: --

Data Reporting Fields

1 messageId
2 toggle

Response Fields: AAAA0000

toggle Button



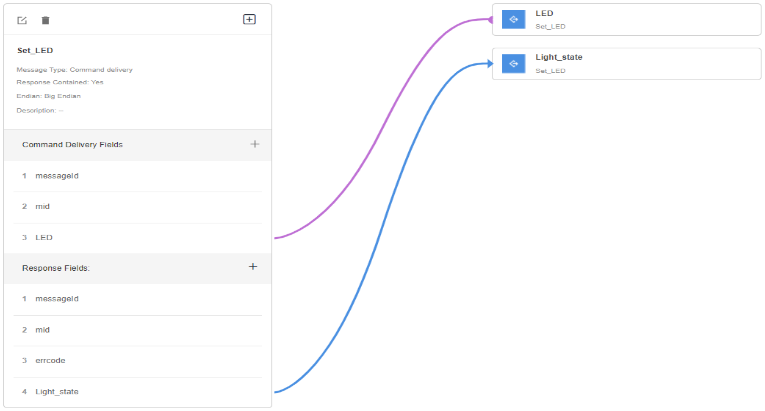
Report_Sensor

Message Type: deviceReq
Response Contained: No
Endian: Big Endian
Message Type: --

Data Reporting Fields

1 messageId
2 data

luminance
Sensor



Set_LED

Message Type: Command delivery
Response Contained: Yes
Endian: Big Endian
Description: --

Command Delivery Fields

1 messageId
2 mid
3 LED

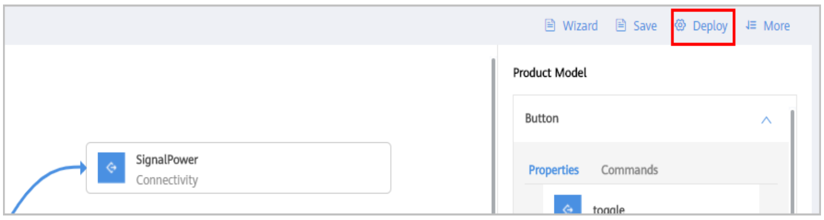
Response Fields:

1 messageId
2 mid
3 errcode
4 Light_state

LED
Set_LED

Light_state
Set_LED

h. Click **Save** and then **Deploy** to deploy the codec on the platform.



2.3 Developing a Smart Smoke Detector Using NB-IoT BearPi

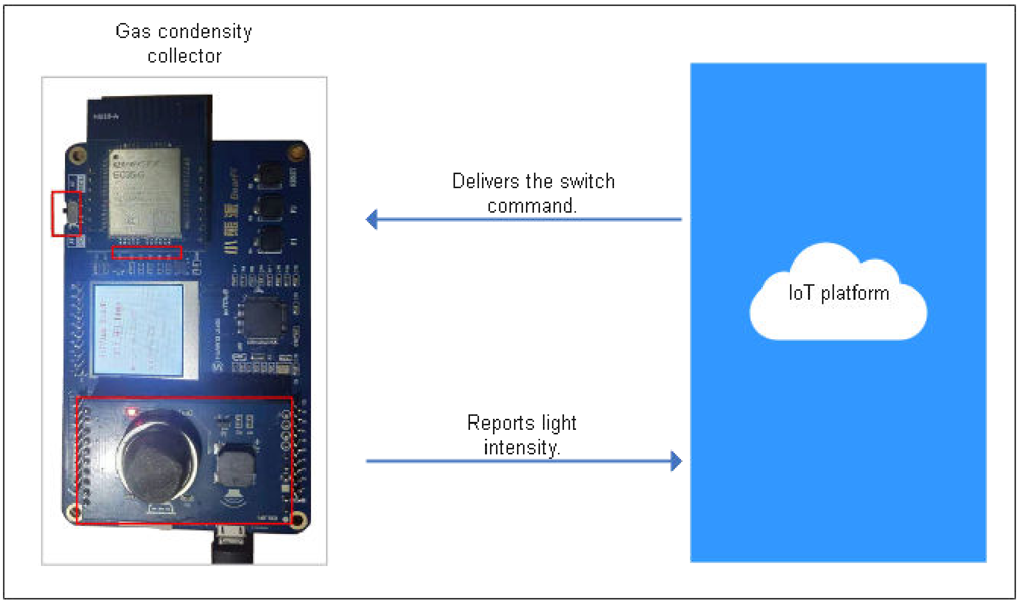
Scenarios

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

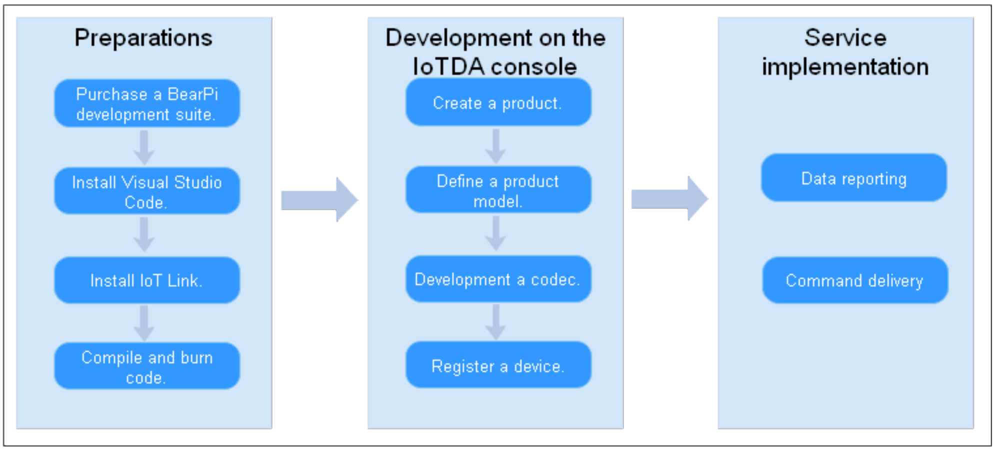
Huawei Cloud IoTDA. A smart smoke detector detects and reports smoke density to the IoTDA console. The beep switch can be remotely controlled on the IoTDA console.



Development Environment

- Development board: BearPi-IoT development suite (including the NB-IoT card, NB-IoT module, smart smoke detector module, and USB data cable)
- 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



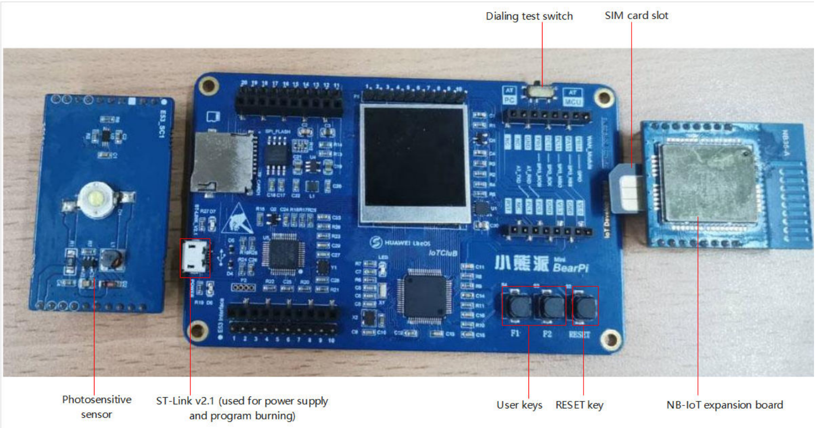
Introduction to the BearPi Development Board

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.

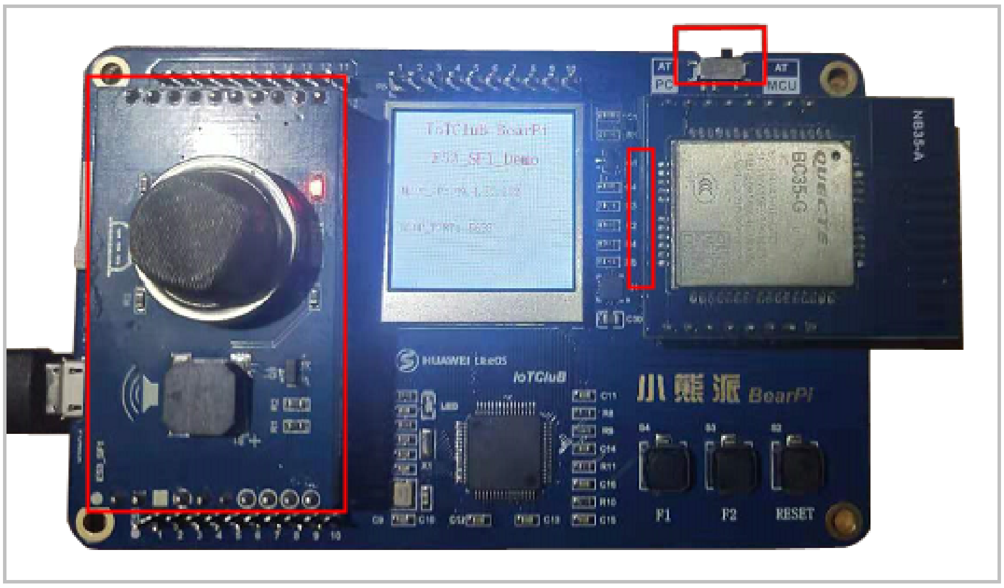
Figure 2-36 BearPi development board



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 2-37**.
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.

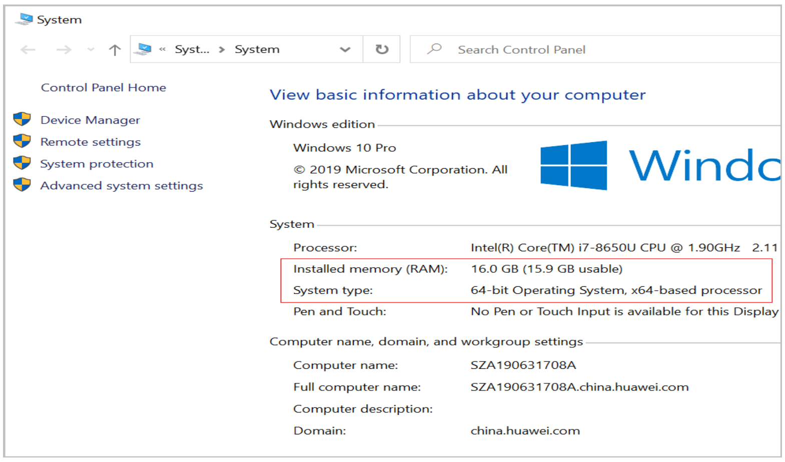
Figure 2-37 Hardware connection



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.



View basic information about your computer

Windows edition

Windows 10 Pro

© 2019 Microsoft Corporation. All rights reserved.

System

Processor: Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz 2.11

Installed memory (RAM): 16.0 GB (15.9 GB usable)

System type: 64-bit Operating System, x64-based processor

Pen and Touch: No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

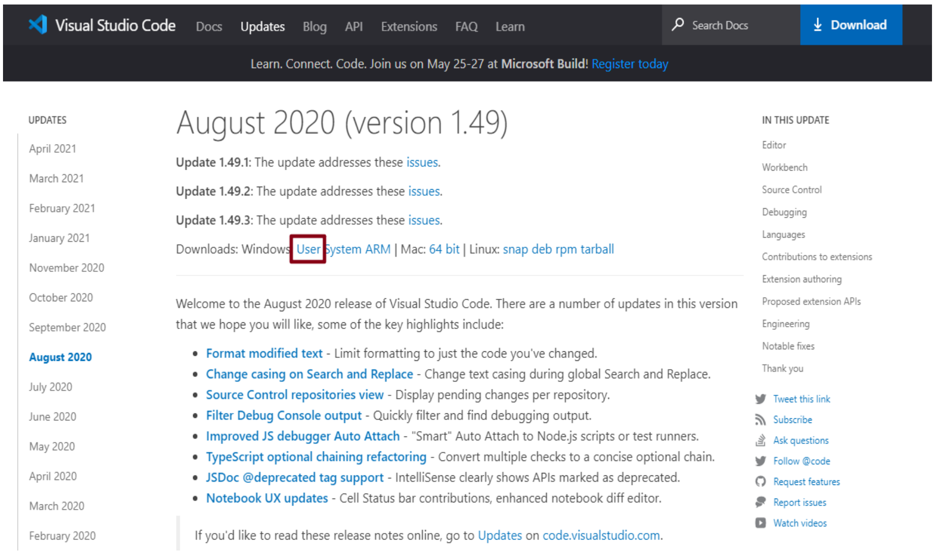
Computer name: SZA190631708A

Full computer name: SZA190631708A.china.huawei.com

Computer description:

Domain: china.huawei.com

Step 2 Click here to download and install a Visual Studio Code version that best 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.



August 2020 (version 1.49)

Update 1.49.1: The update addresses these issues.

Update 1.49.2: The update addresses these issues.

Update 1.49.3: The update addresses these issues.

Downloads: Windows: User System ARM | Mac: 64 bit | Linux: snap deb rpm tarball

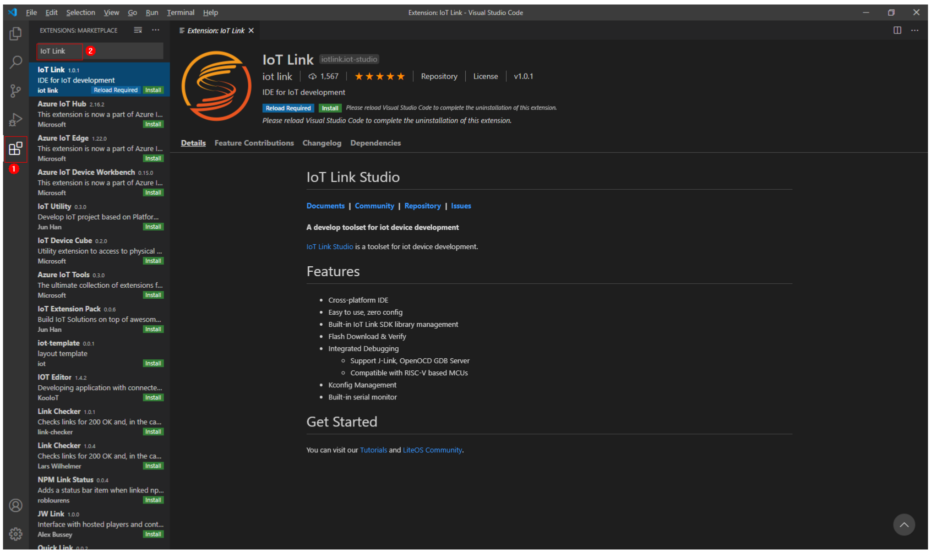
Welcome to the August 2020 release of Visual Studio Code. There are a number of updates in this version that we hope you will like, some of the key highlights include:

- Format modified text - Limit formatting to just the code you've changed.
- Change casing on Search and Replace - Change text casing during global Search and Replace.
- Source Control repositories view - Display pending changes per repository.
- Filter Debug Console output - Quickly filter and find debugging output.
- Improved JS debugger Auto Attach - "Smart" Auto Attach to Node.js scripts or test runners.
- TypeScript optional chaining refactoring - Convert multiple checks to a concise optional chain.
- JSDoc @deprecated tag support - IntelliSense clearly shows APIs marked as deprecated.
- Notebook UX updates - Cell Status bar contributions, enhanced notebook diff editor.

If you'd like to read these release notes online, go to Updates on code.visualstudio.com.

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.



IDE for IoT development

Please reload Visual Studio Code to complete the uninstallation of this extension.

Please reload Visual Studio Code to complete the uninstallation of this extension.

IoT Link Studio

Documents | Community | Repository | Issues

A develop toolset for iot device development

IoT Link Studio is a toolset for iot device development.

Features

- Cross-platform IDE
- Easy to use, zero config
- Built-in IoT Link SDK library management
- Flash Download & Verify
- Integrated Debugging
  - Support J-Link, OpenOCD GDB Server
  - Compatible with RISC-V based MCUs
- Kconfig Management
- Built-in serial monitor

Get Started

You can visit our Tutorials and LiteOS Community.

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

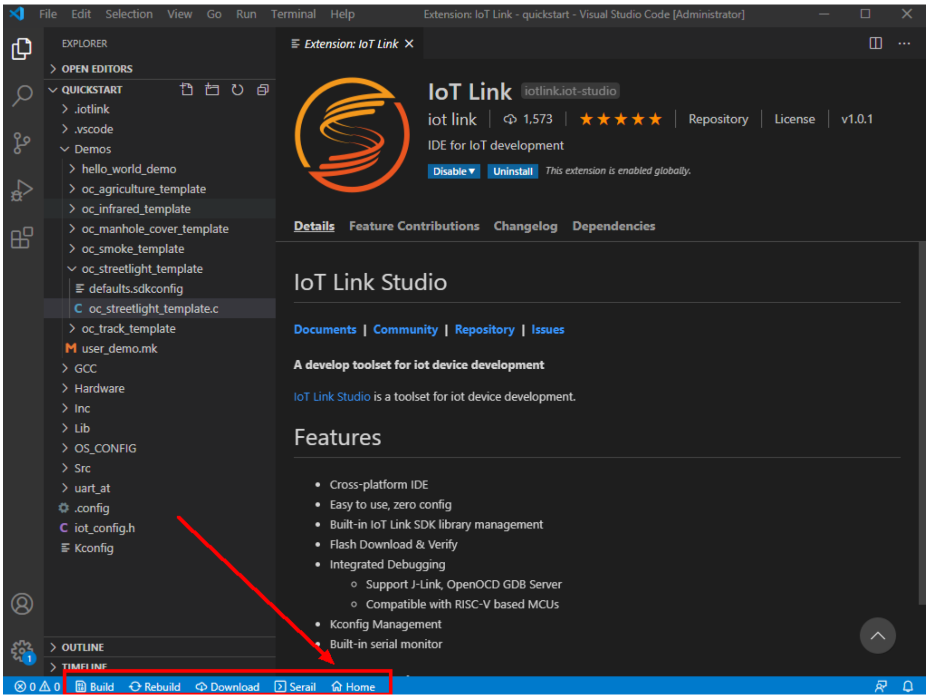
If a proxy is required, click ⚙ in the lower left corner of the Visual Studio Code home page and choose **Settings** > **Application** > **Proxy**, and set **Use the proxy support for extensions** to **on**.

----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 sample code (displayed **after a project is created**).
- **Download** is used to burn the compiled code to the MCU (displayed **after a project is created**).

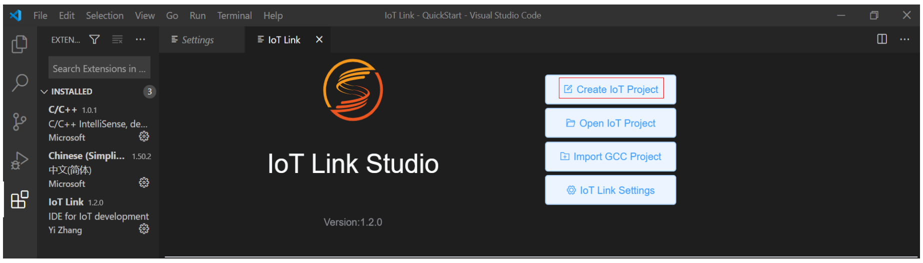


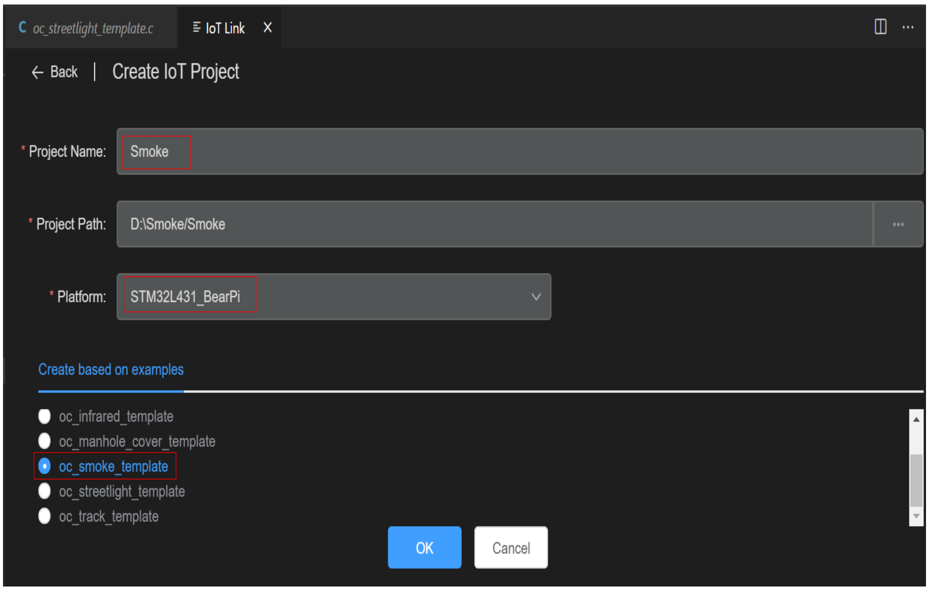
A develop toolset for iot device development

IoT Link Studio is a toolset for iot device development.

- Cross-platform IDE
- Easy to use, zero config
- Built-in IoT Link SDK library management
- Flash Download & Verify
- Integrated Debugging
  - Support J-Link, OpenOCD GDB Server
  - Compatible with RISC-V based MCUs
- Kconfig Management
- Built-in serial monitor

Step 2 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: Smoke

* Project Path: D:\Smoke/Smoke

* Platform: STM32L431_BearPi

○ oc_infrared_template
○ oc_manhole_cover_template
● oc_smoke_template
○ oc_streetlight_template
○ oc_track_template

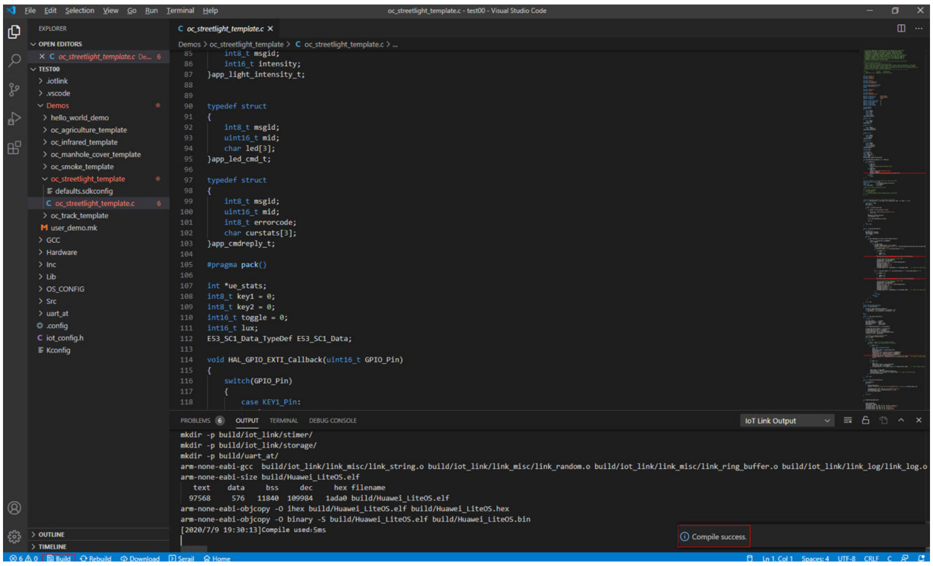
- **Project Name**: Set this parameter as required, for example, **Smoke**.
- **Project Path**: You can use the default installation path or select a path in a disk other than the system disk, for example, **D:\**.
- **Platform**: Currently, the demo applies only to the STM32L431_BearPi hardware platform. 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**.

----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.

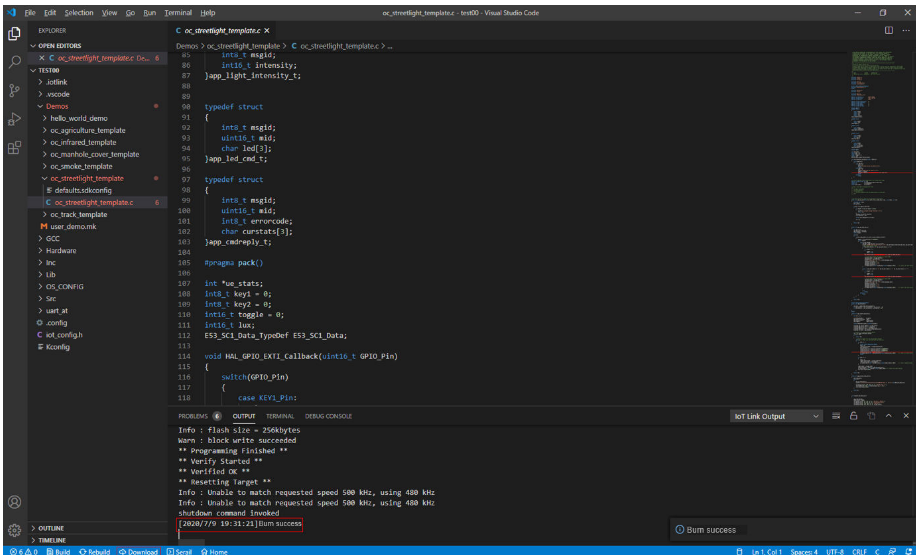


```
    int8_t msgid;
    int16_t intensity;
}app_light_intensity_t;


typedef struct
{
    int8_t msgid;
    uint16_t mid;
    char led[3];
}app_led_cmd_t;

typedef struct
{
    int8_t msgid;
    uint16_t mid;
    int8_t errorcode;
    char curstats[3];
}app_cmdreply_t;

#pragma pack()

int *ue_stats;
int8_t key1 = 0;
int8_t key2 = 0;
int16_t toggle = 0;
int16_t lux;
E53_SC1_Data_TypeDef E53_SC1_Data;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch(GPIO_Pin)
    {
        case KEY1_Pin:
```

```
mkdir -p build/iot_link/stimer/
mkdir -p build/iot_link/storage/
mkdir -p build/uart_at/
arm-none-eabi-gcc  build/iot_link/link_misc/link_string.o build/iot_link/link_misc/link_random.o build/iot_link/link_misc/link_ring_buffer.o build/iot_link/link_log/link_log.o
arm-none-eabi-size build/Huawei_LiteOS.elf
   text    data     bss     dec     hex filename
  97568     576   11840  109984   1ada0 build/Huawei_LiteOS.elf
arm-none-eabi-objcopy -O ihex build/Huawei_LiteOS.elf build/Huawei_LiteOS.hex
arm-none-eabi-objcopy -O binary -S build/Huawei_LiteOS.elf build/Huawei_LiteOS.bin
[2020/7/9 19:30:13]Compile used:5ms
```

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.



```
    int8_t msgid;
    int16_t intensity;
}app_light_intensity_t;


typedef struct
{
    int8_t msgid;
    uint16_t mid;
    char led[3];
}app_led_cmd_t;

typedef struct
{
    int8_t msgid;
    uint16_t mid;
    int8_t errorcode;
    char curstats[3];
}app_cmdreply_t;

#pragma pack()

int *ue_stats;
int8_t key1 = 0;
int8_t key2 = 0;
int16_t toggle = 0;
int16_t lux;
E53_SC1_Data_TypeDef E53_SC1_Data;

void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    switch(GPIO_Pin)
    {
        case KEY1_Pin:
```

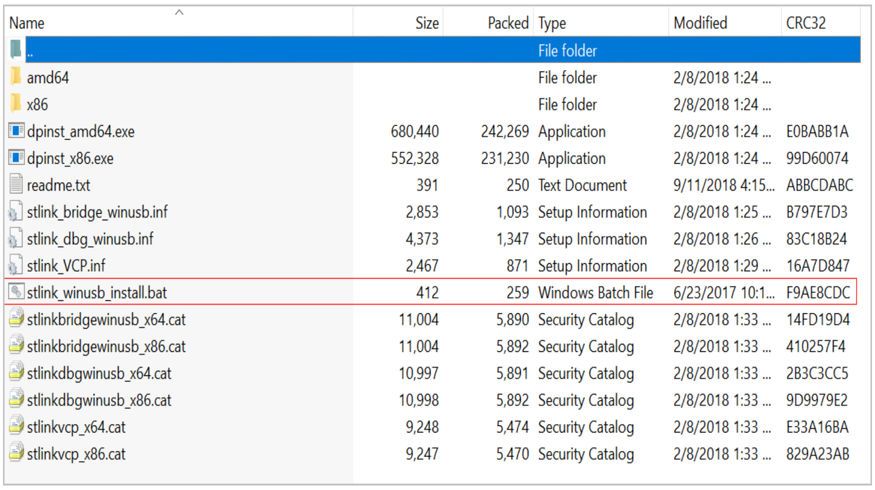
```
Info : flash size = 256kbytes
Warn : block write succeeded
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
Info : Unable to match requested speed 500 kHz, using 480 kHz
Info : Unable to match requested speed 500 kHz, using 480 kHz
shutdown command invoked
[2020/7/9 19:31:21]Burn success
```

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.

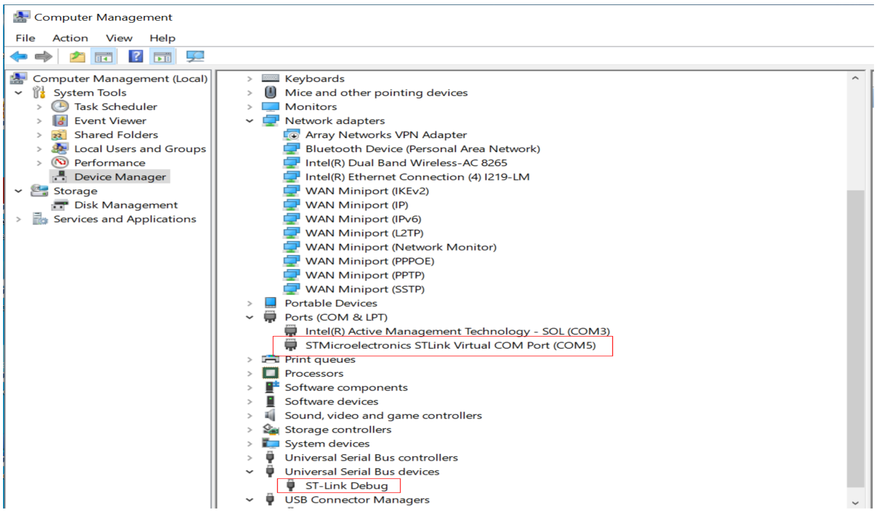
1. Visit the **ST website**, download the ST-Link driver, and double-click the **stlink_winusb_install.bat** file to start automatic installation. This section uses Windows 10 64-bit ST-Link 2.0.1 as an example.



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 ...	2B3C3CC5
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.



Computer Management

File Action View Help

- Computer Management (Local)
  - System Tools
    - Task Scheduler
    - Event Viewer
    - Shared Folders
    - Local Users and Groups
    - Performance
    - Device Manager
  - Storage
    - Disk Management
  - Services and Applications

- Keyboards
- Mice and other pointing devices
- Monitors
- Network adapters
  - Array Networks VPN Adapter
  - Bluetooth Device (Personal Area Network)
  - Intel(R) Dual Band Wireless-AC 8265
  - Intel(R) Ethernet Connection (4) I219-LM
  - WAN Miniport (IKEv2)
  - WAN Miniport (IP)
  - WAN Miniport (IPv6)
  - WAN Miniport (L2TP)
  - WAN Miniport (Network Monitor)
  - WAN Miniport (PPPOE)
  - WAN Miniport (PPTP)
  - WAN Miniport (SSTP)
- Portable Devices
- Ports (COM & LPT)
  - Intel(R) Active Management Technology - SOL (COM3)
  - STMicroelectronics STLink Virtual COM Port (COM5)
- Print queues
- Processors
- Software components
- Software devices
- Sound, video and game controllers
- Storage controllers
- System devices
- Universal Serial Bus controllers
- Universal Serial Bus devices
  - ST-Link Debug
- USB Connector Managers

----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 (profiles), and codecs generated during IoT capability building.

Step 1 Visit the **IoTDA** product page and click **Access Console**.

Step 2 In the navigation pane, click **Products**. Click the drop-down list in the upper right corner, and select the resource space to which the new product belongs.

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

Basic Information

Resource Space	Select the resource space to which the product belongs.
Product Name	Enter a name, for example, **BearPi_Smoke**.
Protocol	Select **LwM2M over CoAP**.
Data Type	Select **Binary**.
Manufacturer	Enter a name, for example, **BearPi**.
Industry	Select **Default**.
Device Type	Enter **Smoke**.

Step 4 After the product is created, click the product to access its details and perform subsequent operations.

----End

Defining a Product Model

On the **Model Definition** tab page, click **Custom Model** to add a service for the product.

Table 2-6 describes the service defined in the product model.

Table 2-6 Device service list

Service ID	Description
Enter **Smoke**.	Detects smoke density in real time.

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

Table 2-7 Smoke

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

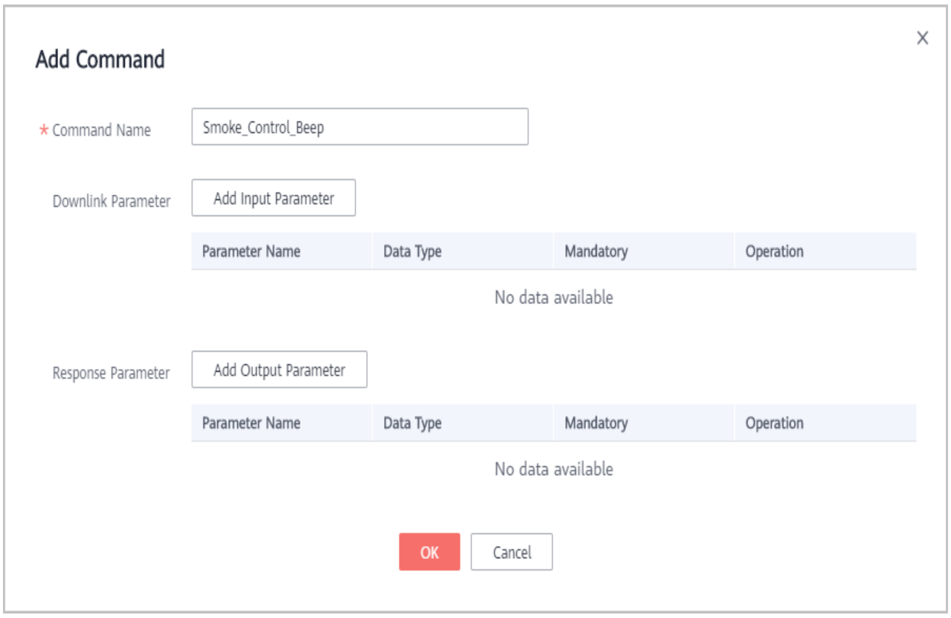
Table 2-8 Smoke commands

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

Adding the Smoke Service

1. 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**.
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 0–65535.
   - **Step**: Enter **0**.
   - **Unit**: Leave it blank.
3. Choose **Smoke**, click **Add Command**, and enter the command name **Smoke_Control_Beep**.

Figure 2-38 Adding a command

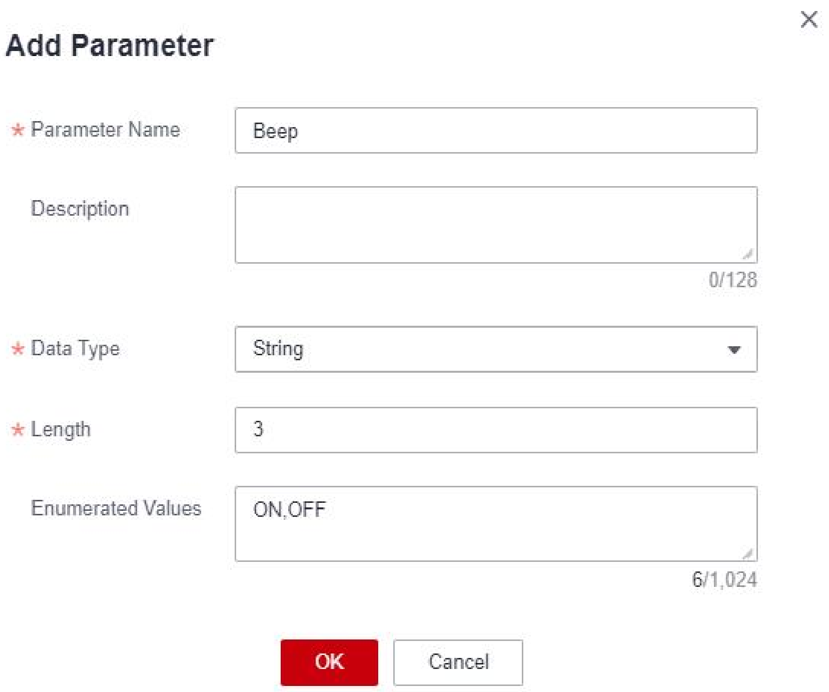


Add Command

* Command Name: Smoke_Control_Beep

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

Figure 2-39 Adding a parameter



Add Parameter

* Parameter Name: Beep

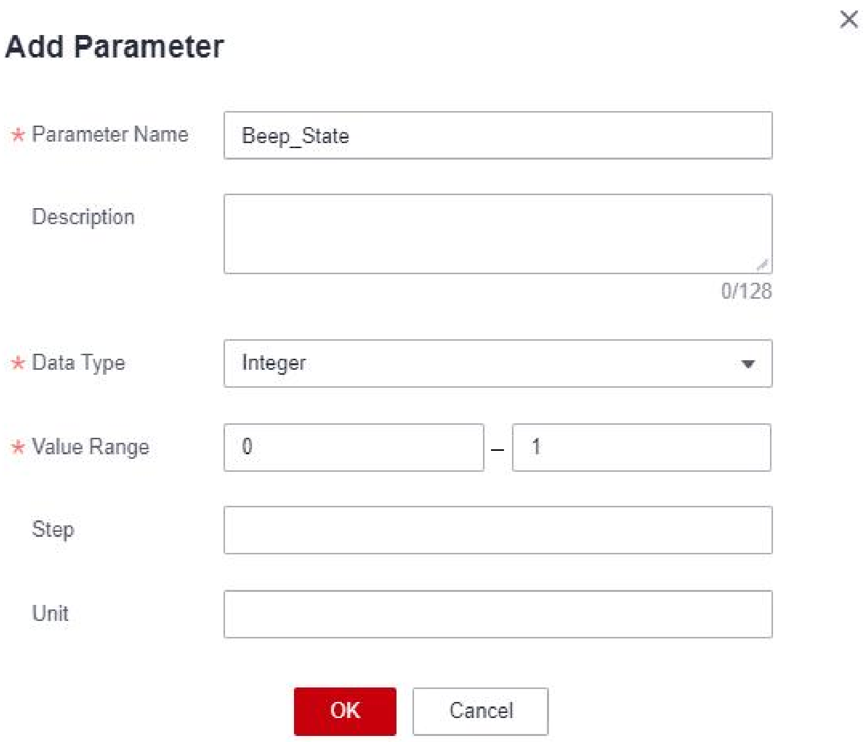
Description: 0/128

* Data Type: String

* Length: 3

Enumerated Values: ON,OFF 6/1,024

Figure 2-40 Adding a parameter



Add Parameter

* Parameter Name: Beep_State

Description: 0/128

* Data Type: Integer

* Value Range: 0 – 1

Step:

Unit:

Developing a Codec

Step 1 On the product details page, click **Develop Codec** on the **Develop Codec** tab page.

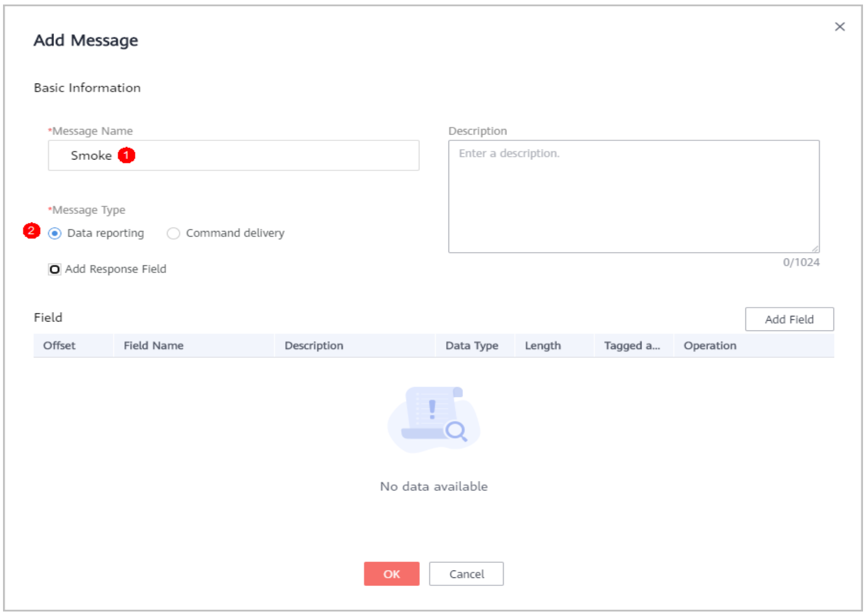
Step 2 Click **Add Message**.

NOTE

Develop the codec based on the operations provided in this section and ensure that the fields are added in the order specified in this section.

Step 3 Add the **Smoke** message. A configuration example is as follows:

- **Message Name**: Enter **Smoke**.
- **Message Type**: Select **Data reporting**.



Add Message

Basic Information

*Message Name: Smoke

Description: Enter a description. 0/1024

*Message Type: ● Data reporting ○ Command delivery

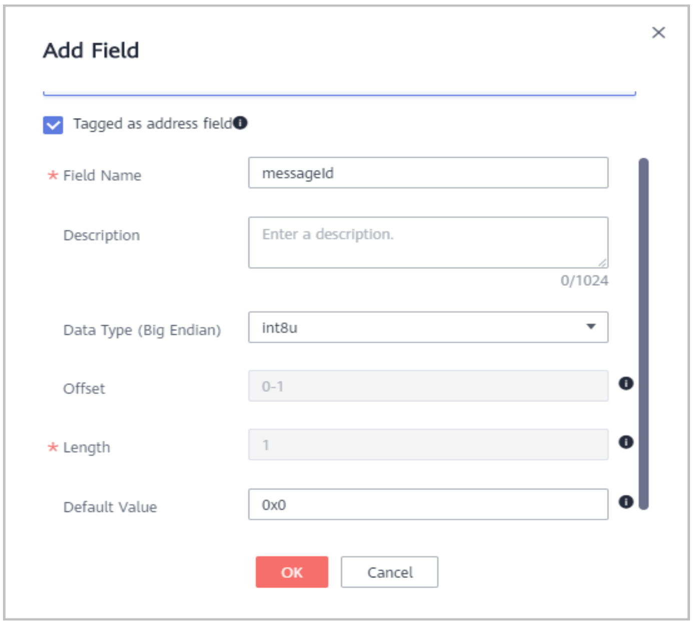
□ Add Response Field

Field

Offset	Field Name	Description	Data Type	Length	Tagged a...	Operation

No data available

1. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, and click **OK**.



Add Field

☑ Tagged as address field

* Field Name: messageId

Description: Enter a description. 0/1024

Data Type (Big Endian): int8u

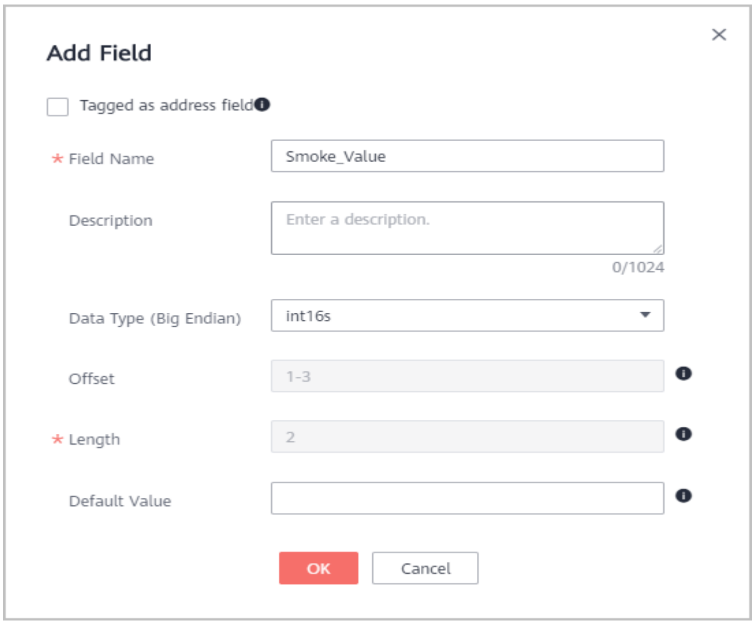
Offset: 0-1

* Length: 1

Default Value: 0x0

OK Cancel

2. Click **Add Field**, add the **Smoke_Value** field, enter related information, and click **OK**.
   - **Field Name**: Enter **Smoke_Value**.
   - **Data Type**: Select **int16s**.



Add Field

□ Tagged as address field

* Field Name: Smoke_Value

Description: Enter a description. 0/1024

Data Type (Big Endian): int16s

Offset: 1-3

* Length: 2

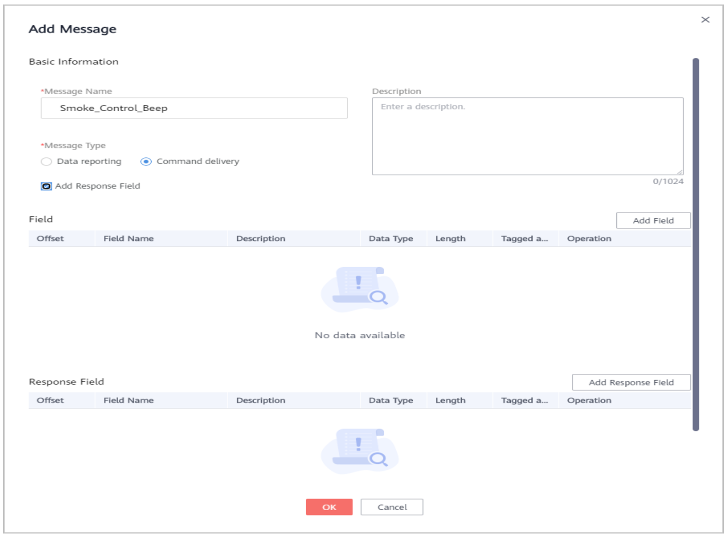
Default Value:

OK Cancel

3. In the **Add Message** dialog box, click **OK** to complete the configuration of **Smoke_Value**.

Step 4 Add the **Smoke_Control_Beep** message. A configuration example is as follows:

- **Message Name**: Enter **Smoke_Control_Beep**.
- **Message Type**: Select **Command delivery**.
- **Add Response Field**: Select this option.



Add Message

Basic Information

*Message Name: Smoke_Control_Beep

Description: Enter a description. 0/1024

*Message Type: ○ Data reporting ● Command delivery

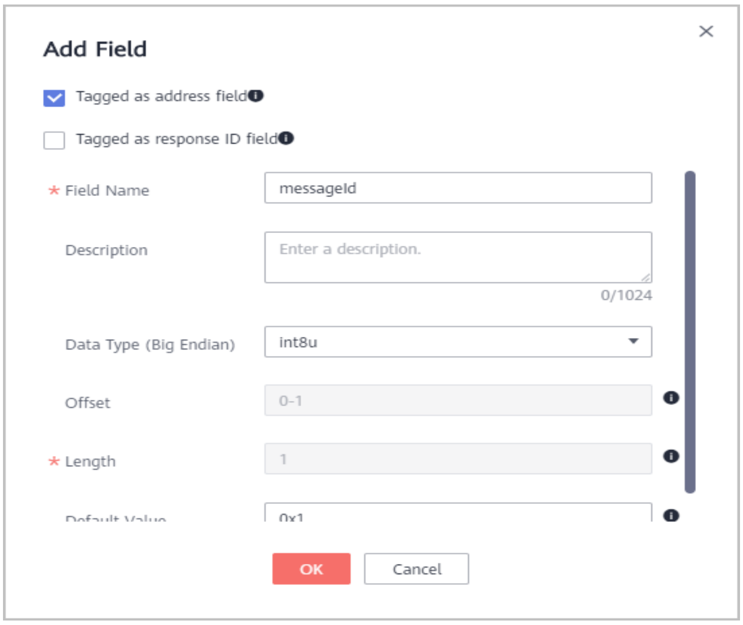
☑ Add Response Field

Offset	Field Name	Description	Data Type	Length	Tagged a...	Operation

No data available

Offset	Field Name	Description	Data Type	Length	Tagged a...	Operation

1. In the **Add Message** dialog box, click **Add Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, set **Default value** to **0x9**, and click **OK**.



Add Field

☑ Tagged as address field

□ Tagged as response ID field

* Field Name: messageId

Description: Enter a description. 0/1024

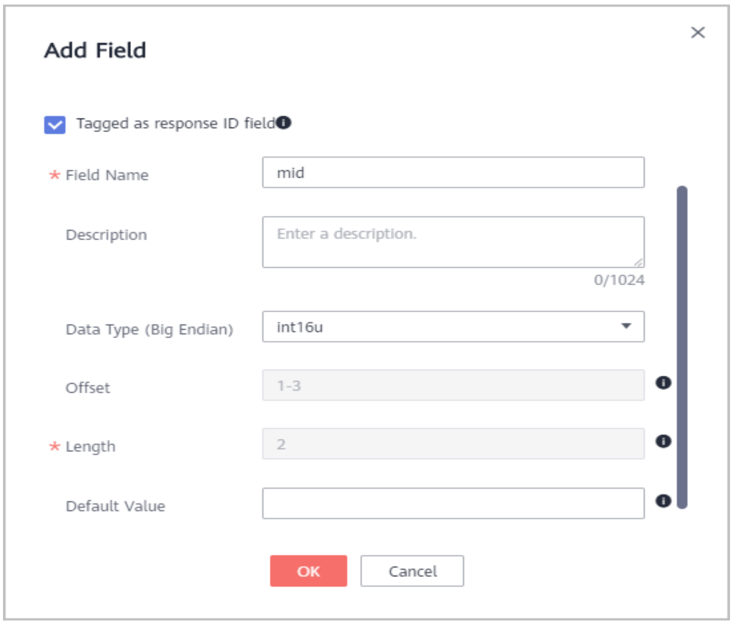
Data Type (Big Endian): int8u

Offset: 0-1

* Length: 1

Default Value: 0x1

2. Click **Add Field**. In the dialog box displayed, select **Tagged as response ID field** to add the address field **mid**, and click **OK**.



Add Field

☑ Tagged as response ID field

* Field Name: mid

Description: Enter a description. 0/1024

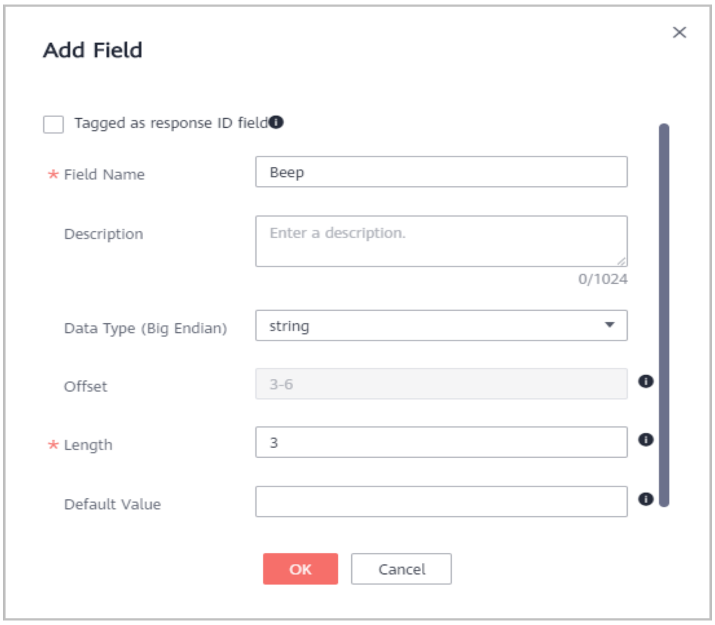
Data Type (Big Endian): int16u

Offset: 1-3

* Length: 2

Default Value:

3. Click **Add Field**. In the dialog box displayed, add the **Beep** field, and click **OK**.
   - **Field Name**: Enter **Beep**.
   - **Data Type**: Select **string**.
   - **Length**: Enter **3**.



Add Field

□ Tagged as response ID field

* Field Name: Beep

Description: Enter a description. 0/1024

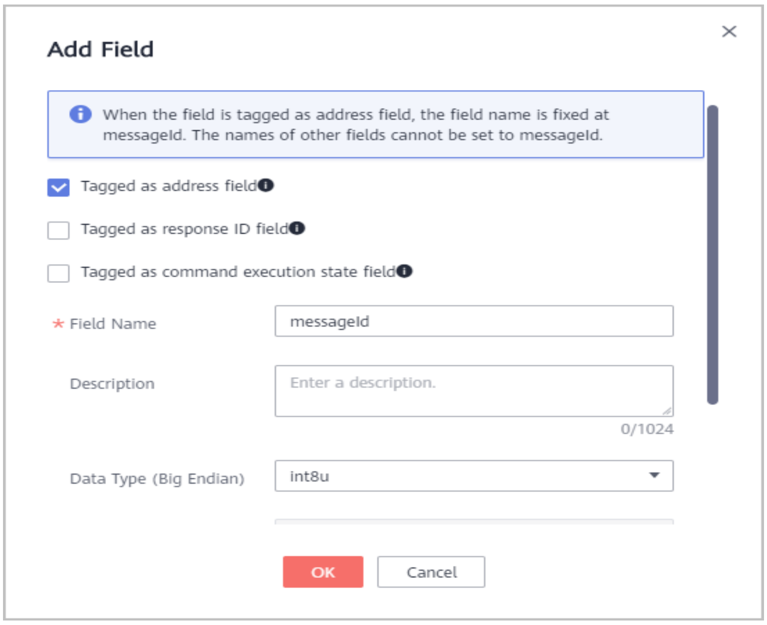
Data Type (Big Endian): string

Offset: 3-6

* Length: 3

Default Value:

4. Click **Add Response Field**. In the dialog box displayed, select **Tagged as address field** to add the address field **messageId**, set **Default value** to **0xa**, and click **OK**.



Add Field

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.

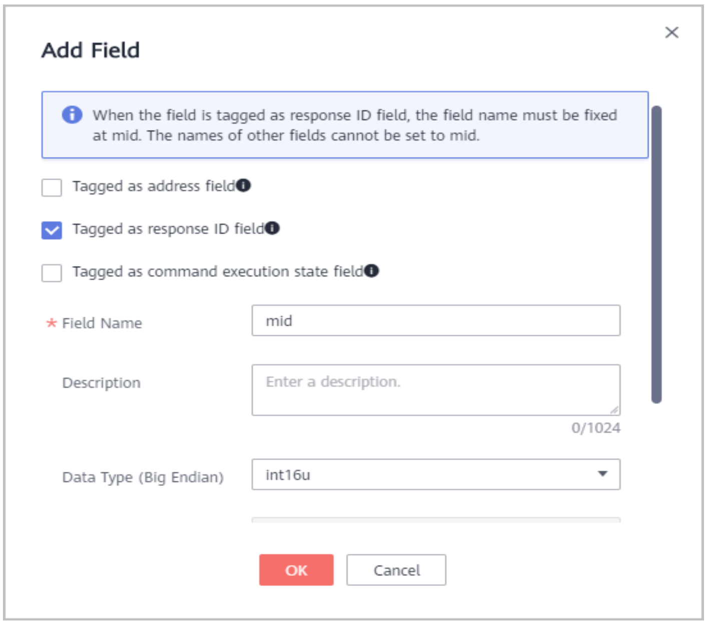
☑ Tagged as address field

□ Tagged as response ID field

□ Tagged as command execution state field

* Field Name	messageId
Description	Enter a description. 0/1024
Data Type (Big Endian)	int8u

5. Click **Add Response Field**. In the dialog box displayed, select **Tagged as response ID field** to add the address field **mid**, and click **OK**.



Add Field

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.

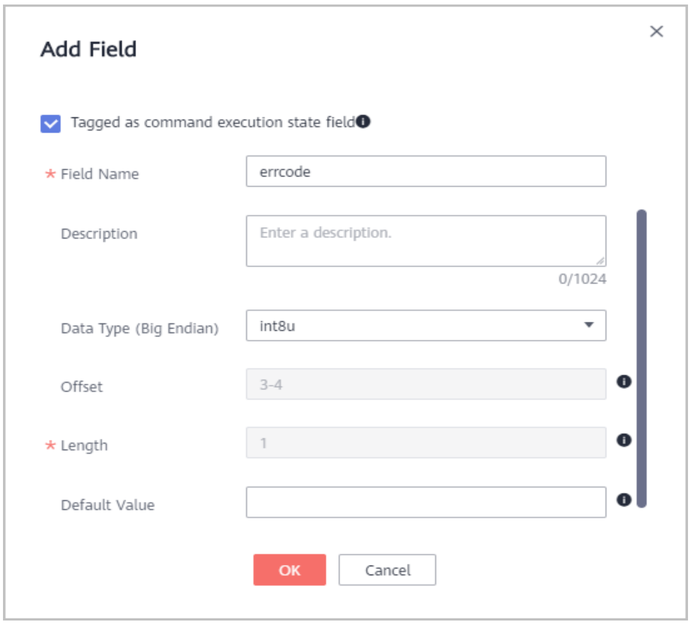
□ Tagged as address field

☑ Tagged as response ID field

□ Tagged as command execution state field

* Field Name	mid
Description	Enter a description. 0/1024
Data Type (Big Endian)	int16u

6. Click **Add Response Field**. In the dialog box displayed, select **Tagged as command execution state field** to add the address field **errcode**, and click **OK**.



Add Field

☑ Tagged as command execution state field

* Field Name: errcode

Description: Enter a description. 0/1024

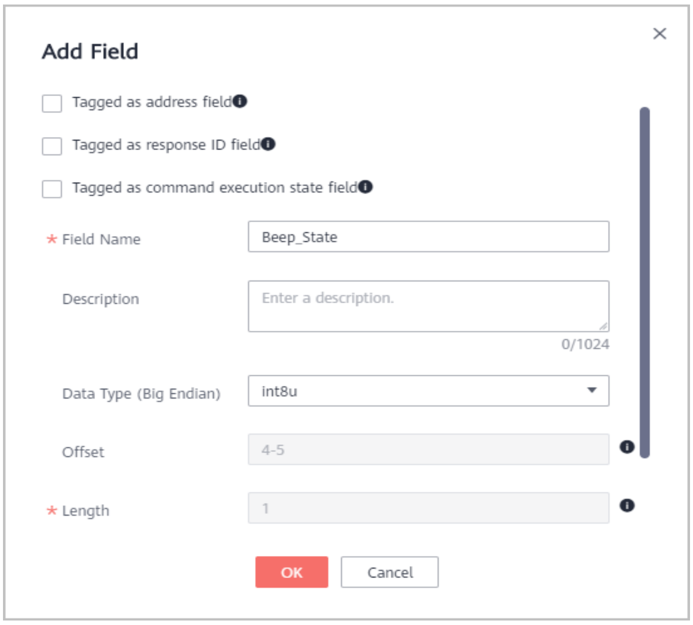
Data Type (Big Endian): int8u

Offset: 3-4

* Length: 1

Default Value:

7. Click **Add Response Field**. In the dialog box displayed, add the **Beep_State** field, and click **OK**.
   - **Field Name**: Enter **Beep_State**.
   - **Data Type**: Select **int8u**.
   - **Length**: Enter **1**.



Add Field

□ Tagged as address field

□ Tagged as response ID field

□ Tagged as command execution state field

* Field Name: Beep_State

Description: Enter a description. 0/1024

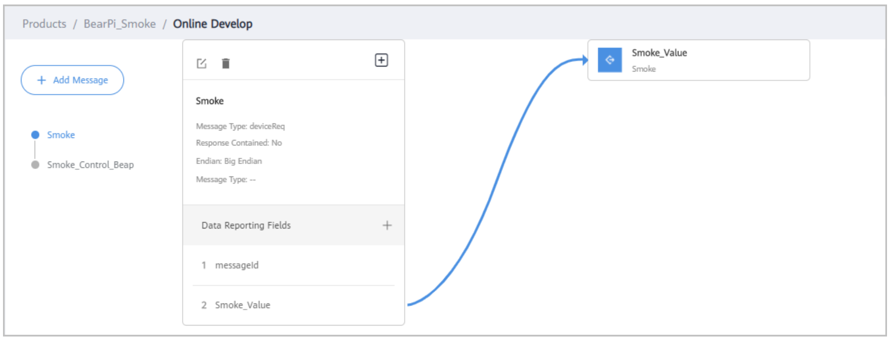
Data Type (Big Endian): int8u

Offset: 4-5

* Length: 1

8. In the **Add Message** dialog box, click **OK** to complete the configuration of **Smoke_Control_Beep**.

Step 5 Map the property fields, command fields, and response fields in **Product Model** on the right with the fields in the data reporting message, command delivery message, and command response.

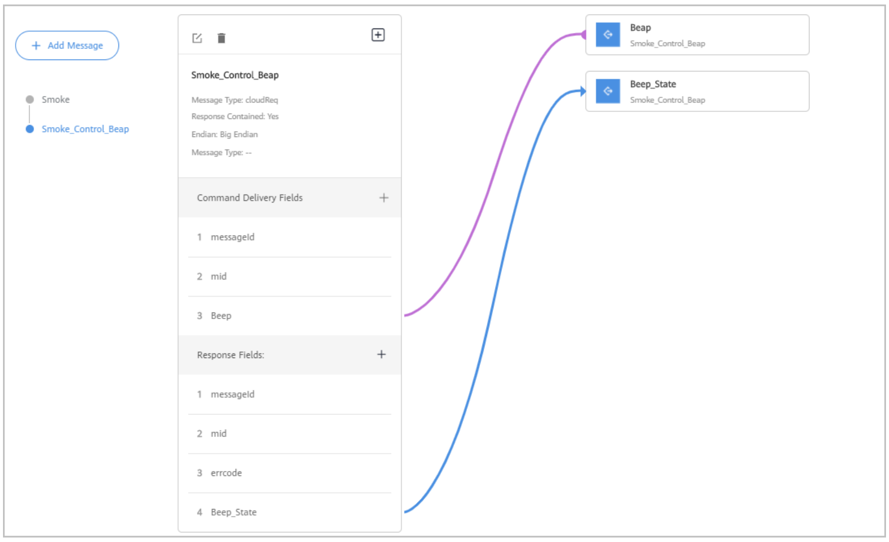


Smoke

Message Type: deviceReq
Response Contained: No
Endian: Big Endian
Message Type: --

Data Reporting Fields

1 messageId
2 Smoke_Value



Smoke_Control_Beap

Message Type: cloudReq
Response Contained: Yes
Endian: Big Endian
Message Type: --

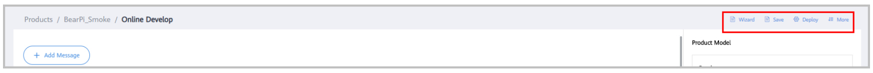
Command Delivery Fields

1 messageId
2 mid
3 Beep

Response Fields:

1 messageId
2 mid
3 errcode
4 Beep_State

Step 6 Click **Save** and then **Deploy** to deploy the codec on the platform.



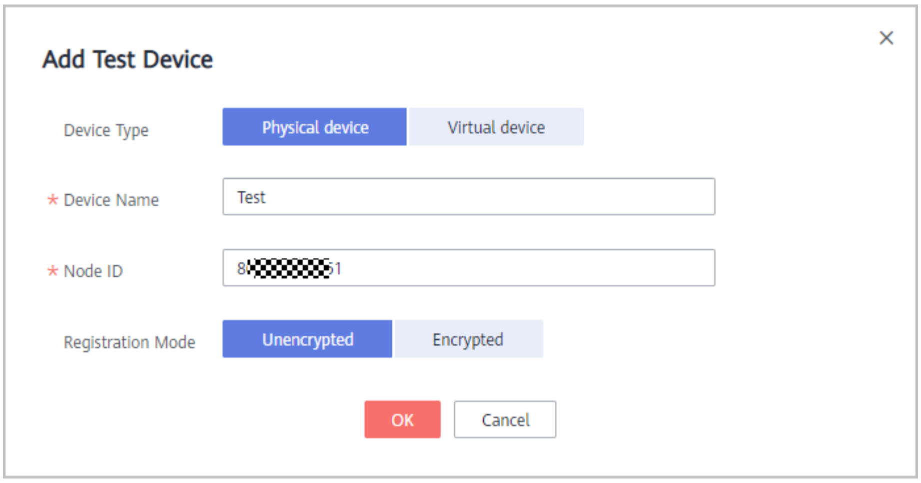
----End

Registering a Device

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

Step 1 On the product details page, click **Add Test Device** on the **Online Debugging** tab page

Step 2 In the **Add Test Device** dialog box, select **Physical device**, and enter the device name and node ID.



Add Test Device

Device Type: Physical device | Virtual device

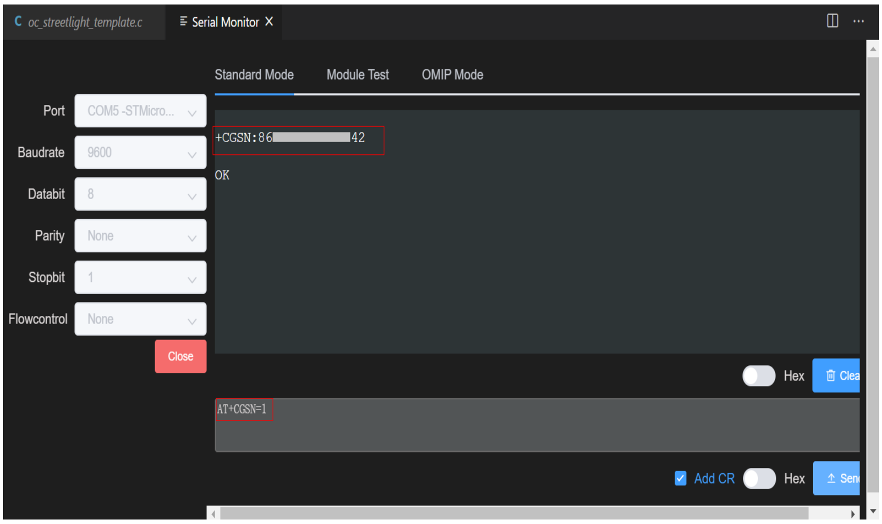
* Device Name: Test

* Node ID: 8[redacted]1

Registration Mode: Unencrypted | Encrypted

- **Device Name**: Customize a name, for example, **Test**.
- **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. Copy the IMEI to the console.

  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.



Port: COM5 -STMicro...
Baudrate: 9600
Databit: 8
Parity: None
Stopbit: 1
Flowcontrol: None

```
+CGSN:86[redacted]42

OK
```

```
AT+CGSN=1
```

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

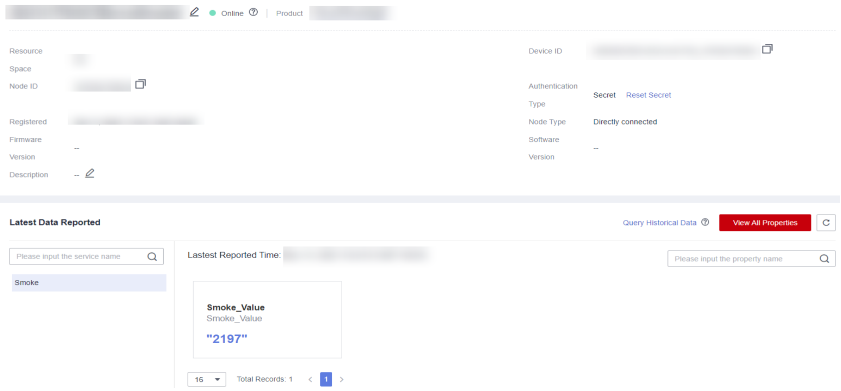
----End

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 choose **Devices** > **All Devices**.

Step 2 Select the device registered in **Registering a Device** and click **View** to view the data reported to the platform.



[redacted] Online | Product [redacted]

Resource Space [redacted]

Node ID [redacted]

Registered [redacted]

Firmware Version --

Description --

Device ID [redacted]

Authentication Type Secret Reset Secret

Node Type Directly connected

Software Version --

Latest Data Reported

Query Historical Data

View All Properties

Please input the service name

Smoke

Lastest Reported Time: [redacted]

Please input the property name

Smoke_Value
Smoke_Value
"2197"

16 Total Records: 1 < 1 >

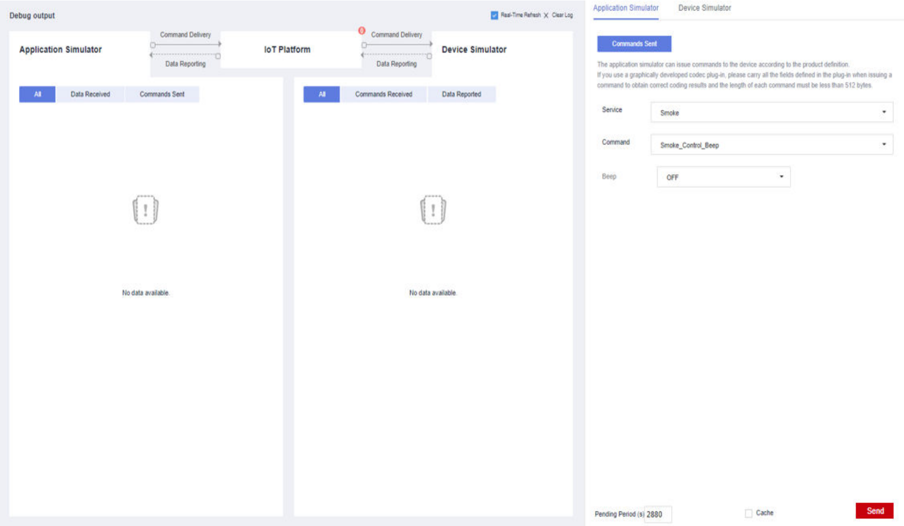
----End

Delivering a Command

Step 1 Log in to the **IoTDA console**. Click the product created in **Creating a Product** to go to the product details page.

Step 2 On the **Online Debugging** tab page, click the registered device to access the debugging page.

Step 3 After setting the command parameters, click **Send**. The smoke detector buzzes. After you send the **OFF** command, the smoke detector stops buzzing.



The application simulator can issue commands to the device according to the product definition.
If you use a graphically developed codec plug-in, please carry all the fields defined in the plug-in when issuing a command to obtain correct coding results and the length of each command must be less than 512 bytes.

Service: Smoke

Command: Smoke_Control_Beep

Beep: OFF

Pending Period (s) 2880

□ Cache

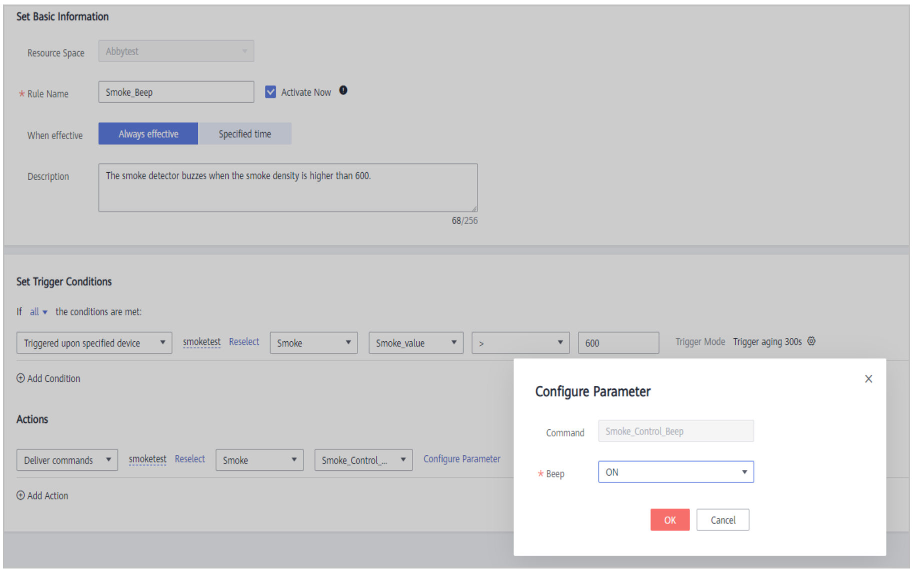
Send

----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.



Set Basic Information

Resource Space: Abbytest

* Rule Name: Smoke_Beep ☑ Activate Now

When effective: Always effective | Specified time

Description: The smoke detector buzzes when the smoke density is higher than 600.

68/256

Set Trigger Conditions

If all the conditions are met:

Triggered upon specified device | smoketest Reselect | Smoke | Smoke_value | > | 600 | Trigger Mode Trigger aging 300s

Actions

Deliver commands | smoketest Reselect | Smoke | Smoke_Control_... | Configure Parameter

Configure Parameter

Command: Smoke_Control_Beep

* Beep: ON

Parameter	Description
Rule Name	Specify the name of the rule to create, for example, **Smoke_Beep**.
Activate upon creation	Select this option.
Effective Period	Select **Always effective**.
Description	Enter a description of the rule, for example, "The smoke detector buzzes when the smoke density is higher than 600."
Set Triggers	1. Click **Add Trigger**. 2. Select **Triggered upon specified device** and select the device added in 2. 3. Select **Smoke** for **Service Type**, **Smoke_Value** for **Property**, **>** for **Operation**, and **600** for **Value**. Click **Trigger Mode**. In the dialog box displayed, set **Trigger Suppression** to **Yes** and **Data Validity Period (s)** to **300**, click **OK**.
Set Actions	1. Click **Add Action**. 2. Select **Deliver Commands**, and select the device created in 2. 3. Select **Smoke** for **Service Type**, and **Smoke_Control_Beep** for **Command**. Click **Configure Parameter**. In the dialog box displayed, set **Beap** to **ON**, and click **OK**.

----End

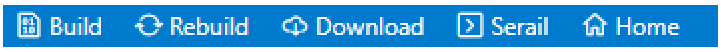
(Optional) Verification

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

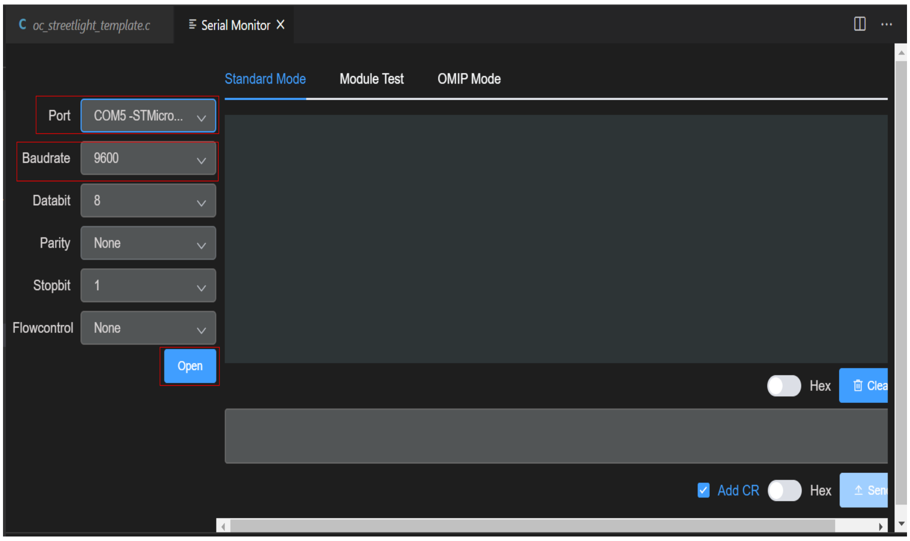
Locating Module Communication Problems Using AT Commands

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.

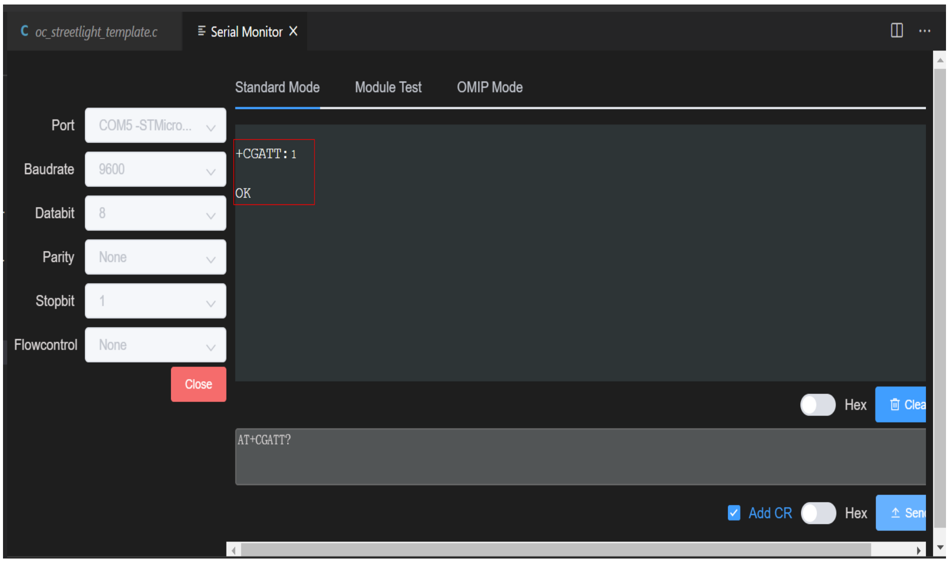


3. Select the port number obtained in **2**, set **Baudrate** to **9600**, and click **Open**.



Port COM5 -STMicro...
Baudrate 9600
Databit 8
Parity None
Stopbit 1
Flowcontrol None

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.



Port COM5 -STMicro...
Baudrate 9600
Databit 8
Parity None
Stopbit 1
Flowcontrol None

```
+CGATT:1

OK
```

```
AT+CGATT?
```

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 a Device Simulator to IoTDA

This topic uses a device simulator as an example to describe how to connect devices to IoTDA using the native MQTT protocol. The simulator is an MQTT client that enables you to easily verify whether devices can interact with the platform to publish or subscribe to messages.

Prerequisites

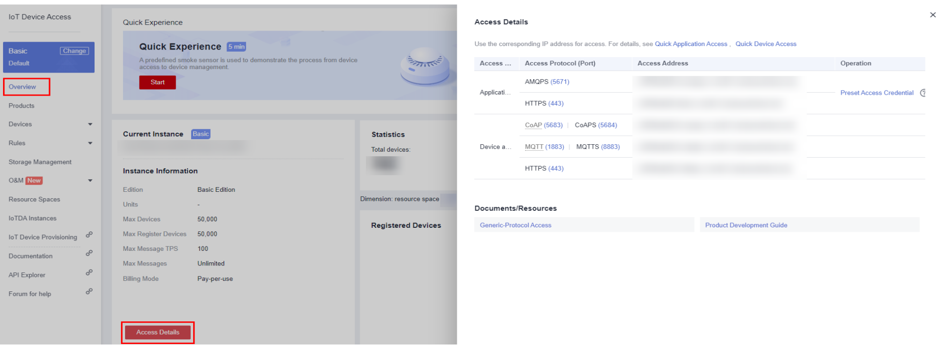
- You have registered a Huawei Cloud account. If you have not registered, click **here** to complete the registration.
- You have subscribed to the IoTDA service. If you have not subscribed to the service, go to the **IoTDA** service page, and click **Access Console** to subscribe to the service.

Obtaining Device Access Information

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

Step 1 Visit the **IoTDA** product page and click **Access Console**.

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



A predefined smoke sensor is used to demonstrate the process from device access to device management.

[redacted]

Instance Information

Edition	Basic Edition
Units	-
Max Devices	50,000
Max Register Devices	50,000
Max Message TPS	100
Max Messages	Unlimited
Billing Mode	Pay-per-use

[redacted]

Access Details

Use the corresponding IP address for access. For details, see Quick Application Access , Quick Device Access

Access ...	Access Protocol (Port)	Access Address	Operation
Applicati...	AMQPS (5671)	[redacted]	Preset Access Credential
	HTTPS (443)	[redacted]	
Device a...	CoAP (5683) | CoAPS (5684)	[redacted]	
	MQTT (1883) | MQTTS (8883)	[redacted]	
	HTTPS (443)	[redacted]	

Documents/Resources

Generic-Protocol Access

Product Development Guide

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

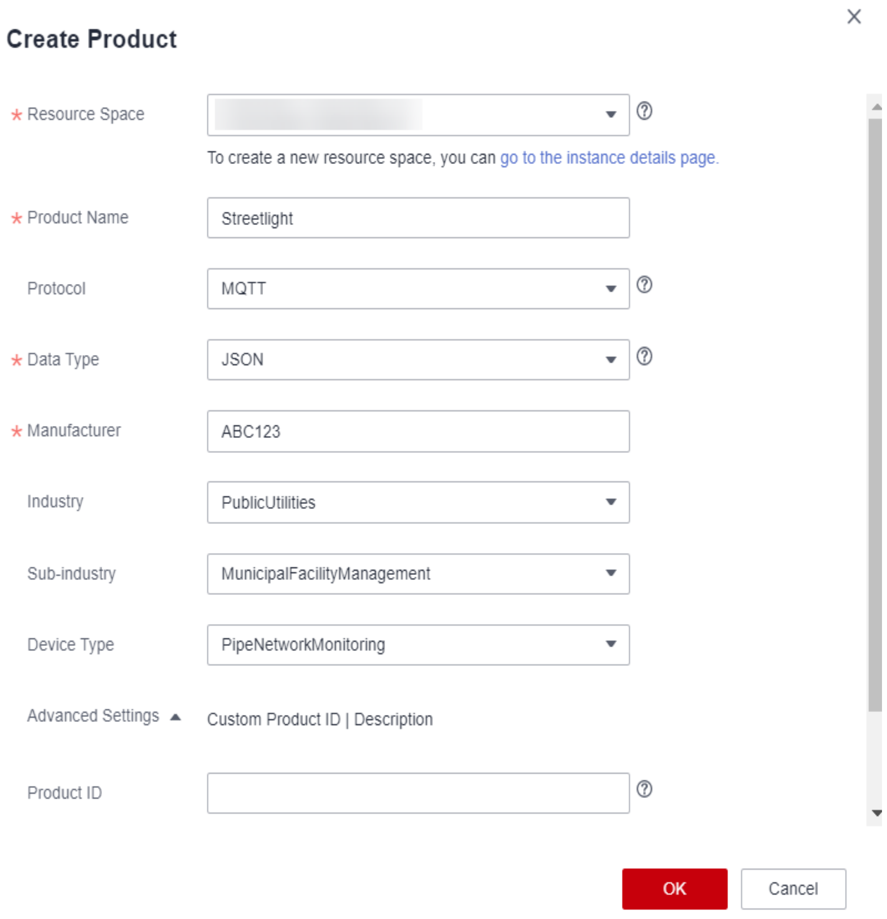
Step 1 Create an MQTT product. (If an MQTT product already exists, skip this step.)

Step 2 Log in to the **console**, choose **Products** in the navigation pane, and click **Create Product** in the upper right corner.

Step 3 Set the parameters as prompted and click **OK**.

Basic Information	
Resource Space	The platform automatically allocates the created product to the default resource space. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create it first.
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Industry	Select the industry to which the product model belongs.
Device Type	If the product model preset on the platform is used, the device type is automatically matched.
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.
Description	Provide a description for the product. Set this parameter based on the site requirements.

Figure 2-41 Creating a product



Create Product

* Resource Space: [redacted]

To create a new resource space, you can go to the instance details page.

* Product Name: Streetlight

Protocol: MQTT

* Data Type: JSON

* Manufacturer: ABC123

Industry: PublicUtilities

Sub-industry: MunicipalFacilityManagement

Device Type: PipeNetworkMonitoring

Advanced Settings ▲ Custom Product ID | Description

Product ID:

----End

Registering a Device

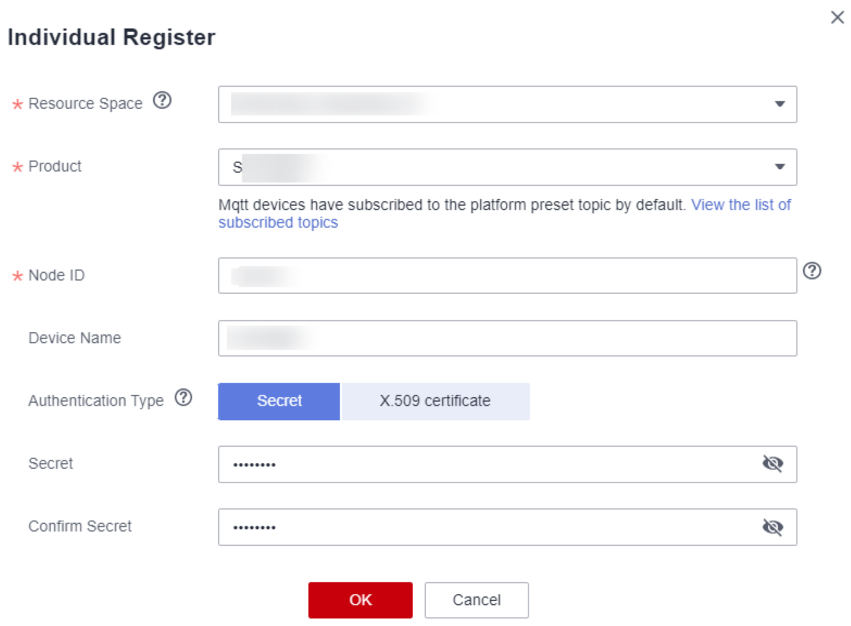
Step 1 On the management console, choose **Devices** > **All Devices** in the navigation pane, and click **Individual Register** in the upper right corner.

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

Parameter	Description
Resource Space	Ensure that the device and the product created in **Creating a Product** belong to the same resource space.
Product	Select the product created in **Creating a Product**.
Node ID	Customize a unique physical identifier for the device. The value can be customized and consists of letters and numbers.
Device Name	Customize the device name.
Authentication Type	Select **Secret**.

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

Figure 2-42 Registering a device



Individual Register

* Resource Space

* Product: S

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID

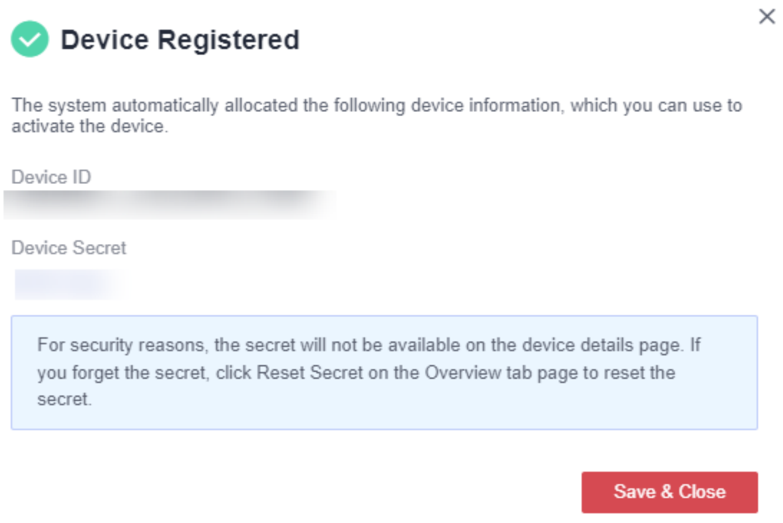
Device Name

Authentication Type: Secret | X.509 certificate

Secret: ••••••••

Confirm Secret: ••••••••

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



Device Registered

The system automatically allocated the following device information, which you can use to activate the device.

Device ID

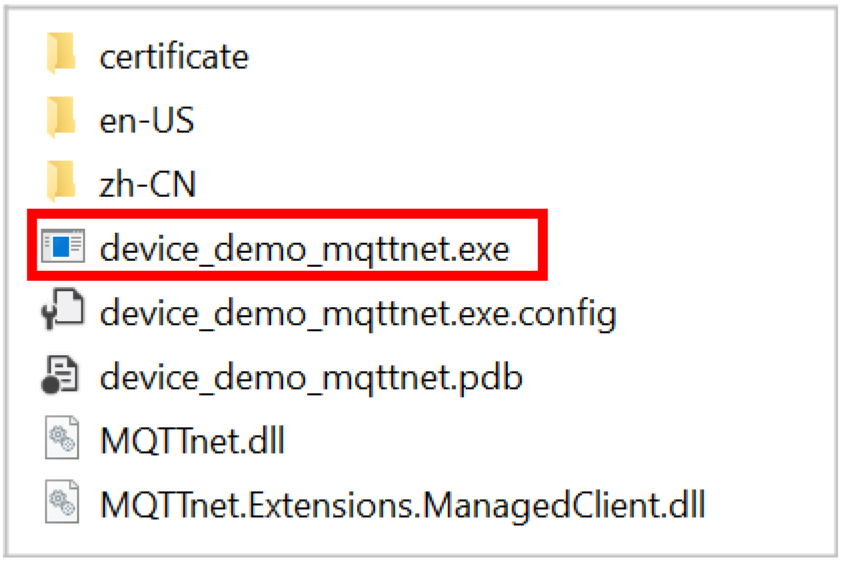
Device Secret

For security reasons, the secret will not be available on the device details page. If you forget the secret, click Reset Secret on the Overview tab page to reset the secret.

----End

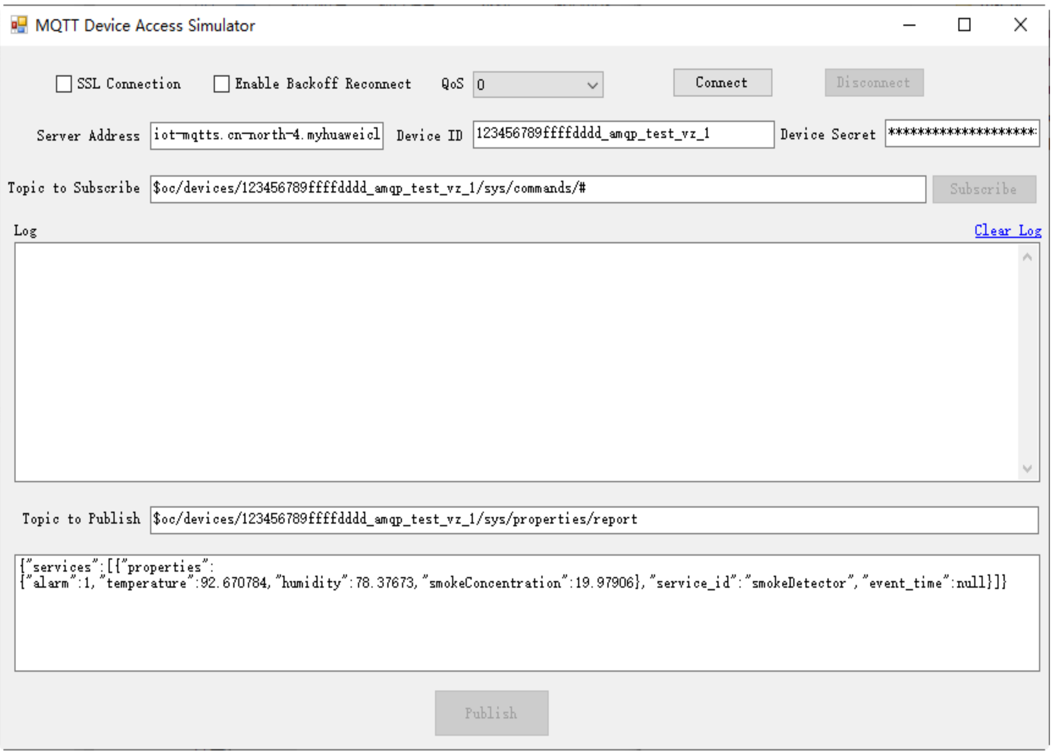
Connecting a Device Simulator to IoTDA

Step 1 Download the **simulator** (for 64-bit operating system by default) and start it.



```
certificate
en-US
zh-CN
device_demo_mqttnet.exe
device_demo_mqttnet.exe.config
device_demo_mqttnet.pdb
MQTTnet.dll
MQTTnet.Extensions.ManagedClient.dll
```

Step 2 Perform operations on the UI.



□ SSL Connection □ Enable Backoff Reconnect QoS 0

Server Address: iot-mqtts.cn-north-4.myhuaweicl

Device ID: 123456789ffffdddd_amqp_test_vz_1

Device Secret: ********************

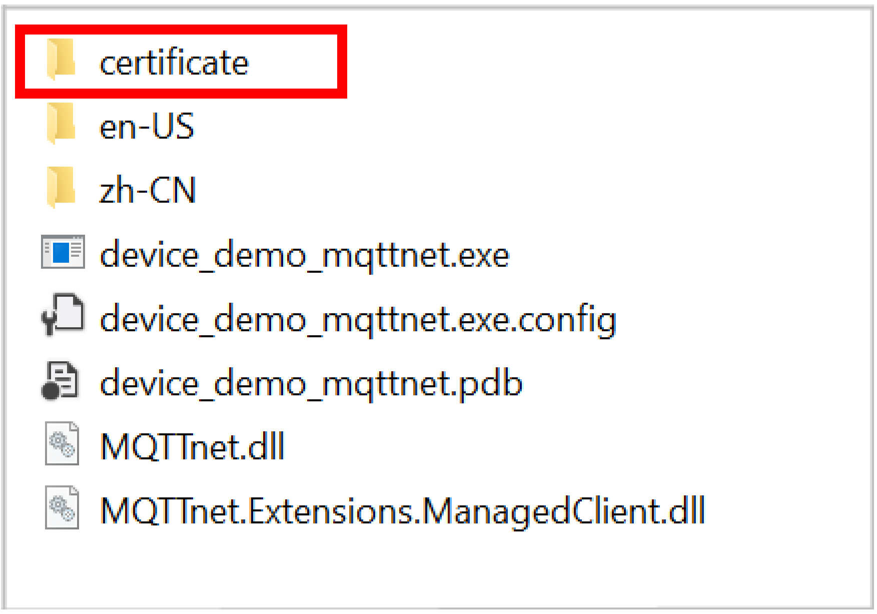
Topic to Subscribe: $oc/devices/123456789ffffdddd_amqp_test_vz_1/sys/commands/#

Topic to Publish: $oc/devices/123456789ffffdddd_amqp_test_vz_1/sys/properties/report

```
{"services":[{"properties":
{"alarm":1,"temperature":92.670784,"humidity":78.37673,"smokeConcentration":19.97906},"service_id":"smokeDetector","event_time":null}]}
```

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.

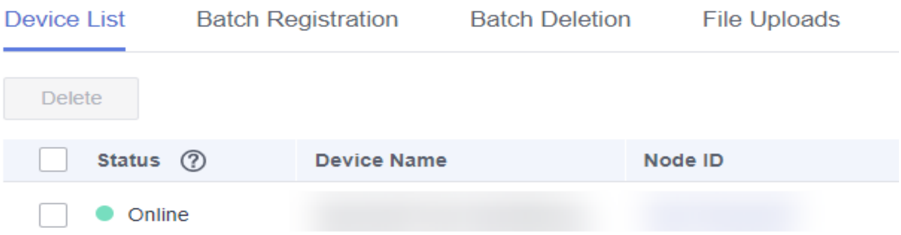


certificate
en-US
zh-CN
device_demo_mqttnet.exe
device_demo_mqttnet.exe.config
device_demo_mqttnet.pdb
MQTTnet.dll
MQTTnet.Extensions.ManagedClient.dll

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.

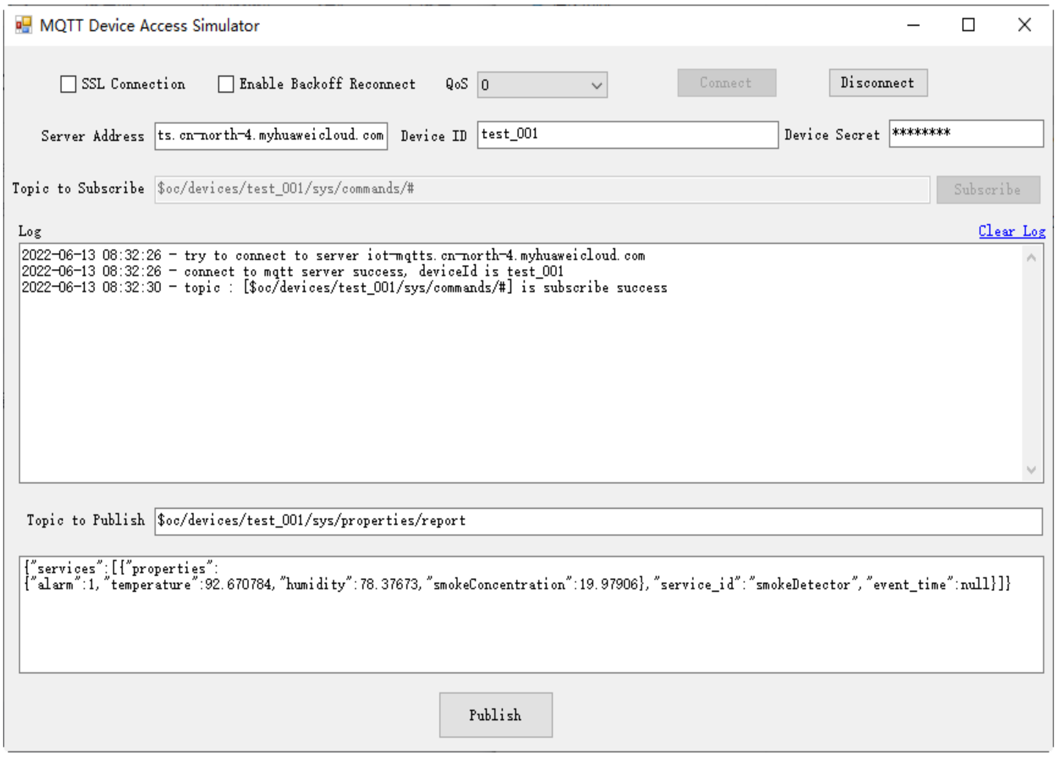


□	Status ⓘ	Device Name	Node ID
□	● Online	[redacted]	[redacted]

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, the following information is displayed in the log area on the home page of the demo:



□ SSL Connection □ Enable Backoff Reconnect QoS 0

Server Address ts.cn-north-4.myhuaweicloud.com Device ID test_001 Device Secret ********

Topic to Subscribe $oc/devices/test_001/sys/commands/#

Log

```
2022-06-13 08:32:26 - try to connect to server iot-mqtts.cn-north-4.myhuaweicloud.com
2022-06-13 08:32:26 - connect to mqtt server success, deviceId is test_001
2022-06-13 08:32:30 - topic : [$oc/devices/test_001/sys/commands/#] is subscribe success
```

Topic to Publish $oc/devices/test_001/sys/properties/report

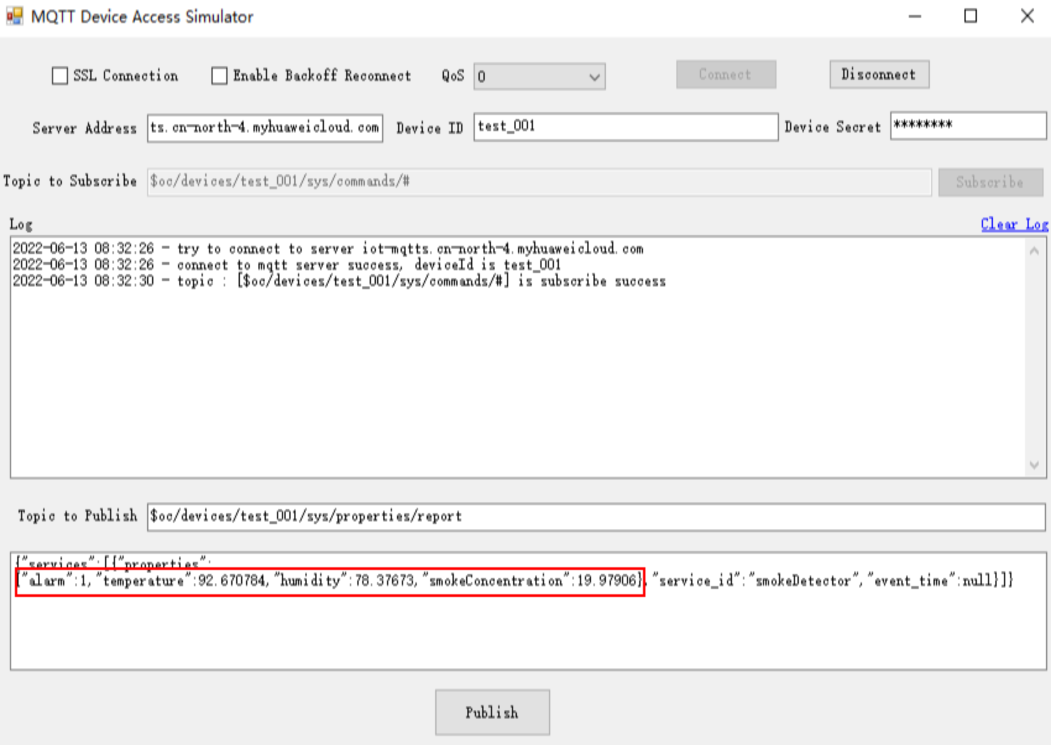
```
{"services":[{"properties":
{"alarm":1,"temperature":92.670784,"humidity":78.37673,"smokeConcentration":19.97906},"service_id":"smokeDetector","event_time":null}]}
```

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.

After a topic is published, the following information is displayed on the demo page:



□ SSL Connection □ Enable Backoff Reconnect QoS 0

Server Address ts.cn-north-4.myhuaweicloud.com Device ID test_001 Device Secret ********

Topic to Subscribe $oc/devices/test_001/sys/commands/#

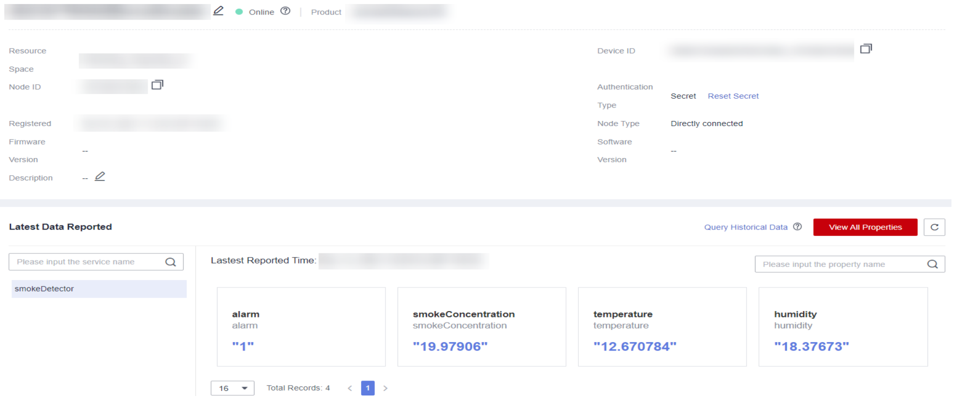
Log

```
2022-06-13 08:32:26 - try to connect to server iot-mqtts.cn-north-4.myhuaweicloud.com
2022-06-13 08:32:26 - connect to mqtt server success, deviceId is test_001
2022-06-13 08:32:30 - topic : [$oc/devices/test_001/sys/commands/#] is subscribe success
```

Topic to Publish $oc/devices/test_001/sys/properties/report

```
{"services":[{"properties":
{"alarm":1,"temperature":92.670784,"humidity":78.37673,"smokeConcentration":19.97906},"service_id":"smokeDetector","event_time":null}]}
```

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



Resource Space
Node ID
Registered
Firmware Version --
Description --
Device ID
Authentication Type: Secret
Node Type: Directly connected
Software Version: --

Latest Data Reported

Please input the service name

smokeDetector

Lastest Reported Time:

Please input the property name

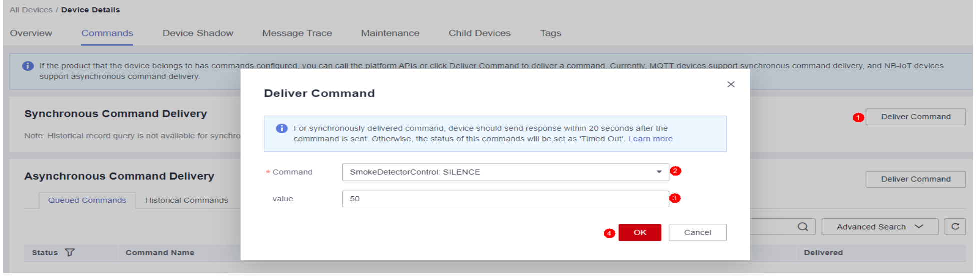
alarm	smokeConcentration	temperature	humidity
alarm	smokeConcentration	temperature	humidity
"1"	"19.97906"	"12.670784"	"18.37673"

16 Total Records: 4 1

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, deliver a command carrying the parameter name **smokeDetector: SILENCE** and parameter value **50**.



If the product that the device belongs to has commands configured, you can call the platform APIs or click Deliver Command to deliver a command. Currently, MQTT devices support synchronous command delivery, and NB-IoT devices support asynchronous command delivery.

Synchronous Command Delivery

Note: Historical record query is not available for synchro

Asynchronous Command Delivery

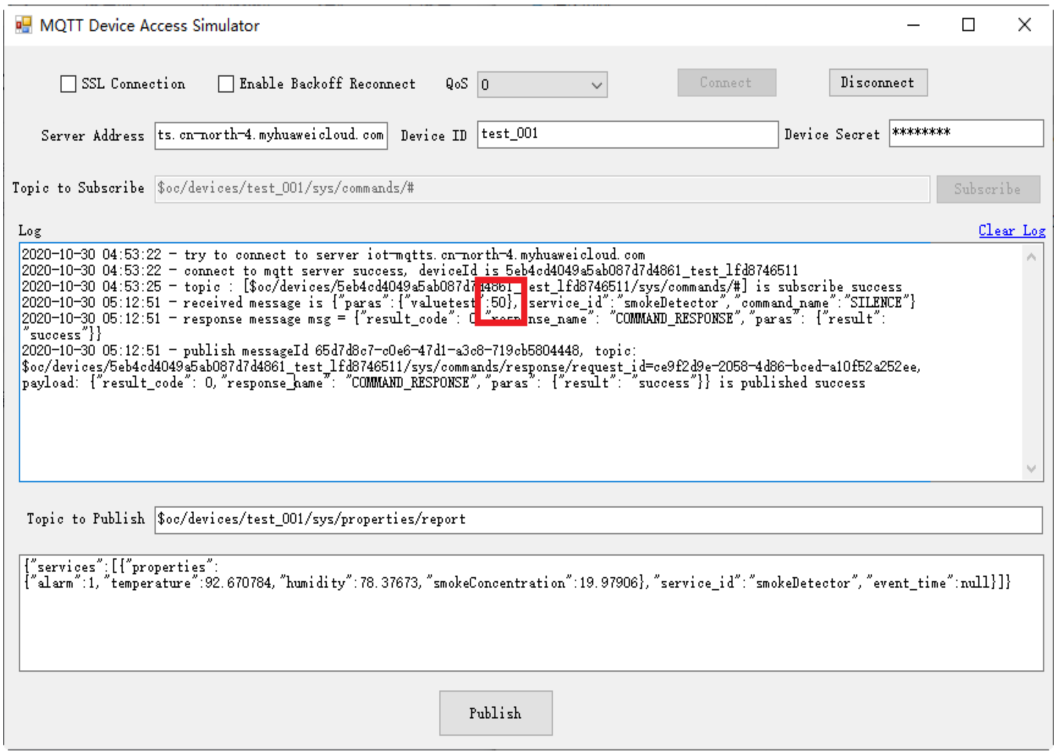
Deliver Command

For synchronously delivered command, device should send response within 20 seconds after the commmand is sent. Otherwise, the status of this commands will be set as 'Timed Out'. Learn more

* Command: SmokeDetectorControl: SILENCE

value: 50

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



□ SSL Connection □ Enable Backoff Reconnect QoS 0

Server Address: ts.cn-north-4.myhuaweicloud.com
Device ID: test_001
Device Secret: ********

Topic to Subscribe: $oc/devices/test_001/sys/commands/#

Log

```
2020-10-30 04:53:22 - try to connect to server iot-mqtts.cn-north-4.myhuaweicloud.com
2020-10-30 04:53:22 - connect to mqtt server success, deviceId is 5eb4cd4049a5ab087d7d4861_test_1fd8746511
2020-10-30 04:53:25 - topic : [$oc/devices/5eb4cd4049a5ab087d7d4861_test_1fd8746511/sys/commands/#] is subscribe success
2020-10-30 05:12:51 - received message is {"paras":{"valuetest":50},"service_id":"smokeDetector","command_name":"SILENCE"}
2020-10-30 05:12:51 - response message msg = {"result_code": 0,"response_name": "COMMAND_RESPONSE","paras": {"result":
"success"}}
2020-10-30 05:12:51 - publish messageId 65d7d8c7-c0e6-47d1-a3c8-719cb5804448, topic:
$oc/devices/5eb4cd4049a5ab087d7d4861_test_1fd8746511/sys/commands/response/request_id=ce9f2d9e-2058-4d86-bced-a10f52a252ee,
payload: {"result_code": 0,"response_name": "COMMAND_RESPONSE","paras": {"result": "success"}} is published success
```

Topic to Publish: $oc/devices/test_001/sys/properties/report

```
{"services":[{"properties":
{"alarm":1,"temperature":92.670784,"humidity":78.37673,"smokeConcentration":19.97906},"service_id":"smokeDetector","event_time":null}]}
```

----End

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

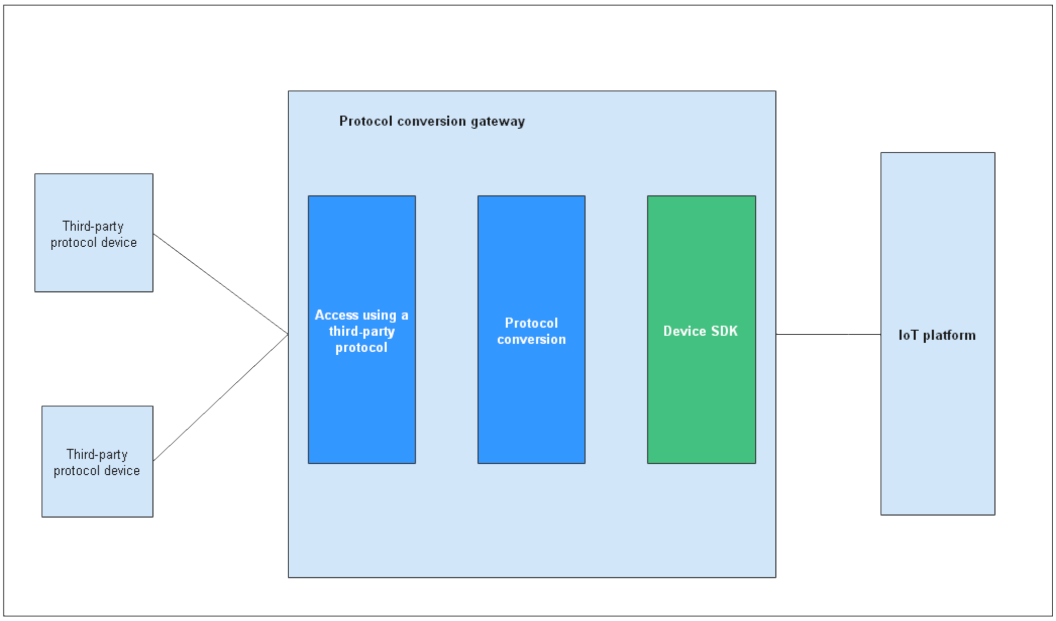
Scenarios

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.

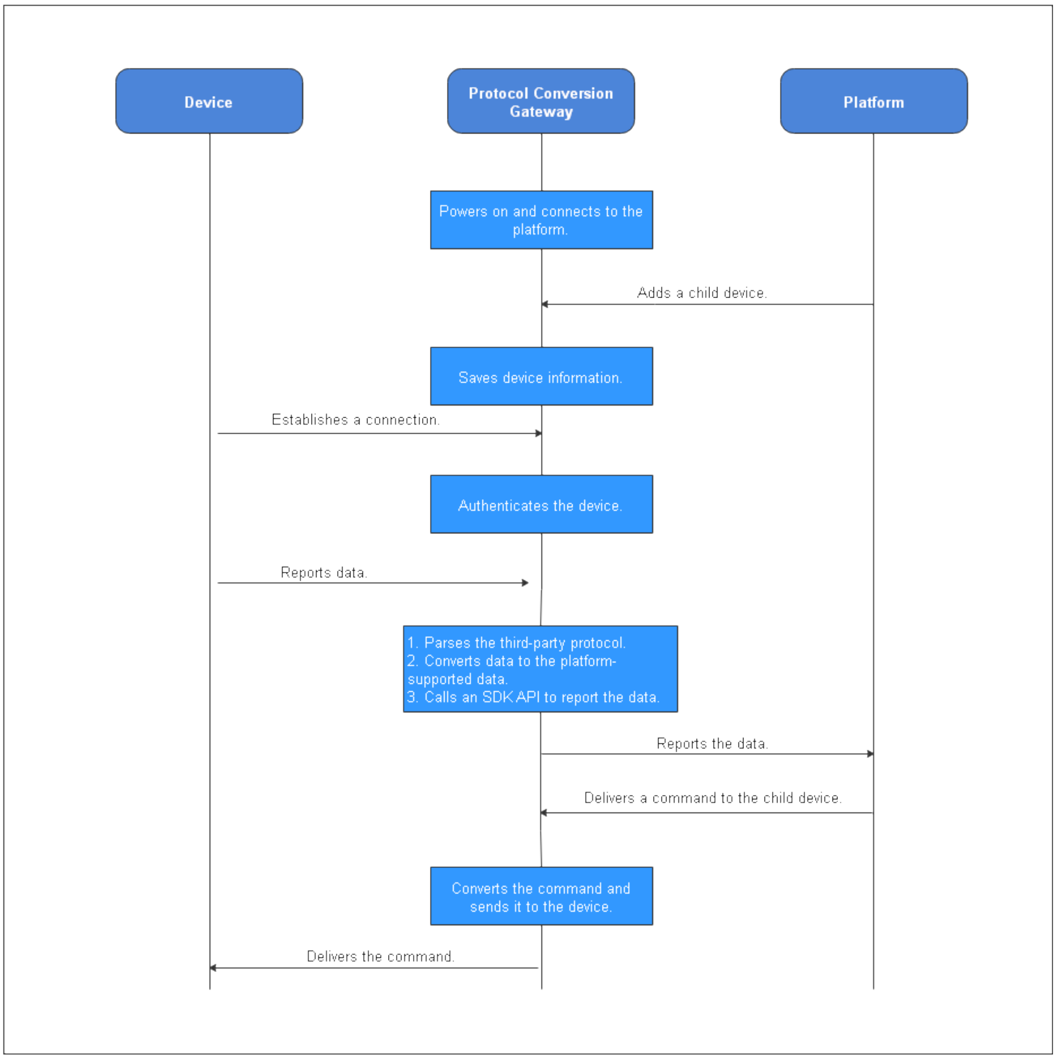

Protocol conversion gateway
Third-party protocol device
Third-party protocol device
Access using a third-party protocol
Protocol conversion
Device SDK
IoT platform

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


Device
Protocol Conversion Gateway
Platform
Powers on and connects to the platform.
Adds a child device.
Saves device information.
Establishes a connection.
Authenticates the device.
Reports data.
1. Parses the third-party protocol.
2. Converts data to the platform-supported data.
3. Calls an SDK API to report the data.
Reports the data.
Delivers a command to the child device.
Converts the command and sends it to the device.
Delivers the command.

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 Constant-Temperature Air Conditioner

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** product page and click **Access Console**.

Step 2 In the navigation pane, click **Products**. Click the drop-down list in the upper right corner, and select the resource space to which the new product belongs.

Step 3 Click **Create Product** in the upper right corner to create a constant-temperature air conditioner product, set the parameters, and click **OK**.

Basic Information	
Product Name	Enter a value, for example, **aircondition**.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value.
Industry	Customize the values.
Device Type	

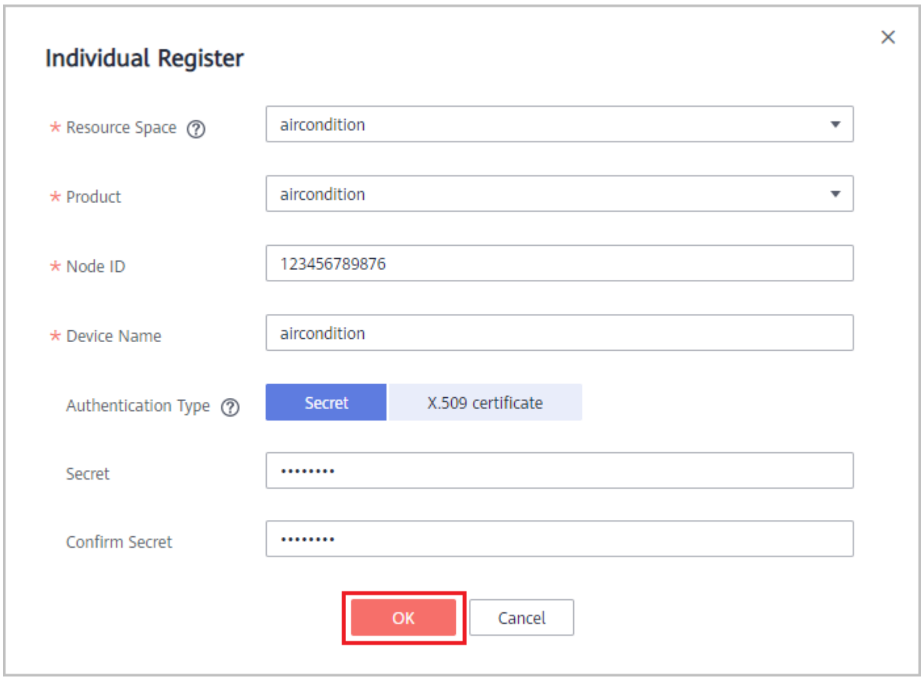
Step 4 After the product is created, click the corresponding product to access its details.

Step 5 On the **Model Definition** tab page, click **Custom Model** and configure the product model based on the table below.

Service data

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

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



Individual Register

* Resource Space: aircondition
* Product: aircondition
* Node ID: 123456789876
* Device Name: aircondition

Authentication Type: Secret

Secret: ••••••••

Confirm Secret: ••••••••

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 **Secret**.
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

----End

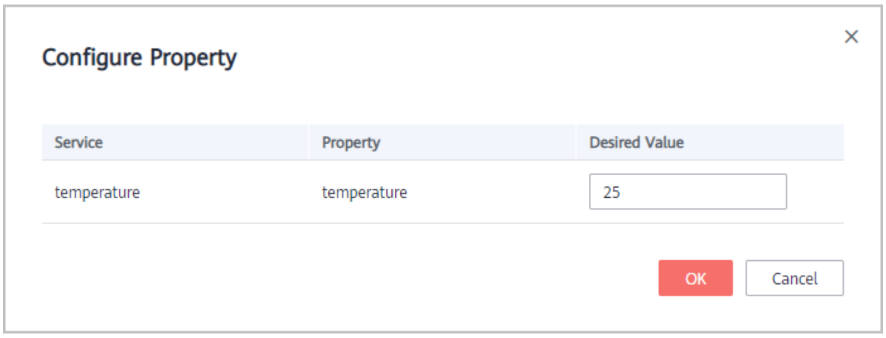
Configuring a Device Shadow

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**.


Configure Property

Service	Property	Desired Value
temperature	temperature	25

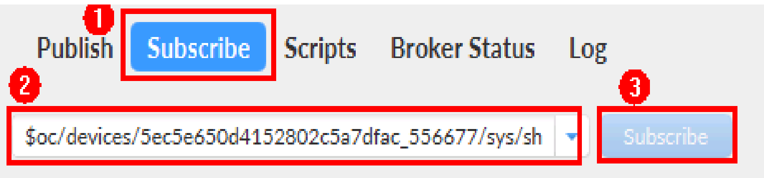
----End

Verifying the Configurations

Method 1:

You can use MQTT.fx to simulate device verification.

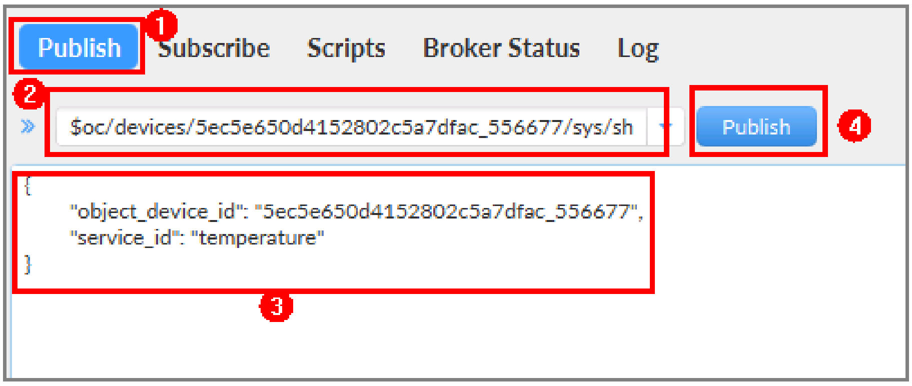
1. Use MQTT.fx to simulate a constant-temperature air conditioner and connect it to the platform. For details, see **Developing an MQTT-based Smart Street Light Online**.
2. On the **Subscribe** tab page, enter **topic=$oc/devices/{device_id}/sys/shadow/get/response/#** of the device shadow (where {device_id}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


$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/sh

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:

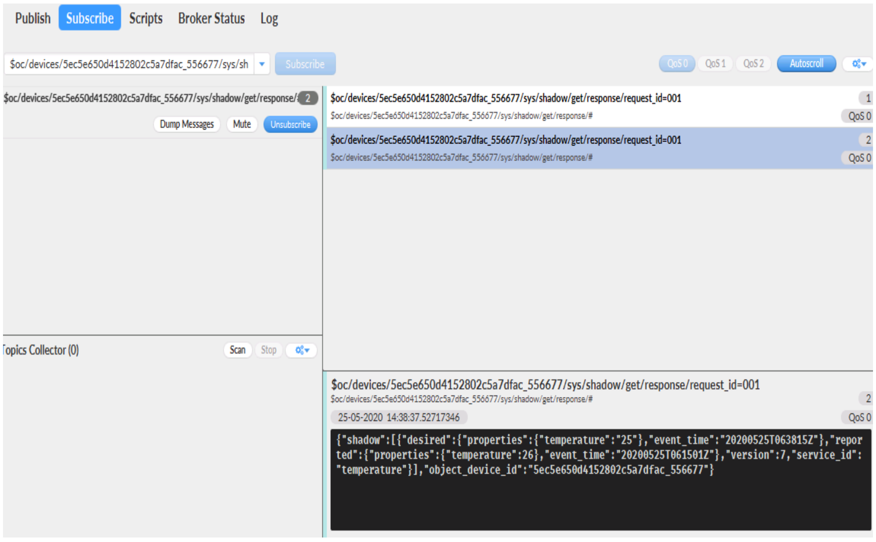
```
{
    "object_device_id": "40fe3542-f4cc-4b6a-98c3-61a49ba1acd4",
  "service_id": temperature"
}
```



```
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/sh
```

```
{
    "object_device_id": "5ec5e650d4152802c5a7dfac_556677",
    "service_id": "temperature"
}
```

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



```
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/sh

$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/# 2

$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/request_id=001   1
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/#   QoS 0
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/request_id=001   2
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/#   QoS 0

$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/request_id=001
$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/shadow/get/response/#   2
25-05-2020 14:38:37.52717346   QoS 0
{"shadow":[{"desired":{"properties":{"temperature":"25"},"event_time":"20200525T063815Z"},"repor
ted":{"properties":{"temperature":26},"event_time":"20200525T061501Z"},"version":7,"service_id":
"temperature"}],"object_device_id":"5ec5e650d4152802c5a7dfac_556677"}
```

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.

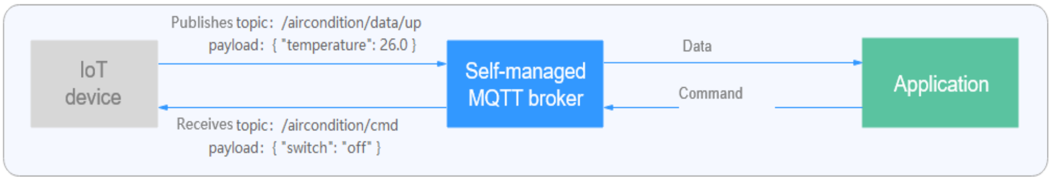
2.7 Migrating a Custom Topic to the Cloud

Scenarios

As the number of IoT devices increases, self-managed IoT networks of enterprises face many problems, such as server capacity expansion, hardware investment, and O&M costs. To address these problems, Huawei Cloud provides a simplified solution to migrate enterprise devices quickly to IoTDA with few changes.

Solution Overview

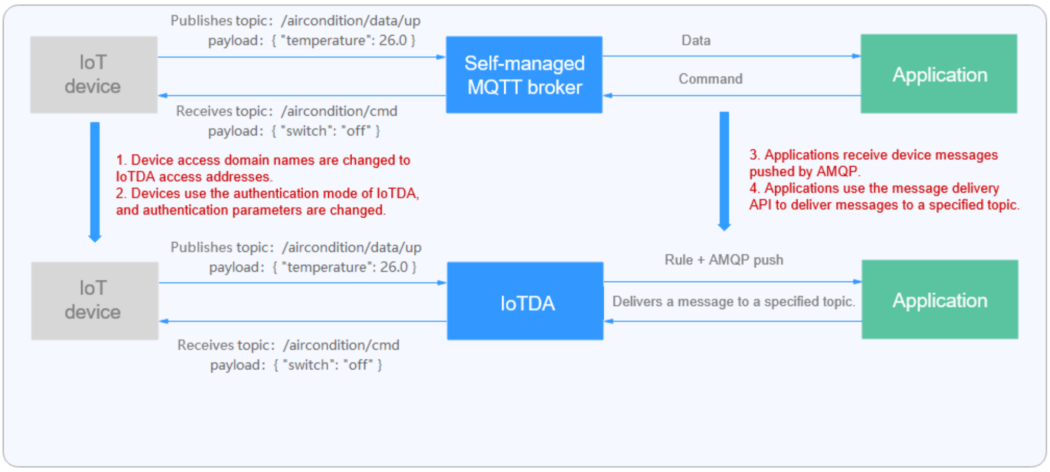
Assume that enterprise devices are connected to the self-managed MQTT cluster. The following figure shows the service architecture.


IoT device
Publishes topic: /aircondition/data/up
payload: { "temperature": 26.0 }
Receives topic: /aircondition/cmd
payload: { "switch": "off" }
Self-managed MQTT broker
Data
Command
Application

The MQTT-based data reporting and command delivery services are defined as follows:

Scenario	Topic	Payload
Device data reporting	/aircondition/data/up	{ "temperature": 26.0 }
Server delivering commands	/aircondition/cmd	{ "switch": "off" }

Huawei Cloud provides the following cost-effective solution to migrate devices to IoTDA without changing the original topic and payload packet formats.


IoT device
Publishes topic: /aircondition/data/up
payload: { "temperature": 26.0 }
Receives topic: /aircondition/cmd
payload: { "switch": "off" }
Self-managed MQTT broker
Data
Command
Application
1. Device access domain names are changed to IoTDA access addresses.
2. Devices use the authentication mode of IoTDA, and authentication parameters are changed.
3. Applications receive device messages pushed by AMQP.
4. Applications use the message delivery API to deliver messages to a specified topic.
IoT device
Publishes topic: /aircondition/data/up
payload: { "temperature": 26.0 }
Receives topic: /aircondition/cmd
payload: { "switch": "off" }
IoTDA
Rule + AMQP push
Delivers a message to a specified topic.
Application

There are three core changes after migrating enterprise devices to the cloud:

- Device access domain names are changed to IoTDA access addresses.
- Rules are configured to forward device data to applications, AMQP consumer group, or Huawei Cloud services.
- Applications use the message delivery API to deliver messages to a specified topic.

Procedure

1. Configure a product, device, and data forwarding rule on the console. For details, see **Cloud Development**.
2. Develop services on the device side. For details, see **Device-side Development**.
3. Develop services on the server side to implement device data receiving and control command delivery. For details, see **Server-side Development**.

Cloud Development

Step 1 Visit the **IoTDA** product page and click **Access Console**.

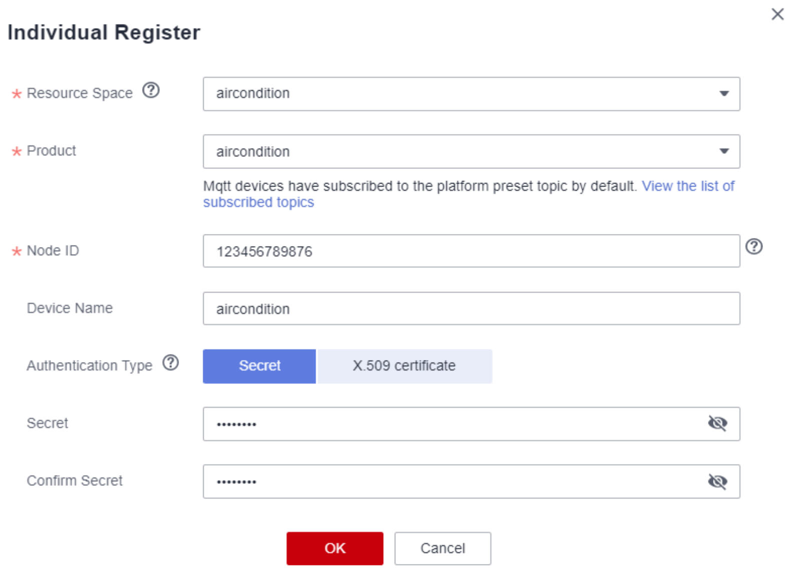
Step 2 Create a product.

1. In the navigation pane, choose **Products**.
2. Click **Create Product** in the upper right corner to create a product to be migrated, set the parameters, and click **OK**.

Basic Information	
Product Name	Enter a value, for example, **aircondition**.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize a name based on the device manufacturer.
Industry	Select **Default**.
Device Type	Set this parameter as required, for example, **aircondition**.

Step 3 Register a device.

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


Individual Register

* Resource Space: aircondition

* Product: aircondition

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: 123456789876

Device Name: aircondition

Authentication Type: Secret | X.509 certificate

Secret: ••••••••

Confirm Secret: ••••••••

Parameter	Description
Product	Select the product created in 2.

Parameter	Description
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 an authentication type based on the existing device authentication type. **Secret** is used in this example.
Secret	Enter the secret of the device to be migrated.

2. Set device parameters and click **OK**.

Step 4 Create a data forwarding rule.

1. In the navigation pane, choose **Rules** > **Data Forwarding**, and click **Create Rule** in the upper right corner.
2. Set the parameters based on the table below and click **Create Rule**.

Parameter	Description
Rule Name	Specify the name of a rule to create.
Description	Describe the rule.
Data Source	Select **Device message**. The device reports data based on the original topic and payload during cloud migration. By default, IoTDA processes messages based on the device message process.
Trigger	After the data source is selected, the platform automatically matches the trigger event.
Resource Space	You can select a single resource space or all resource spaces.

3. Under **Set Forwarding Target**, click **Add**. On the displayed page, set the parameters based on the table below and click **OK**.

Parameter	Description
Forwarding Target	Select a data forwarding target, for example, **AMQP message queue**.

Parameter	Description
Message Queue	Click **Select** to select a message queue. – If no message queue is available, create one. The queue name must be unique under a tenant and can contain 8–128 characters, including letters, numbers, underscores (_), hyphens (-), periods (.), and colons (:). – To delete a message queue, click **Delete** on the right of the message queue. **NOTE** A subscribed queue cannot be deleted.

4. After the rule is defined, click **Enable Rule** to start forwarding data to the AMQP message queue.

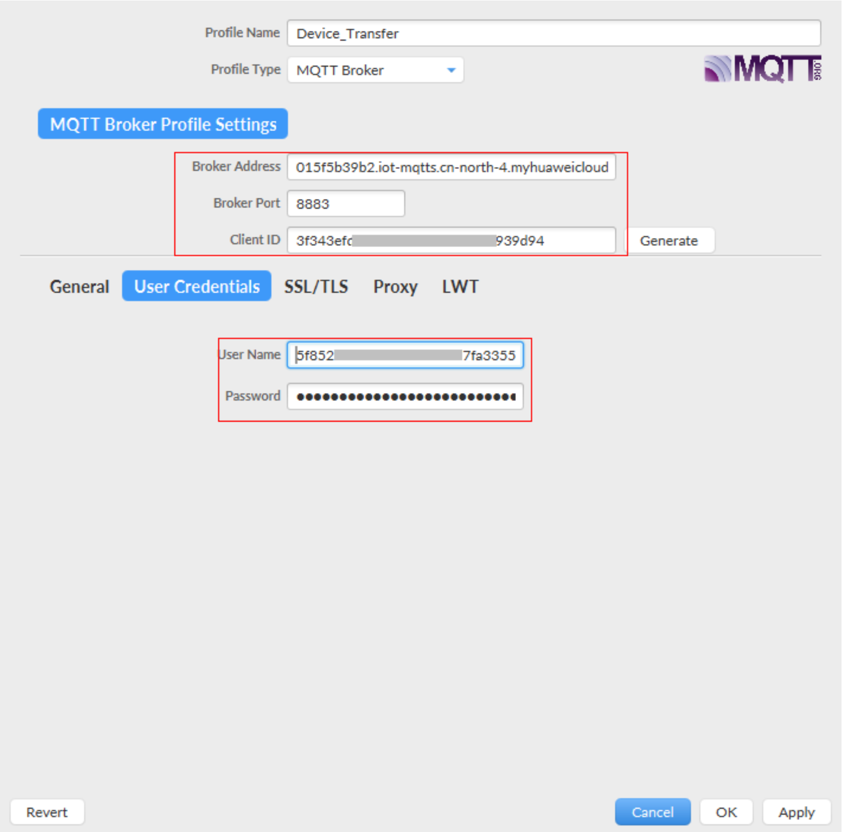
----End

Device-side Development

After the cloud configuration is complete, you need to develop services on the device side. For details about the device development process, see **Development on the Device Side**. This section uses MQTT.fx as an example to describe how to implement MQTT connection establishment, data reporting, and command receiving with minimum modifications on the device side in cloud migration.

Step 1 Establish an MQTT connection between a device and IoTDA.

1. Set authentication parameters by referring to the following table.



Profile Name: Device_Transfer

Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: 015f5b39b2.iot-mqtts.cn-north-4.myhuaweicloud

Broker Port: 8883

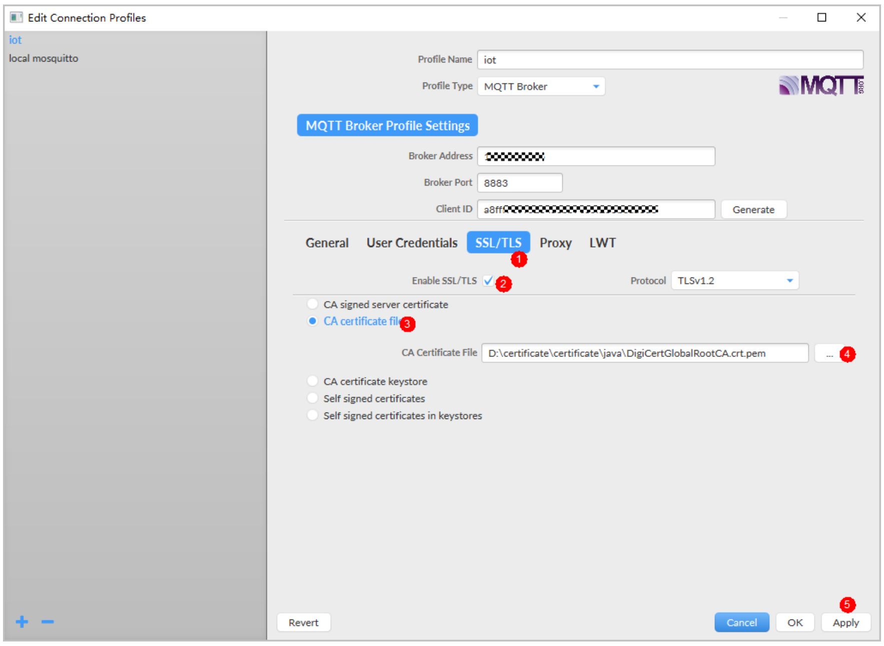
Client ID: 3f343efc[redacted]939d94

User Name: 5f852[redacted]7fa3355

Password: ●●●●●●●●●●●●●●●●●●●●●●●●●●●

Parameter	Mandatory	Description
Broker Address	Yes	IoTDA access address for MQTT devices. Obtain it by referring to **Obtaining Resources**.
Broker Port	Yes	8883.
Client ID	No	Client ID before the migration.
User Name	Yes	Device ID generated during device registration in **3**. By default, the device ID generated on the console is prefixed with the product ID. In the device migration scenario, if the **User Name** parameter on the device side cannot be modified, you can call the API **Creating a Device** to set the device ID to the value of **User Name** before the migration.
Password	Yes	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. The device secret is returned by IoTDA upon successful device registration.

2. Configure the **SSL/TLS** authentication parameters by referring to the following table and click **Apply**.



Edit Connection Profiles

Profile Name: iot
Profile Type: MQTT Broker
Broker Address: [redacted]
Broker Port: 8883
Client ID: a8ff[redacted]

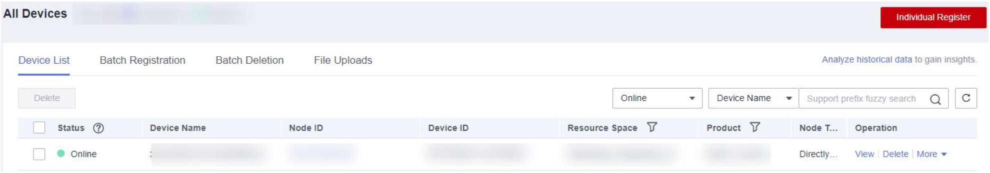
Enable SSL/TLS: ✓
Protocol: TLSv1.2

○ CA signed server certificate
● CA certificate file
CA Certificate File: D:\certificate\certificate\java\DigiCertGlobalRootCA.crt.pem
○ CA certificate keystore
○ Self signed certificates
○ Self signed certificates in keystores

Parameter	Mandatory	Description
Enable SSL/TLS	Yes	Select **Enable SSL/TLS**.

Parameter	Mandatory	Description
CA certificate file	Yes	Upload the CA certificate obtained from **Certificates**.

Step 2 After a device is connected to IoTDA, the device uses the topic and payload format before the migration to report data. For non-predefined topics, IoTDA forwards data reported by devices to third-party applications or other Huawei Cloud services for processing based on the device message processing process.



All Devices [redacted]

Individual Register

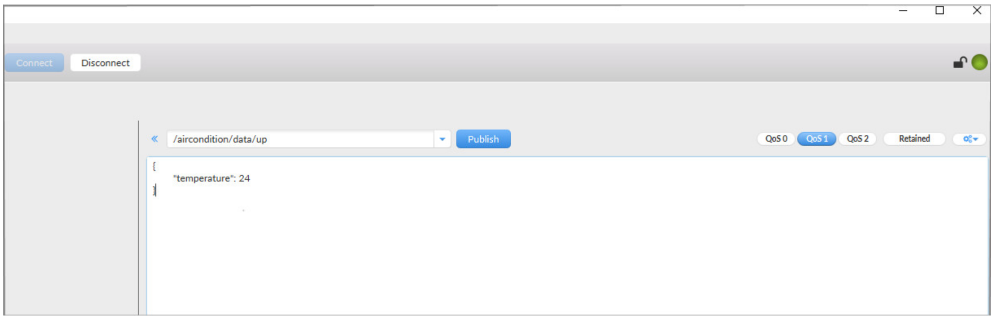
Device List | Batch Registration | Batch Deletion | File Uploads

Analyze historical data to gain insights.

Delete

Online ▾ | Device Name ▾ | Support prefix fuzzy search

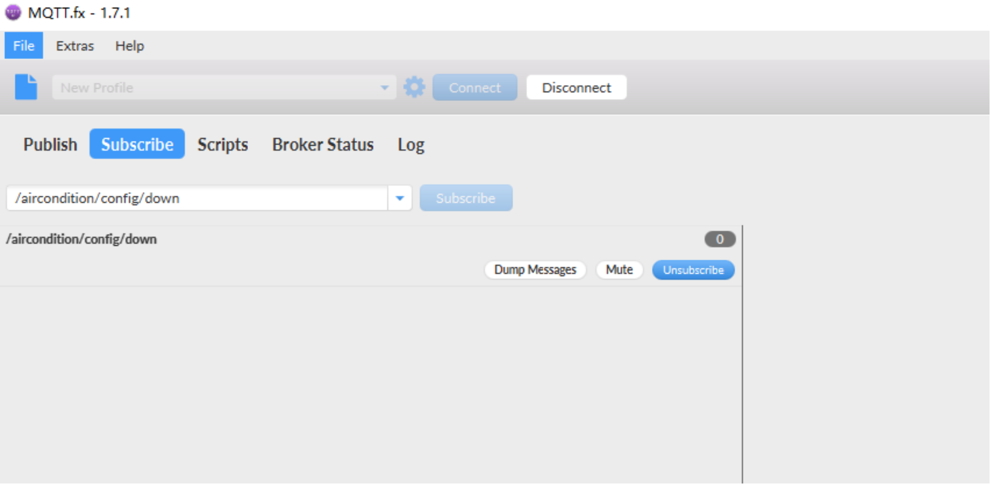
□	Status	Device Name	Node ID	Device ID	Resource Space	Product	Node T...	Operation
□	● Online	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	Directly...	View | Delete | More ▾



/aircondition/data/up

```
{
    "temperature": 24
}
```

Step 3 Subscribe to topics before the migration and receives commands from the application.



MQTT.fx - 1.7.1

/aircondition/config/down

/aircondition/config/down

----End

Server-side Development

Develop services on the server side to implement device data receiving and control command delivery. This section uses a Java script as an example to describe how to receive device data and deliver control commands.

- **Receiving device data**

  The server receives messages using the AMQP client. For details, see **AMQP Client Access** and **Java SDK Access Example**. For details about the AMQP message receiving API, see **Pushing a Device Property Reporting Notification**.

  The following shows an example of the message received by the AMQP client. The **body** field carries the original **topic** and **payload** reported by the device.

```
{
    "resource": "device.message",
    "event": "report",
    "event_time": "20201114T034403Z",
    "notify_data": {
        "header": {
            "app_id": "QAksSBSBBQpWYtEKC3LrrOboNk0a",
            "device_id": "5fae45e358115902ce609882_20201113",
            "node_id": "20201113",
            "product_id": "5fae45e358115902ce609882",
            "gateway_id": "5fae45e358115902ce609882_20201113"
        },
        "body": {
            "topic": "/aircondition/data/up",
            "content": {
                "temperature": 26.0
            }
        }
    }
}
```

  Java core code example:

```
package com.huawei.iot.amqp.jms;

import org.apache.qpid.jms.JmsConnection;
import org.apache.qpid.jms.JmsConnectionFactory;
import org.apache.qpid.jms.JmsConnectionListener;
import org.apache.qpid.jms.message.JmsInboundMessageDispatch;
import org.apache.qpid.jms.transports.TransportOptions;
import org.apache.qpid.jms.transports.TransportSupport;

import javax.jms.*;
import javax.naming.Context;
import javax.naming.InitialContext;
import java.net.URI;
import java.util.Hashtable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.LinkedBlockingQueue;
import java.util.concurrent.ThreadPoolExecutor;
import java.util.concurrent.TimeUnit;

public class HwIotAmqpJavaClientDemo{
    // Asynchronous thread pool. You can adjust the parameters based on service features or use other
asynchronous processing modes.
    private final static ExecutorService executorService = new
ThreadPoolExecutor(Runtime.getRuntime().availableProcessors(),
            Runtime.getRuntime().availableProcessors() * 2, 60,
            TimeUnit.SECONDS, new LinkedBlockingQueue<>(5000));

    public static void main(String[] args) throws Exception{
        // accessKey for the access credential.
        String accessKey = "${yourAccessKey}";
        long timeStamp = System.currentTimeMillis();
        // Method to assemble userName. For details, see AMQP Client Access.
        String userName = "accessKey=" + accessKey + "|timestamp=" + timeStamp;
        // accessCode for the access credential.
        String password = "${yourAccessCode}";
        // Assemble the connection URL according to the qpid-jms specifications.
        String connectionUrl = "amqps://${UUCID}.iot-amqps.cn-north-4.myhuaweicloud.com:5671?
amqp.vhost=default&amqp.idleTimeout=8000&amqp.saslMechanisms=PLAIN";
        Hashtable<String, String> hashtable = new Hashtable<>();
        hashtable.put("connectionfactory.HwConnectionURL", connectionUrl);
        // Queue name. You can use DefaultQueue.
        String queueName = "${yourQueue}";
        hashtable.put("queue.HwQueueName", queueName);
        hashtable.put(Context.INITIAL_CONTEXT_FACTORY,
"org.apache.qpid.jms.jndi.JmsInitialContextFactory");
        Context context = new InitialContext(hashtable);
        JmsConnectionFactory cf = (JmsConnectionFactory) context.lookup("HwConnectionURL");
        // Multiple queues can be created for one connection. Match queue.HwQueueName with
queue.HwQueueName.
        Destination queue = (Destination) context.lookup("HwQueueName");

        // Trust the server.
        TransportOptions to = new TransportOptions(); to.setTrustAll(true);
        cf.setSslContext(TransportSupport.createJdkSslContext(to));

        // Create a connection.
        Connection connection = cf.createConnection(userName, password);
        ((JmsConnection) connection).addConnectionListener(myJmsConnectionListener);
        // Create a session.
        // Session.CLIENT_ACKNOWLEDGE: After receiving a message, manually call
message.acknowledge().
        // Session.AUTO_ACKNOWLEDGE: The SDK automatically responds with an ACK message.
(recommended processing)
        Session session = connection.createSession(false, Session.AUTO_ACKNOWLEDGE);
        connection.start();
        // Create a receiver link.
        MessageConsumer consumer = session.createConsumer(queue);
        // Messages can be processed in either of the following ways:
        // 1. Proactively pull data (recommended processing). For details, see receiveMessage(consumer).
        // 2. Add a listener. For details, see consumer.setMessageListener(messageListener). The server
proactively pushes data to the client at an acceptable data rate.
        receiveMessage(consumer);
        // consumer.setMessageListener(messageListener);
    }

    private static void receiveMessage(MessageConsumer consumer) throws JMSException{
        while (true){
            try{
                // It is recommended that received messages be processed asynchronously. Ensure that the
receiveMessage function does not contain time-consuming logic.
                Message message = consumer.receive(); processMessage(message);
            } catch (Exception e) {
                System.out.println("receiveMessage hand an exception: " + e.getMessage());
                e.printStackTrace();
            }
        }

    }

    private static MessageListener messageListener = new MessageListener(){
        @Override
        public void onMessage(Message message){
            try {
                // It is recommended that received messages be processed asynchronously. Ensure that the
onMessage function does not contain time-consuming logic.
                // If the service processing takes a long time and blocks the thread, the normal callback
after the SDK receives the message may be affected.
                executorService.submit(() -> processMessage(message));
            } catch (Exception e){
                System.out.println("submit task occurs exception: " + e.getMessage());
                e.printStackTrace();
            }
        }
    };

    /**
```

```
     * Service logic for processing the received messages
     */
    private static void processMessage(Message message) {
        try {
            String body = message.getBody(String.class); String content = new String(body);
            System.out.println("receive an message, the content is " + content);
        } catch (Exception e){
            System.out.println("processMessage occurs error: " + e.getMessage());
            e.printStackTrace();
        }
    }

    private static JmsConnectionListener myJmsConnectionListener = new JmsConnectionListener(){
        /**
         * Connection established.
         */
        @Override
        public void onConnectionEstablished(URI remoteURI){
            System.out.println("onConnectionEstablished, remoteUri:" + remoteURI);
        }

        /**
         * The connection fails after the maximum number of retries is reached.
         */
        @Override
        public void onConnectionFailure(Throwable error){
            System.out.println("onConnectionFailure, " + error.getMessage());
        }

        /**
         * Connection interrupted.
         */
        @Override
        public void onConnectionInterrupted(URI remoteURI){
            System.out.println("onConnectionInterrupted, remoteUri:" + remoteURI);
        }

        /**
         * Automatic reconnection.
         */
        @Override
        public void onConnectionRestored(URI remoteURI){
            System.out.println("onConnectionRestored, remoteUri:" + remoteURI);
        }

        @Override
        public void onInboundMessage(JmsInboundMessageDispatch envelope){
            System.out.println("onInboundMessage, " + envelope);
        }

        @Override
        public void onSessionClosed(Session session, Throwable cause){
            System.out.println("onSessionClosed, session=" + session + ", cause =" + cause);
        }

        @Override
        public void onConsumerClosed(MessageConsumer consumer, Throwable cause){
            System.out.println("MessageConsumer, consumer=" + consumer + ", cause =" + cause);
        }

        @Override
        public void onProducerClosed(MessageProducer producer, Throwable cause){
            System.out.println("MessageProducer, producer=" + producer + ", cause =" + cause);
        }
    };
}
```

- **Delivering control commands**

An application calls the message delivery API to deliver control commands. For details, see **Delivering a Message to a Device**.

Key parameters for device message delivery are as follows.

Parameter	Mandatory	Type	Location	Description
X-Auth-Token	Yes	String	Header	User token. You can obtain the token by calling the IAM API **Obtaining a User Token Through Password Authentication**. **X-Subject-Token** in the response header returned by the API is the desired user token. For details about how to obtain the token, see **Token Authentication**.
Instance-Id	No	String	Header	Instance ID. This parameter is required only when the API is called from the management plane in the physical multi-tenant scenario.
project_id	Yes	String	Path	Project ID. For details on how to obtain a project ID, see **Obtaining a Project ID**.
device_id	Yes	String	Path	ID of the device to be migrated.
message	Yes	String	Body	Message executed by the device. The value is a string. The specific format depends on the application and device.
topic_full_name	No	String(128)	Body	Topic of the device to be migrated.

Example response:

```
POST https://{Endpoint}/v5/iot/{project_id}/devices/{device_id}/messages
Content-Type: application/json
X-Auth-Token: ********
Instance-Id: ********

{
  "message": "{\"switch\":\"off\"}",
  "topic_full_name": "/aircondition/cmd"

}
```

Java core code example:

```
public class DeviceMessage {
   public static void main(String[] args) throws KeyManagementException,
NoSuchAlgorithmException, IOException {
        String token = Authentication.getToken();
        Map<String, String> headers = new HashMap<String, String>();
        headers.put("Content-Type", "application/json");
        headers.put("X-Auth-Token", token);

        Message message = new Message();
        message.setMessage("{\"switch\" :\"off\"}");
        message.setTopic_full_name("/aircondition/cmd");

        String projectId = "11111";
        String deviceId = "5fae45e358115902ce609882_20201113";
        String url = "https://iotda.cn-north-4.myhuaweicloud.com/v5/iot/%s/devices/%s/messages";
        url = String.format(url, projectId, deviceId);
        HttpUtils httpUtils = new HttpUtils();
        httpUtils.initClient();
        StreamClosedHttpResponse httpResponse = httpUtils.doPost(url, headers,
JsonUtils.Obj2String(message));
        System.out.println(httpResponse.getContent());
    }
}
```

2.8 Connecting and Debugging an NB-IoT Device Simulator

Scenarios

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

(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** product page and click **Access Console**.

Step 2 In the navigation pane, choose **Resource Spaces** and click **Create Resource Space** in the upper right corner.

Step 3 In the displayed dialog box, specify **Space Name** and click **OK**.

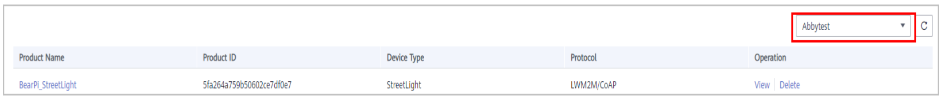
----End

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** product page and click **Access Console**.

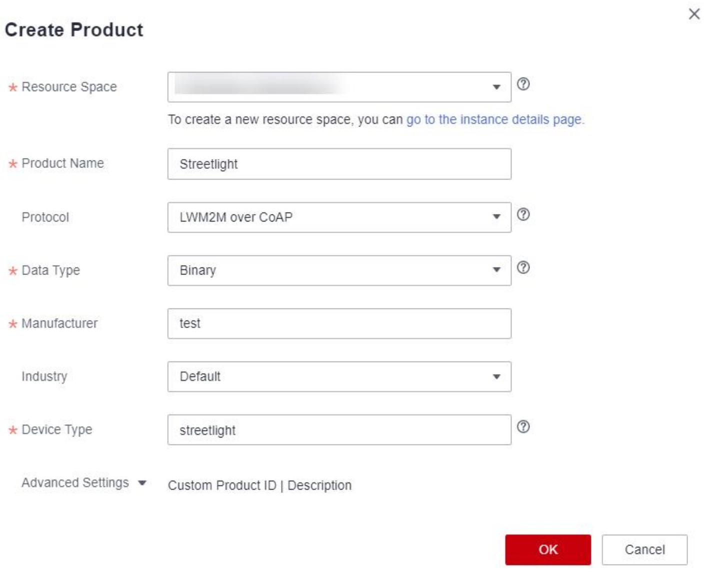
Step 2 In the navigation pane, choose **Products**.



Abbytest

Product Name	Product ID	Device Type	Protocol	Operation
BearPi_StreetLight	5fa264a759b50602ce7df0e7	StreetLight	LWM2M/CoAP	View | Delete

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



Create Product

* Resource Space:
To create a new resource space, you can go to the instance details page.

* Product Name: Streetlight

Protocol: LWM2M over CoAP

* Data Type: Binary

* Manufacturer: test

Industry: Default

* Device Type: streetlight

Advanced Settings: Custom Product ID | Description

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

Industry	Select **Default**.
Device Type	Enter **streetlight**.

----End

Defining a Product Model

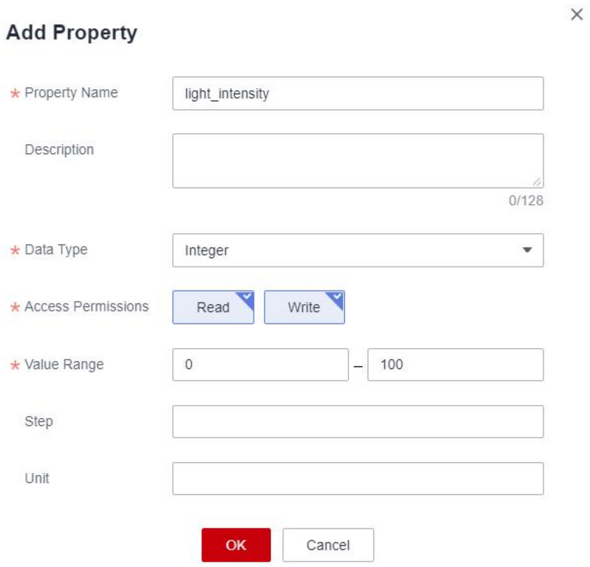
Step 1 Click the name of the product created in 3 to go to the product details page.

Step 2 On the **Model Definition** tab page, click **Custom Model** to add a service for the product.

- **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 0–100 (light intensity range).


Add Property

* Property Name: light_intensity

Description: 0/128

* Data Type: Integer

* Access Permissions: Read Write

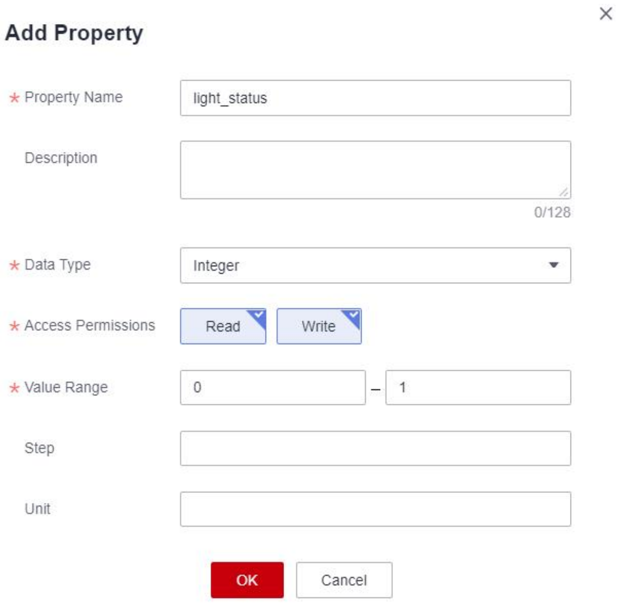
* Value Range: 0 – 100

Step

Unit

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 0–1. **0** indicates that the light is turned off. **1** indicates that the light is turned on.



Add Property

* Property Name: light_status

Description: 0/128

* Data Type: Integer

* Access Permissions: Read, Write

* Value Range: 0 – 1

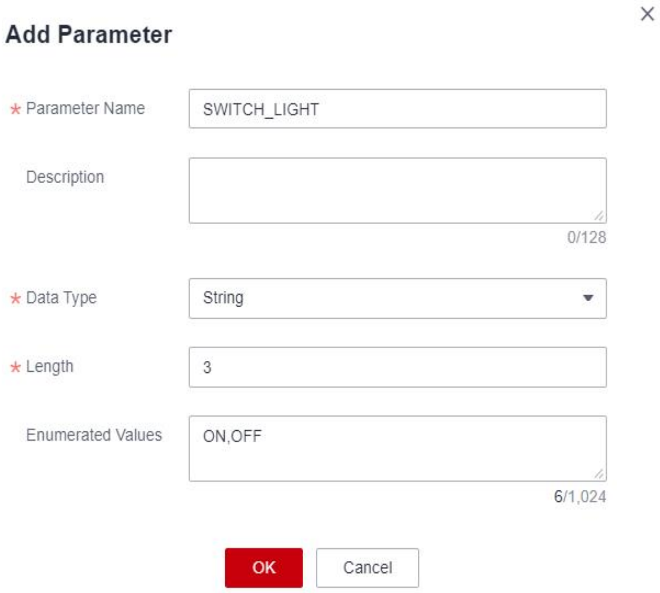
Step:

Unit:

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 2-43 Adding a command parameter



Add Parameter

* Parameter Name: SWITCH_LIGHT

Description: 0/128

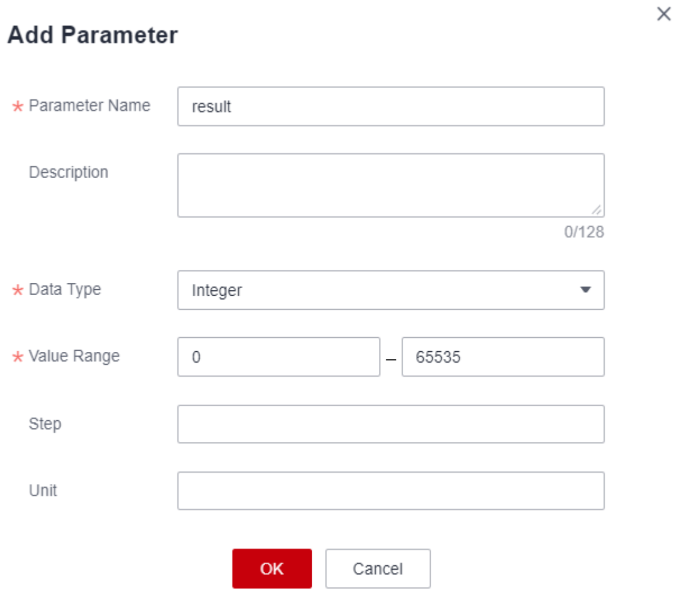
* Data Type: String

* Length: 3

Enumerated Values: ON,OFF 6/1,024

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

Figure 2-44 Adding a response parameter



Add Parameter

* Parameter Name: result
Description:
0/128
* Data Type: Integer
* Value Range: 0 – 65535
Step:
Unit:

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.

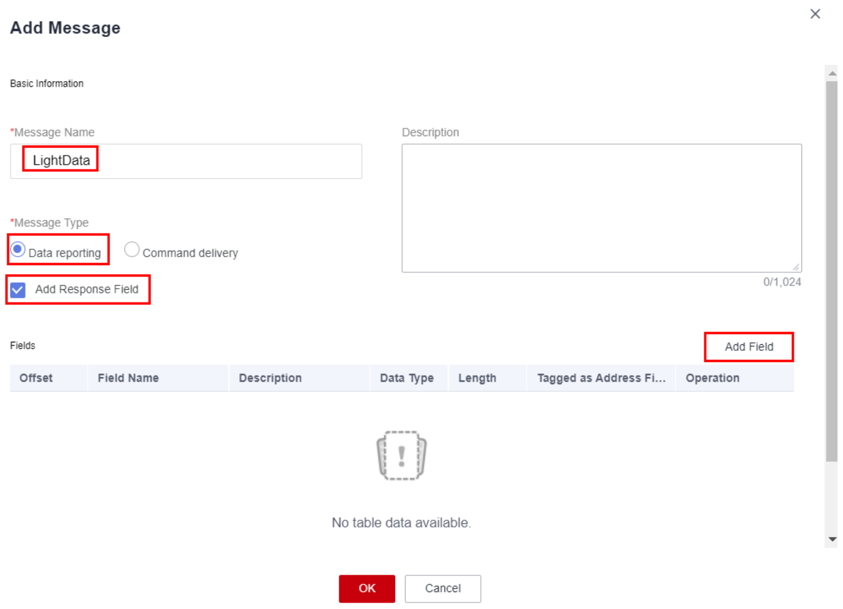
NOTE

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**.



Add Message

Basic Information

*Message Name: LightData

Description: 0/1,024

*Message Type: ● Data reporting ○ Command delivery

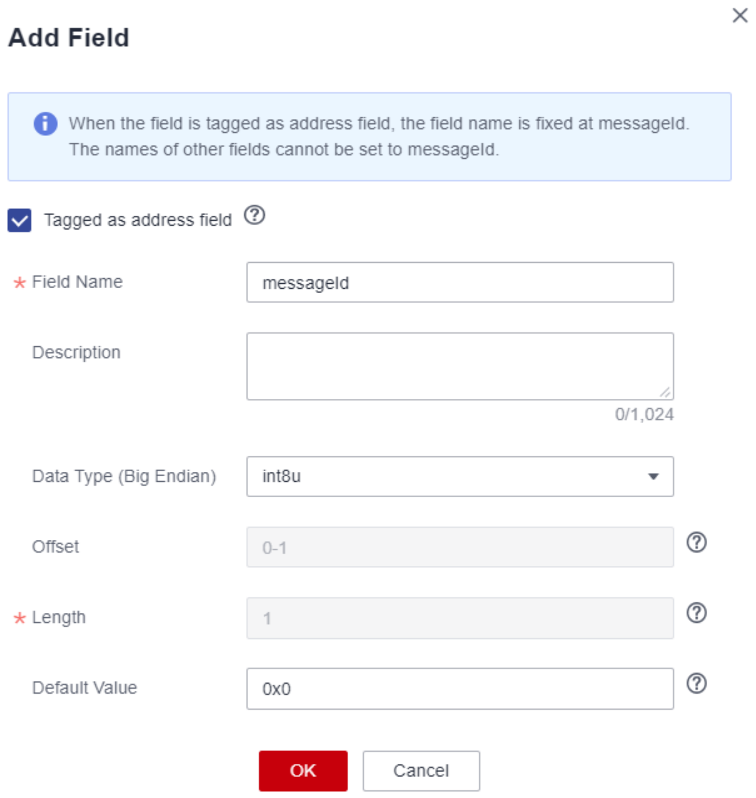
☑ Add Response Field

Fields

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

No table data available.

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



Add Field

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: messageId

Description: 0/1,024

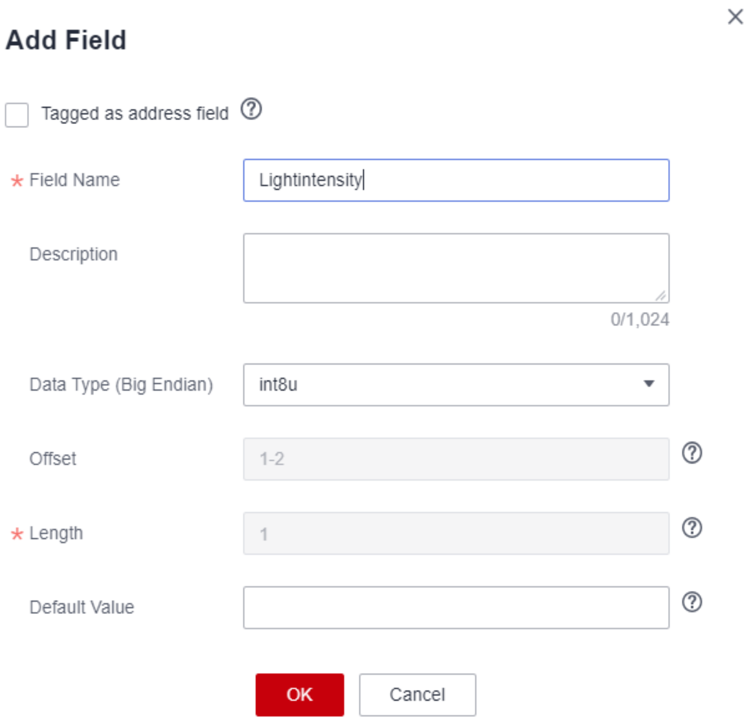
Data Type (Big Endian): int8u

Offset: 0-1

* Length: 1

Default Value: 0x0

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



Add Field

□ Tagged as address field

* Field Name: Lightintensity

Description:

0/1,024

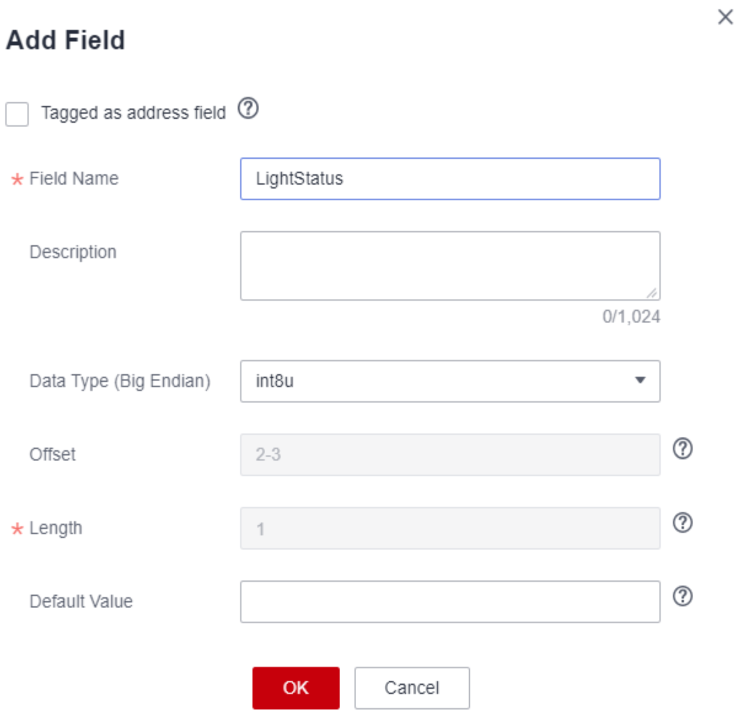
Data Type (Big Endian): int8u

Offset: 1-2

* Length: 1

Default Value:

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**.



Add Field

□ Tagged as address field

* Field Name: LightStatus

Description:

0/1,024

Data Type (Big Endian): int8u

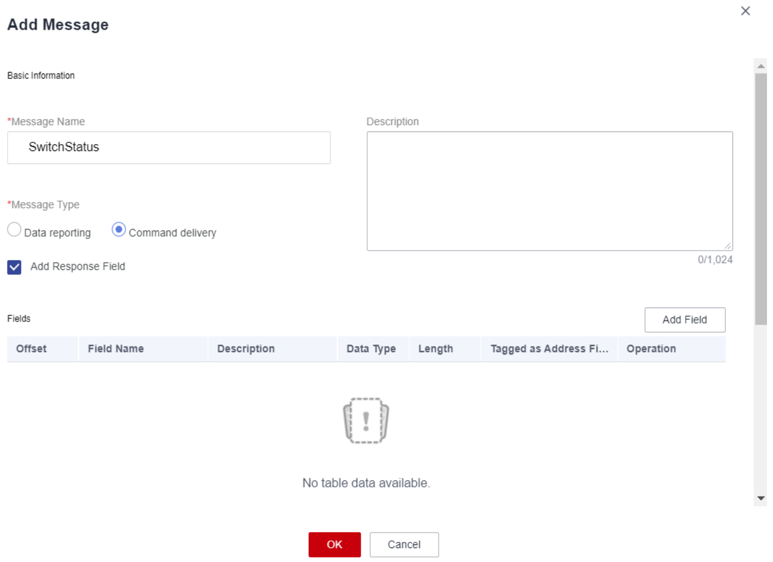
Offset: 2-3

* Length: 1

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.



Add Message

Basic Information

*Message Name: SwitchStatus

Description: 0/1,024

*Message Type: ○ Data reporting ● Command delivery

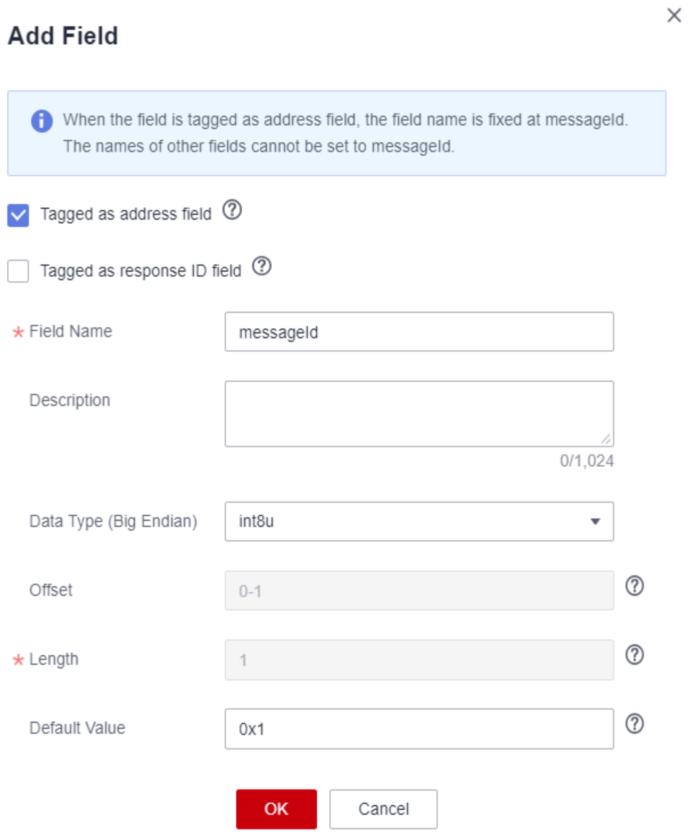
☑ Add Response Field

Fields

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

No table data available.

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



Add Field

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

* Field Name: messageId

Description: 0/1,024

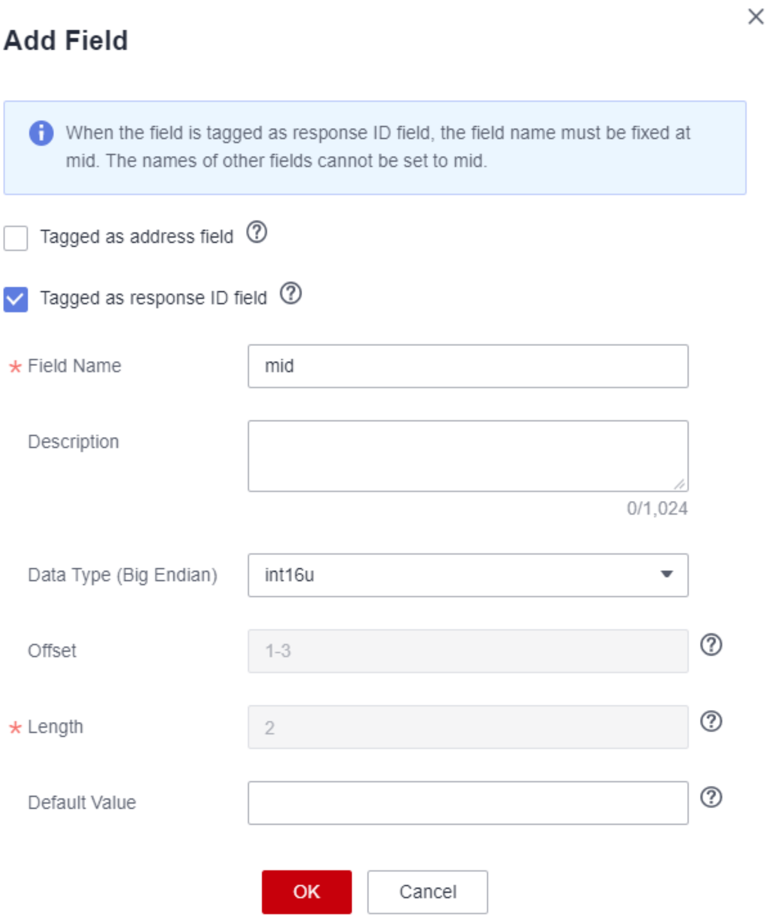
Data Type (Big Endian): int8u

Offset: 0-1

* Length: 1

Default Value: 0x1

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


Add Field

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: mid

Description: 0/1,024

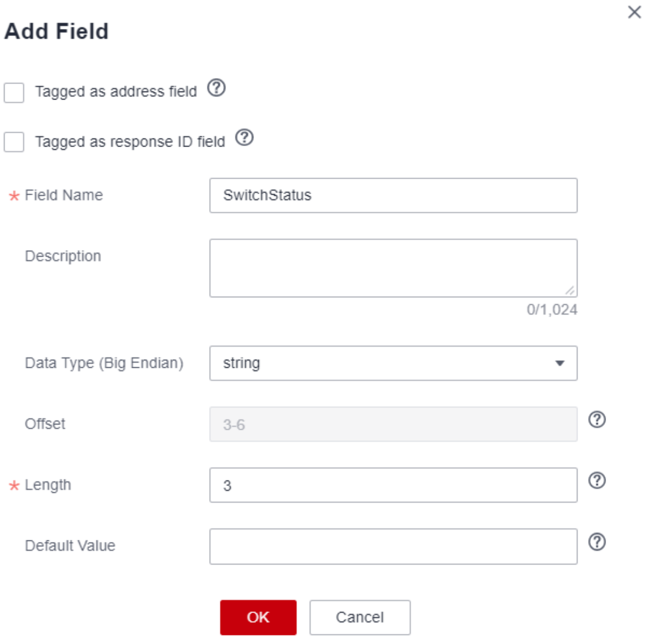
Data Type (Big Endian): int16u

Offset: 1-3

* Length: 2

Default Value:

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


Add Field

□ Tagged as address field

□ Tagged as response ID field

* Field Name: SwitchStatus

Description: 0/1,024

Data Type (Big Endian): string

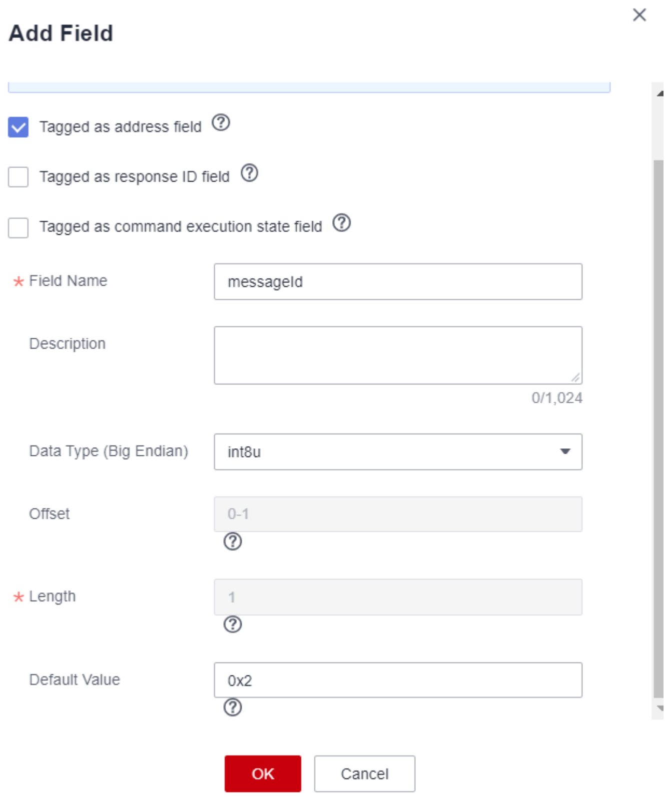
Offset: 3-6

* Length: 3

Default Value:

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.



Add Field

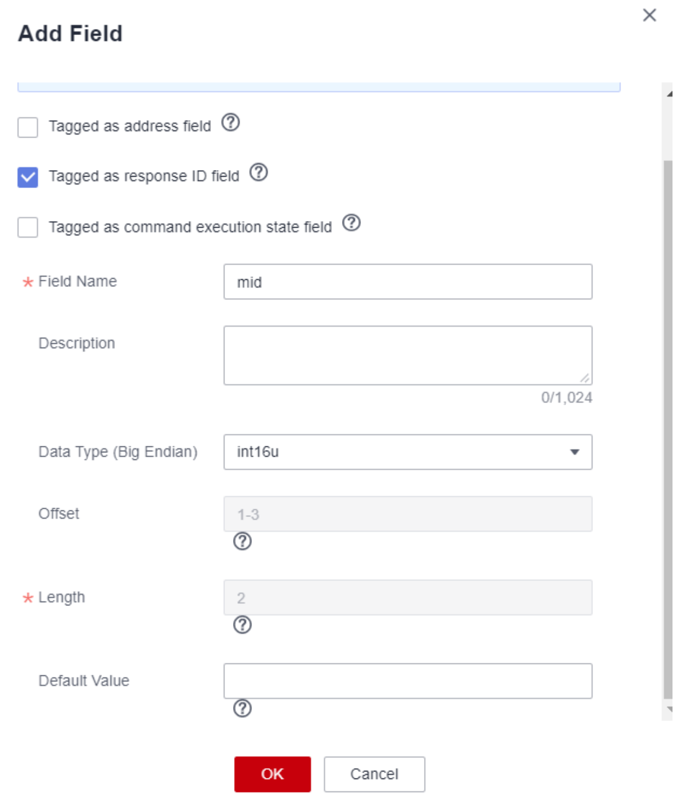
☑ Tagged as address field

□ Tagged as response ID field

□ Tagged as command execution state field

* Field Name	messageId
Description	0/1,024
Data Type (Big Endian)	int8u
Offset	0-1
* Length	1
Default Value	0x2

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



Add Field

□ Tagged as address field

☑ Tagged as response ID field

□ Tagged as command execution state field

* Field Name: mid

Description: 0/1,024

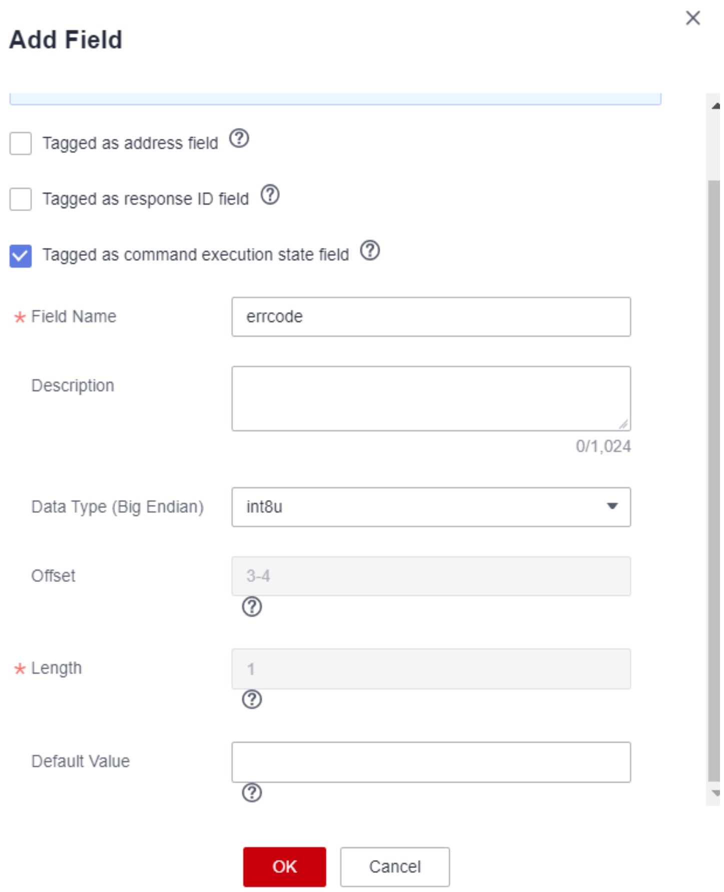
Data Type (Big Endian): int16u

Offset: 1-3

* Length: 2

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.



Add Field

□ Tagged as address field

□ Tagged as response ID field

☑ Tagged as command execution state field

* Field Name: errcode

Description: 0/1,024

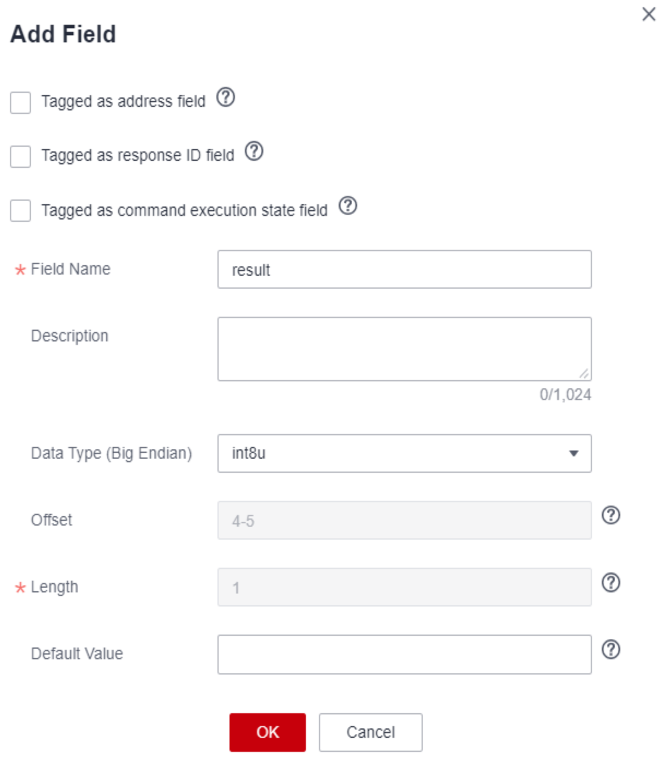
Data Type (Big Endian): int8u

Offset: 3-4

* Length: 1

Default Value:

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



Add Field

□ Tagged as address field

□ Tagged as response ID field

□ Tagged as command execution state field

* Field Name: result

Description: 0/1,024

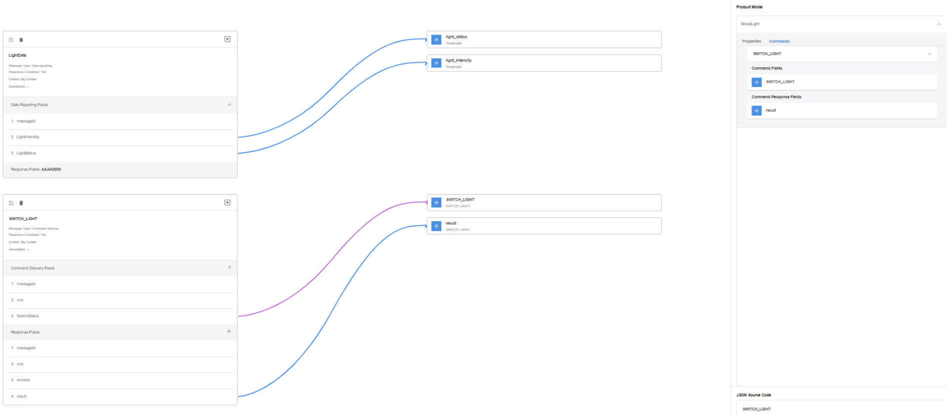
Data Type (Big Endian): int8u

Offset: 4-5

* Length: 1

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.



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

----End

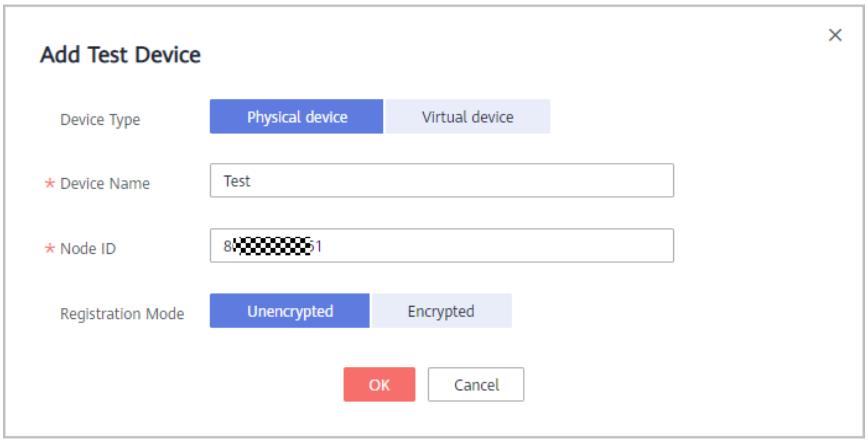
Debugging the Codec Online Using a Physical Device

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.
- **Registration Mode**: Select **Unencrypted**. Note: If DTLS is used for device access, select **Encrypted** and keep the secret properly.



Add Test Device

Device Type [✓ Physical device] Virtual device

* Device Name: Test

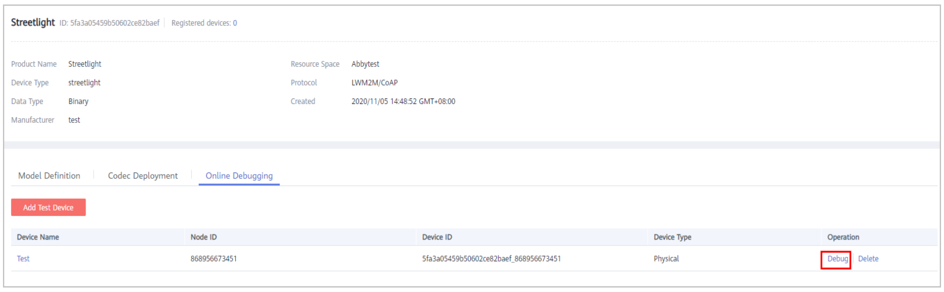
* Node ID: 8[redacted]1

Registration Mode [✓ Unencrypted] Encrypted

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.



Streetlight ID: 5fa3a05459b50602ce82baef | Registered devices: 0

Product Name	Streetlight	Resource Space	Abbytest
Device Type	streetlight	Protocol	LWM2M/CoAP
Data Type	Binary	Created	2020/11/05 14:48:52 GMT+08:00
Manufacturer	test		

Device Name	Node ID	Device ID	Device Type	Operation
Test	868956673451	5fa3a05459b50602ce82baef_868956673451	Physical	Debug Delete

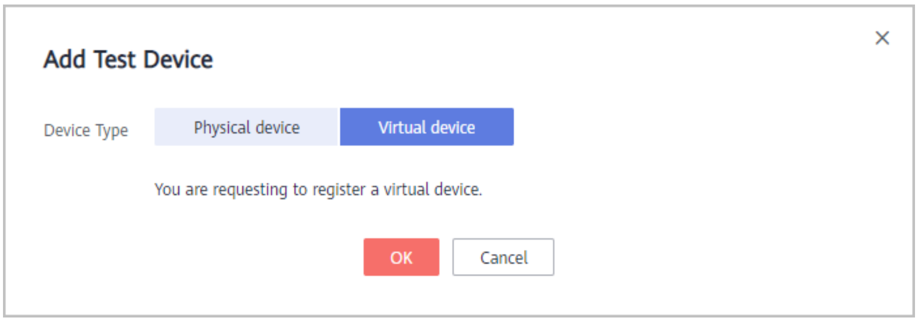
Step 4 Simulate a scenario where a control command is remotely delivered. Specifically, under **Application Simulator**, select **StreetLight** for **Service**, **SWITCH_LIGHT** for **Command**, **ON** for **SWITCH_LIGHT**, and click **Send**. The street light is turned on.

----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 **Simulator**. Only one virtual device can be created for each product.

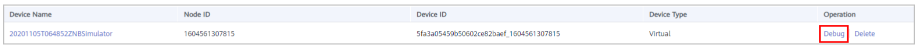


Add Test Device

Device Type: Physical device | Virtual device

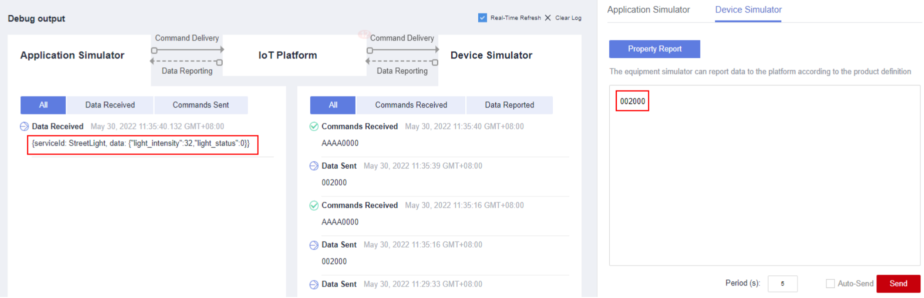
You are requesting to register a virtual device.

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



Device Name	Node ID	Device ID	Device Type	Operation
20201105T064852ZNBSimulator	1604561307815	5fa3a05459b50602ce82baef_1604561307815	Virtual	Debug Delete

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 messageId, 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** should be displayed.



```
Data Received May 30, 2022 11:35:40.132 GMT+08:00
{serviceId: StreetLight, data: {"light_intensity":32,"light_status":0}}
```

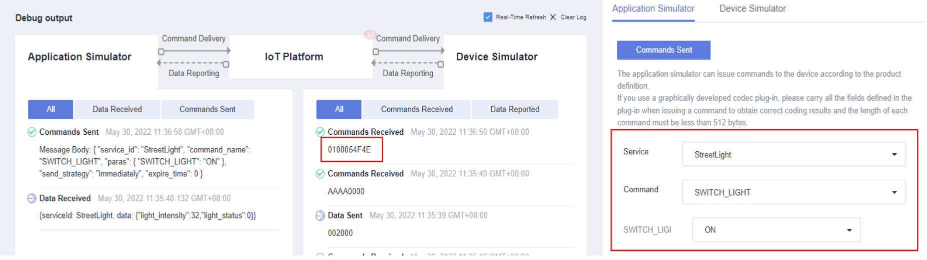
```
Commands Received May 30, 2022 11:35:40 GMT+08:00
AAAA0000
Data Sent May 30, 2022 11:35:39 GMT+08:00
002000
Commands Received May 30, 2022 11:35:16 GMT+08:00
AAAA0000
Data Sent May 30, 2022 11:35:16 GMT+08:00
002000
Data Sent May 30, 2022 11:29:33 GMT+08:00
```

The equipment simulator can report data to the platform according to the product definition

002000

Period (s): 5

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** should be displayed.



```
Commands Sent May 30, 2022 11:36:50 GMT+08:00
Message Body: { "service_id": "StreetLight", "command_name":
"SWITCH_LIGHT", "paras": { "SWITCH_LIGHT": "ON" },
"send_strategy": "immediately", "expire_time": 0 }
Data Received May 30, 2022 11:35:40.132 GMT+08:00
{serviceId: StreetLight, data: {"light_intensity":32,"light_status":0}}
```

```
Commands Received May 30, 2022 11:36:50 GMT+08:00
0100054F4E
Commands Received May 30, 2022 11:35:40 GMT+08:00
AAAA0000
Data Sent May 30, 2022 11:35:39 GMT+08:00
002000
```

The application simulator can issue commands to the device according to the product definition.
If you use a graphically developed codec plug-in, please carry all the fields defined in the plug-in when issuing a command to obtain correct coding results and the length of each command must be less than 512 bytes.

Service: StreetLight

Command: SWITCH_LIGHT

SWITCH_LIGI: ON

----End

Debugging Using an Offline Simulator

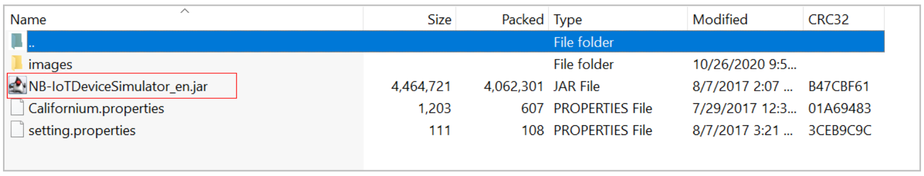
The NB-IoT device simulator is used to simulate the access of NB-IoT devices (devices using LwM2M over CoAP) to the platform for data reporting and command delivery.

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.

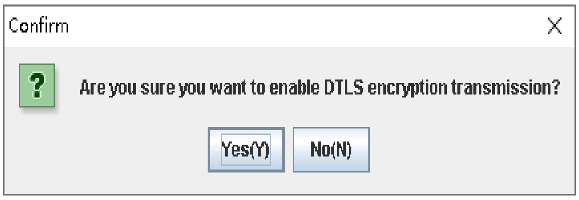


Name	Size	Packed	Type	Modified	CRC32
..			File folder		
images			File folder	10/26/2020 9:5...	
NB-IoTDeviceSimulator_en.jar	4,464,721	4,062,301	JAR File	8/7/2017 2:07 ...	B47CBF61
Californium.properties	1,203	607	PROPERTIES File	7/29/2017 12:3...	01A69483
setting.properties	111	108	PROPERTIES File	8/7/2017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



Confirm

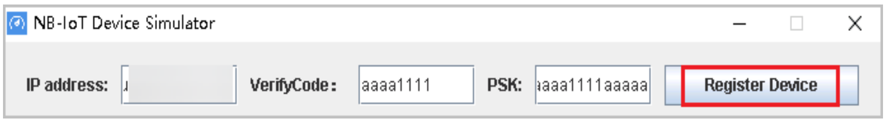
Are you sure you want to enable DTLS encryption transmission?

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**.

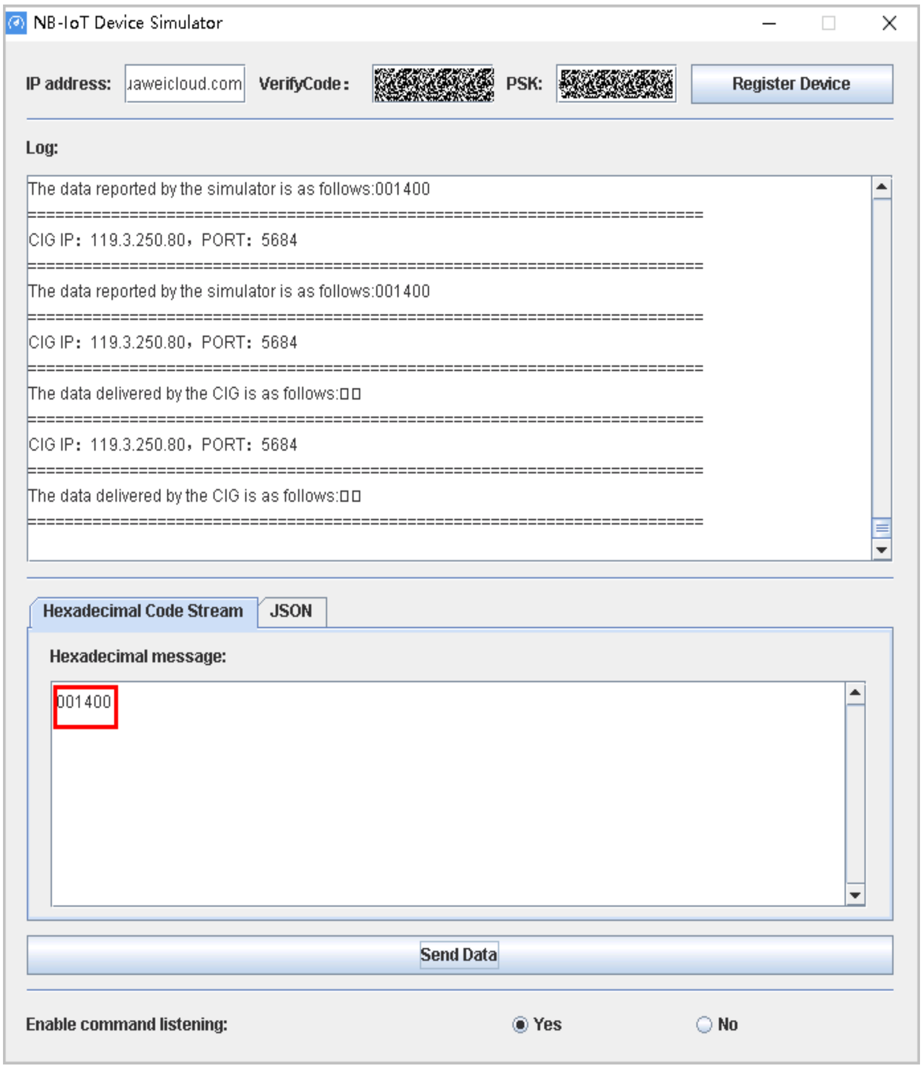


NB-IoT Device Simulator

IP address: [redacted] VerifyCode: aaaa1111 PSK: aaaa1111aaaaa

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 messageId, the second byte 14 indicates the light intensity, and the third byte 00 indicates the switch status.)



IP address: ıaweicloud.com VerifyCode: [redacted] PSK: [redacted]

Log:

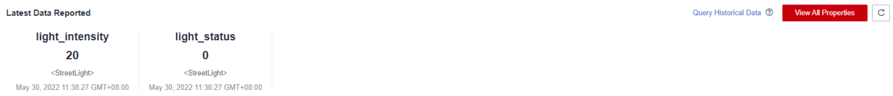
```
The data reported by the simulator is as follows:001400
==========================================================================
CIG IP: 119.3.250.80, PORT: 5684
==========================================================================
The data reported by the simulator is as follows:001400
==========================================================================
CIG IP: 119.3.250.80, PORT: 5684
==========================================================================
The data delivered by the CIG is as follows:□□
==========================================================================
CIG IP: 119.3.250.80, PORT: 5684
==========================================================================
The data delivered by the CIG is as follows:□□
==========================================================================
```

Hexadecimal message:

```
001400
```

Enable command listening: ● Yes ○ No

The data is reported successfully. You can return to the management 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**.



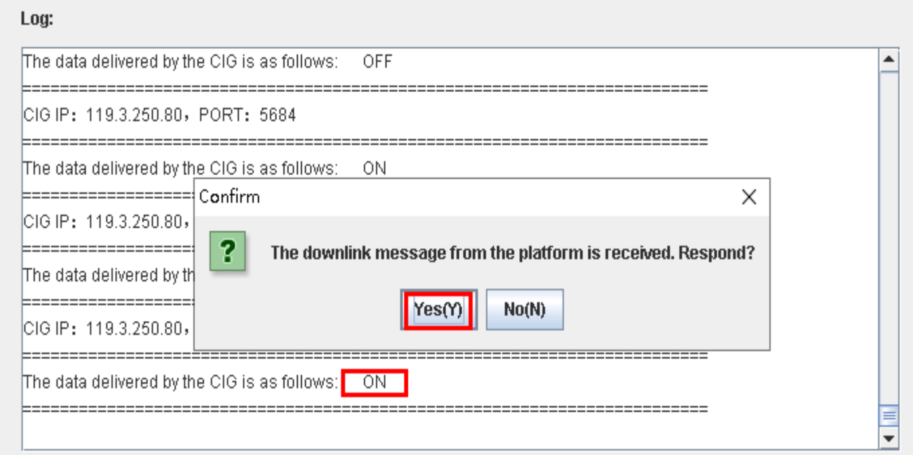
Latest Data Reported

light_intensity	light_status
20	0
<StreetLight>	<StreetLight>
May 30, 2022 11:38:27 GMT+08:00	May 30, 2022 11:38:27 GMT+08:00

Step 6 Simulate the remote command delivery scenario. On the management 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.



Log:

```
The data delivered by the CIG is as follows:    OFF
========================================================================
CIG IP: 119.3.250.80, PORT: 5684
========================================================================
The data delivered by the CIG is as follows:    ON
=================
CIG IP: 119.3.250.80,
=================
The data delivered by th
=================
CIG IP: 119.3.250.80,
========================================================================
The data delivered by the CIG is as follows:    ON
========================================================================
```

Confirm

The downlink message from the platform is received. Respond?

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

2.9 Upgrading the MQTT Device Firmware in OTA Mode

Scenarios

Message Queuing Telemetry Transport (MQTT) is a publish/subscribe messaging protocol that transports messages between clients and a server. It is suitable for remote sensors and control devices (such as smart street lamps) that have limited computing capabilities and work in low-bandwidth, unreliable networks through persistent connections. Firmware upgrade is a basic function that network devices must support. Online upgrades are important especially when the firmware needs to be upgraded due to security vulnerabilities, software bugs, function optimization, and device performance improvement. This topic 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 2-45 Upgrade process

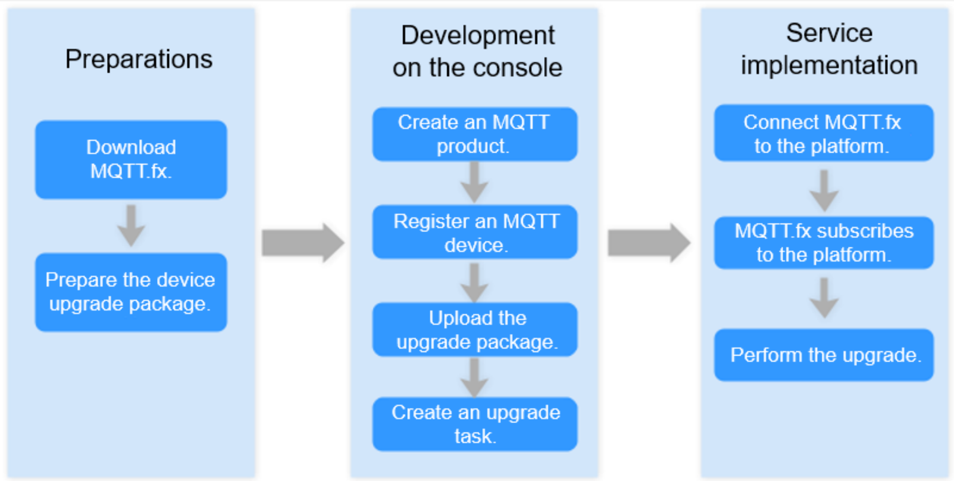
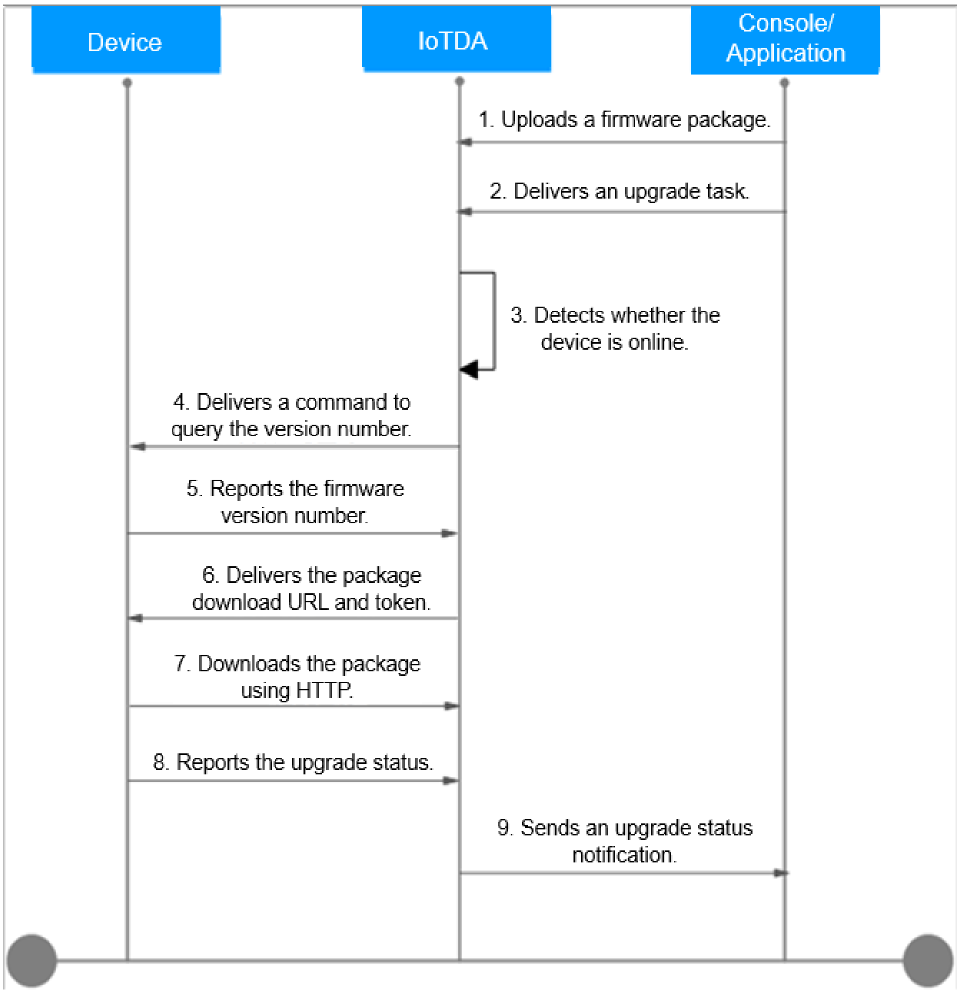

Preparations
Download MQTT.fx.
Prepare the device upgrade package.
Development on the console
Create an MQTT product.
Register an MQTT device.
Upload the upgrade package.
Create an upgrade task.
Service implementation
Connect MQTT.fx to the platform.
MQTT.fx subscribes to the platform.
Perform the upgrade.

Figure 2-46 MQTT device upgrade process


Device
IoTDA
Console/ Application
1. Uploads a firmware package.
2. Delivers an upgrade task.
3. Detects whether the device is online.
4. Delivers a command to query the version number.
5. Reports the firmware version number.
6. Delivers the package download URL and token.
7. Downloads the package using HTTP.
8. Reports the upgrade status.
9. Sends an upgrade status notification.

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

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.

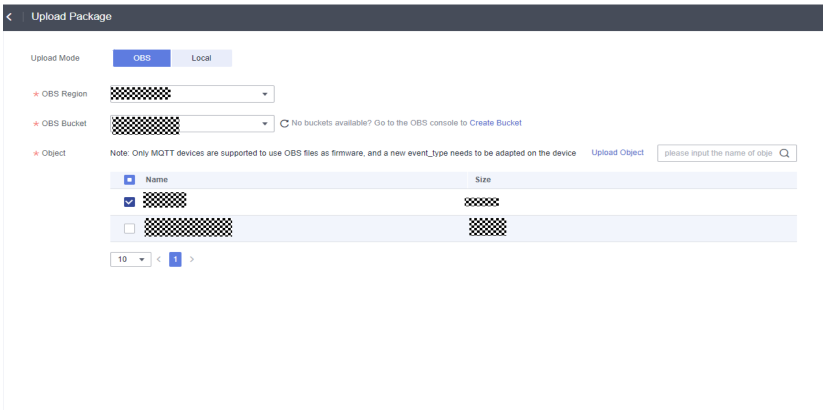
Uploading an Upgrade Package

Step 1 Visit the **IoTDA** product page and click **Access Console**.

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 page displayed, you can upload a firmware package from OBS or your local PC.



Upload Package

Upload Mode: OBS | Local

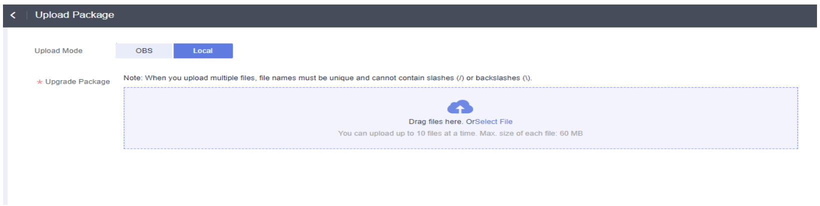
* OBS Region: [redacted]

* OBS Bucket: [redacted] No buckets available? Go to the OBS console to Create Bucket

* Object: Note: Only MQTT devices are supported to use OBS files as firmware, and a new event_type needs to be adapted on the device Upload Object please input the name of obje

■	Name	Size
☑	[redacted]	[redacted]
□	[redacted]	[redacted]

10 ‹ 1 ›

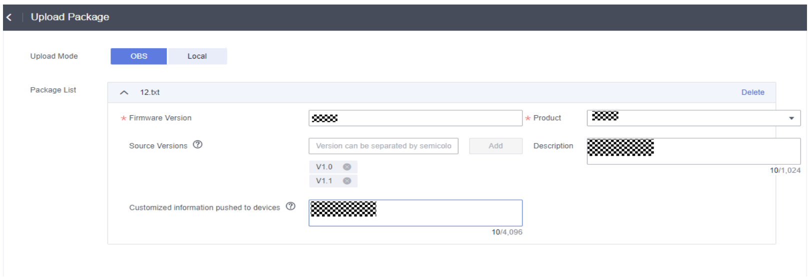


Note: When you upload multiple files, file names must be unique and cannot contain slashes (/) or backslashes (\).

Drag files here. OrSelect File

You can upload up to 10 files at a time. Max. size of each file: 60 MB

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



12.txt

* Firmware Version: [redacted]

* Product: [redacted]

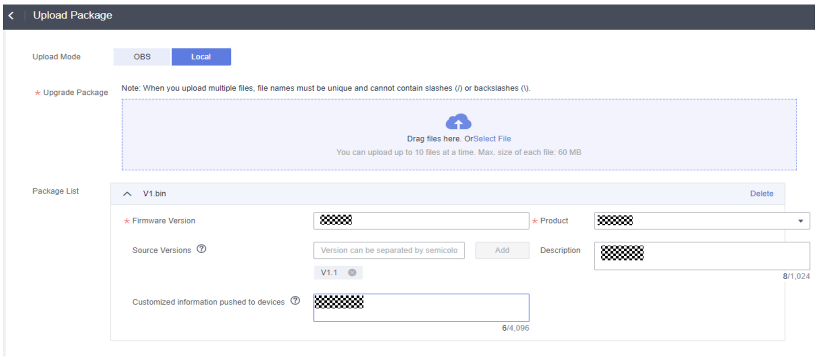
Source Versions: Version can be separated by semicolo

V1.0

V1.1

Description: [redacted] 10/1,024

Customized information pushed to devices: [redacted] 10/4,096



Note: When you upload multiple files, file names must be unique and cannot contain slashes (/) or backslashes (\).

Drag files here. OrSelect File

You can upload up to 10 files at a time. Max. size of each file: 60 MB

V1.bin

* Firmware Version: [redacted]

* Product: [redacted]

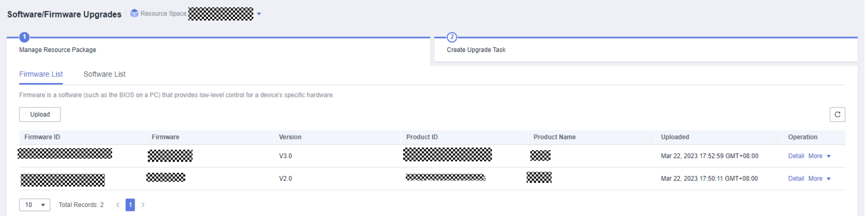
Source Versions: Version can be separated by semicolo

V1.1

Description: [redacted] 8/1,024

Customized information pushed to devices: [redacted] 6/4,096

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



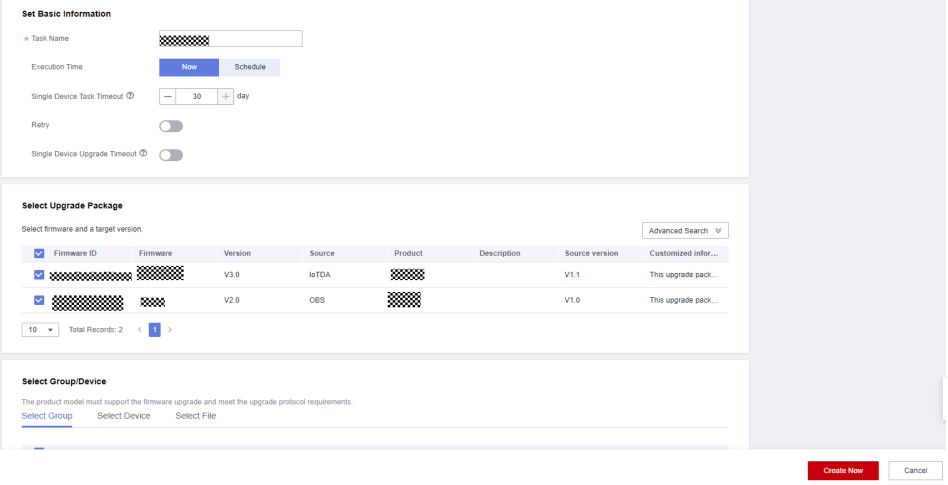
Resource Space [redacted]

Firmware is a software (such as the BIOS on a PC) that provides low-level control for a device's specific hardware.

Firmware ID	Firmware	Version	Product ID	Product Name	Uploaded	Operation
[redacted]	[redacted]	V3.0	[redacted]	[redacted]	Mar 22, 2023 17:52:59 GMT+08:00	Detail More
[redacted]	[redacted]	V2.0	[redacted]	[redacted]	Mar 22, 2023 17:50:11 GMT+08:00	Detail More

10 Total Records: 2 < 1 >

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



Set Basic Information

* Task Name: [redacted]

Execution Time: Now | Schedule

Single Device Task Timeout: 30 day

Retry

Single Device Upgrade Timeout

Select Upgrade Package

Select firmware and a target version.

☑	Firmware ID	Firmware	Version	Source	Product	Description	Source version	Customized infor...
☑	[redacted]	[redacted]	V3.0	IoTDA	[redacted]		V1.1	This upgrade pack...
☑	[redacted]	[redacted]	V2.0	OBS	[redacted]		V1.0	This upgrade pack...

10 Total Records: 2 < 1 >

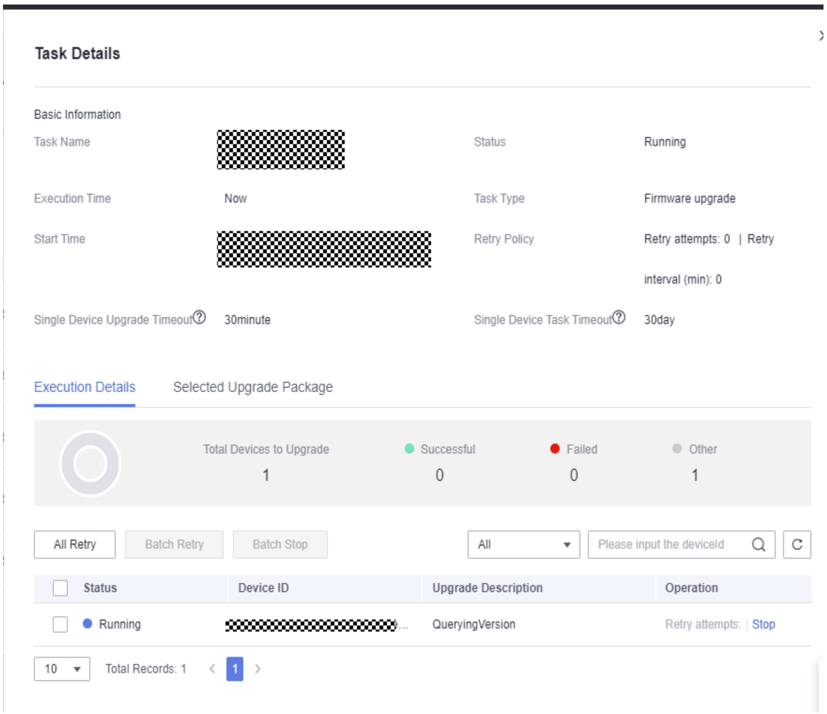
Select Group/Device

The product model must support the firmware upgrade and meet the upgrade protocol requirements.

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 **Detail** 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.



Task Details

Basic Information

Task Name	[redacted]	Status	Running
Execution Time	Now	Task Type	Firmware upgrade
Start Time	[redacted]	Retry Policy	Retry attempts: 0 | Retry interval (min): 0
Single Device Upgrade Timeout	30minute	Single Device Task Timeout	30day

Execution Details | Selected Upgrade Package

Total Devices to Upgrade	Successful	Failed	Other
1	0	0	1

All Retry | Batch Retry | Batch Stop | All | Please input the deviceId

□	Status	Device ID	Upgrade Description	Operation
□	● Running	[redacted]	QueryingVersion	Retry attempts: | Stop

10 Total Records: 1 < 1 >

----End

Service Implementation

Step 1 Use MQTT.fx to simulate device access to the platform.

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.

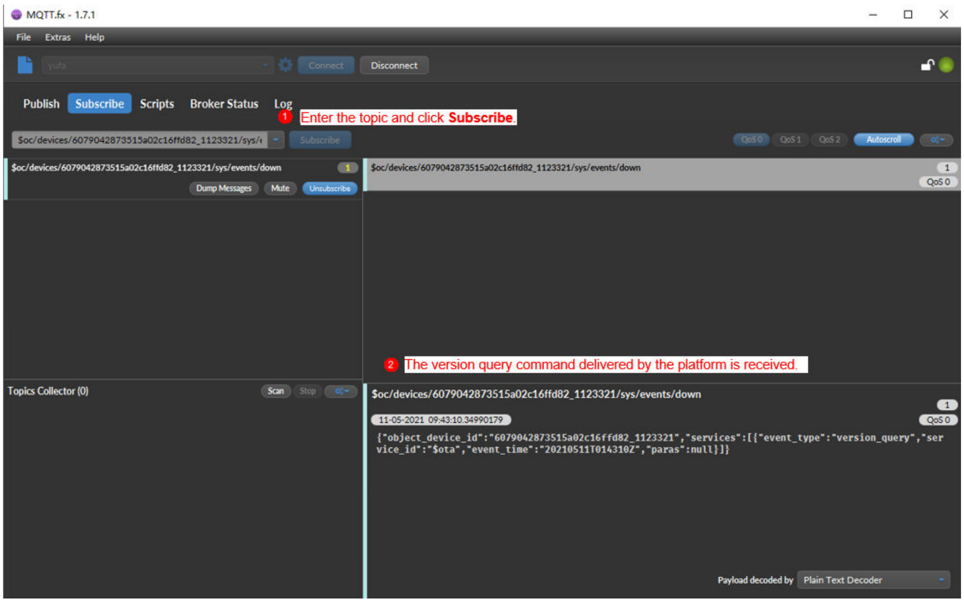
Topic

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

Parameters

For details, see **Platform Delivering an Event to Obtain Version Information**.

Figure 2-47 Subscribe/Push



1 Enter the topic and click **Subscribe**.

$oc/devices/6079042873515a02c16ffd82_1123321/sys/events/down

2 The version query command delivered by the platform is received.

$oc/devices/6079042873515a02c16ffd82_1123321/sys/events/down

11-05-2021 09:43:10.34990179

```
{"object_device_id":"6079042873515a02c16ffd82_1123321","services":[{"event_type":"version_query","ser
vice_id":"$ota","event_time":"20210511T014310Z","paras":null}]}
```

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

Topic

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

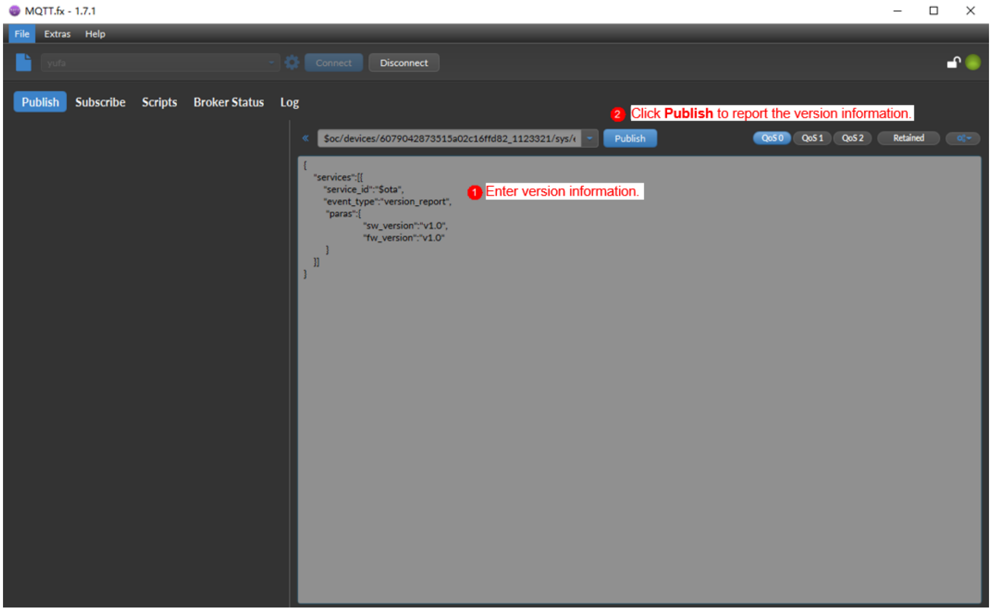
Parameters

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 2-48 Reporting version numbers



$oc/devices/6079042873515a02c16ffd82_1123321/sys/

```
{
  "services":[{
    "service_id":"$ota",
    "event_type":"version_report",
     "paras":{
            "sw_version":"v1.0",
            "fw_version":"v1.0"
     }
  }]
}
```

1 Enter version information.

2 Click **Publish** to report the version information.

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

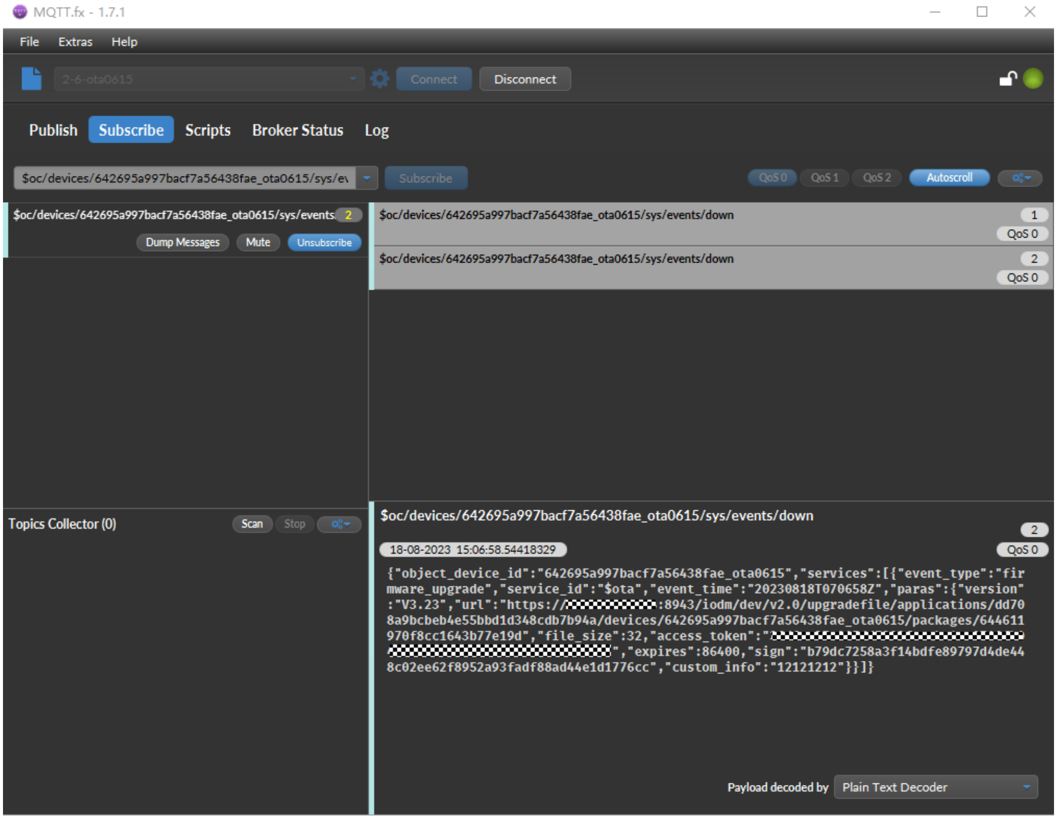
Topic

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

Parameters

For details, see **Platform Delivering an Upgrade Event**.

Figure 2-49 Obtaining the upgrade notification



$oc/devices/642695a997bacf7a56438fae_ota0615/sys/ev

$oc/devices/642695a997bacf7a56438fae_ota0615/sys/events 2

$oc/devices/642695a997bacf7a56438fae_ota0615/sys/events/down 1 QoS 0

$oc/devices/642695a997bacf7a56438fae_ota0615/sys/events/down 2 QoS 0

$oc/devices/642695a997bacf7a56438fae_ota0615/sys/events/down 2

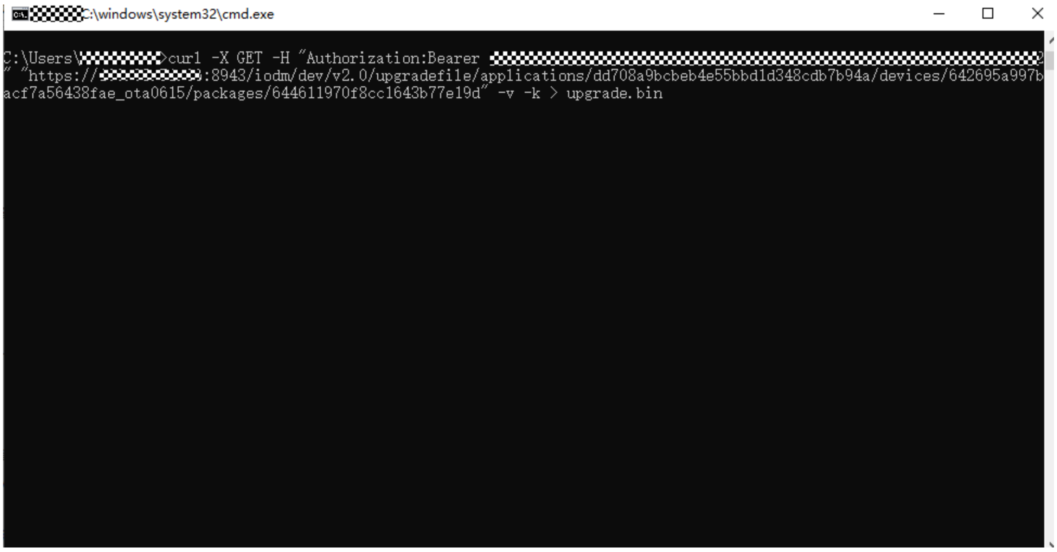
18-08-2023 15:06:58.54418329 QoS 0

```
{"object_device_id":"642695a997bacf7a56438fae_ota0615","services":[{"event_type":"fir
mware_upgrade","service_id":"$ota","event_time":"20230818T070658Z","paras":{"version"
:"V3.23","url":"https://[redacted]:8943/iodm/dev/v2.0/upgradefile/applications/dd70
8a9bcbeb4e55bbd1d348cdb7b94a/devices/642695a997bacf7a56438fae_ota0615/packages/644611
970f8cc1643b77e19d","file_size":32,"access_token":"[redacted]","expires":86400,"sign":"b79dc7258a3f14bdfe89797d4de44
8c02ee62f8952a93fadf88ad44e1d1776cc","custom_info":"12121212"}}]}
```

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 2-50 Downloading the upgrade package



```
C:\Users\[redacted]>curl -X GET -H "Authorization:Bearer [redacted]
" "https://[redacted]:8943/iodm/dev/v2.0/upgradefile/applications/dd708a9bcbeb4e55bbd1d348cdb7b94a/devices/642695a997b
acf7a56438fae_ota0615/packages/644611970f8cc1643b77e19d" -v -k > upgrade.bin
```

Example

```
curl -X GET -H "Authorization:Bearer ****************************" "https://100.93.28.202:8943/iodm/dev/v2.0/
upgradefile/applications/ddecccc223574aee9466fe8f7e16a205/devices/
6079042873515a02c16ffd82_123456789/packages/4f201f38c281ca5d40794a3f" -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: $oc/devices/{device_id}/sys/events/up

Parameters

For details, see **Device Reporting the Upgrade Status**.

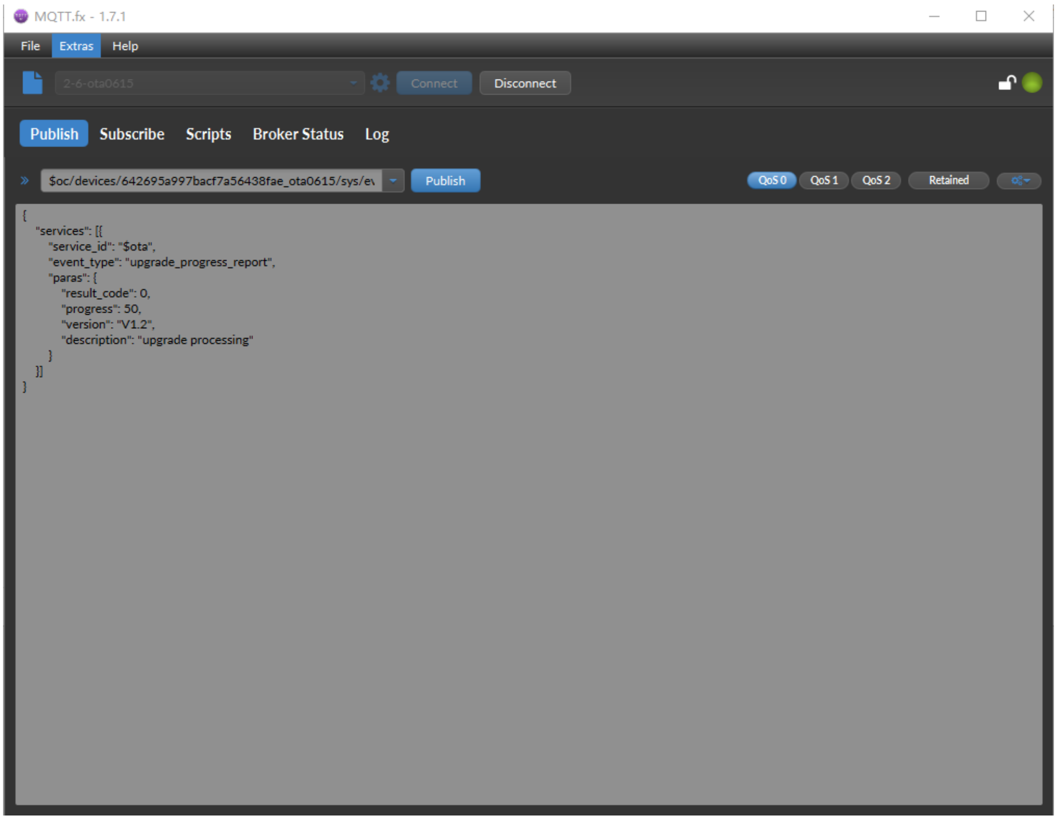
Example

```
Topic: $oc/devices/{device_id}/sys/events/up
Data format:
 {     "object_device_id": "{object_device_id}",
    "services": [{
        "service_id": "$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.

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

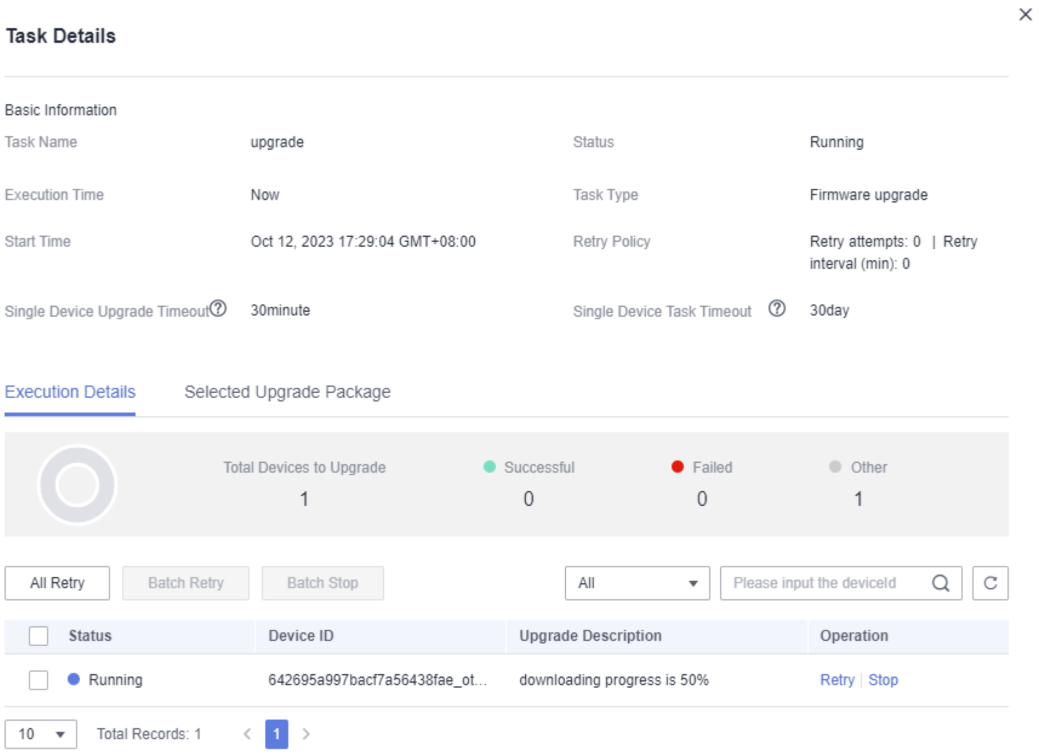
Figure 2-51 Reporting the upgrade progress (50%)



$oc/devices/642695a997bacf7a56438fae_ota0615/sys/ev

```
{
   "services": [{
      "service_id": "$ota",
      "event_type": "upgrade_progress_report",
      "paras": {
         "result_code": 0,
         "progress": 50,
         "version": "V1.2",
         "description": "upgrade processing"
      }
   }]
}
```

Figure 2-52 Querying the upgrade progress



Basic Information

Task Name	upgrade	Status	Running
Execution Time	Now	Task Type	Firmware upgrade
Start Time	Oct 12, 2023 17:29:04 GMT+08:00	Retry Policy	Retry attempts: 0 | Retry interval (min): 0
Single Device Upgrade Timeout	30minute	Single Device Task Timeout	30day

Total Devices to Upgrade	● Successful	● Failed	● Other
1	0	0	1

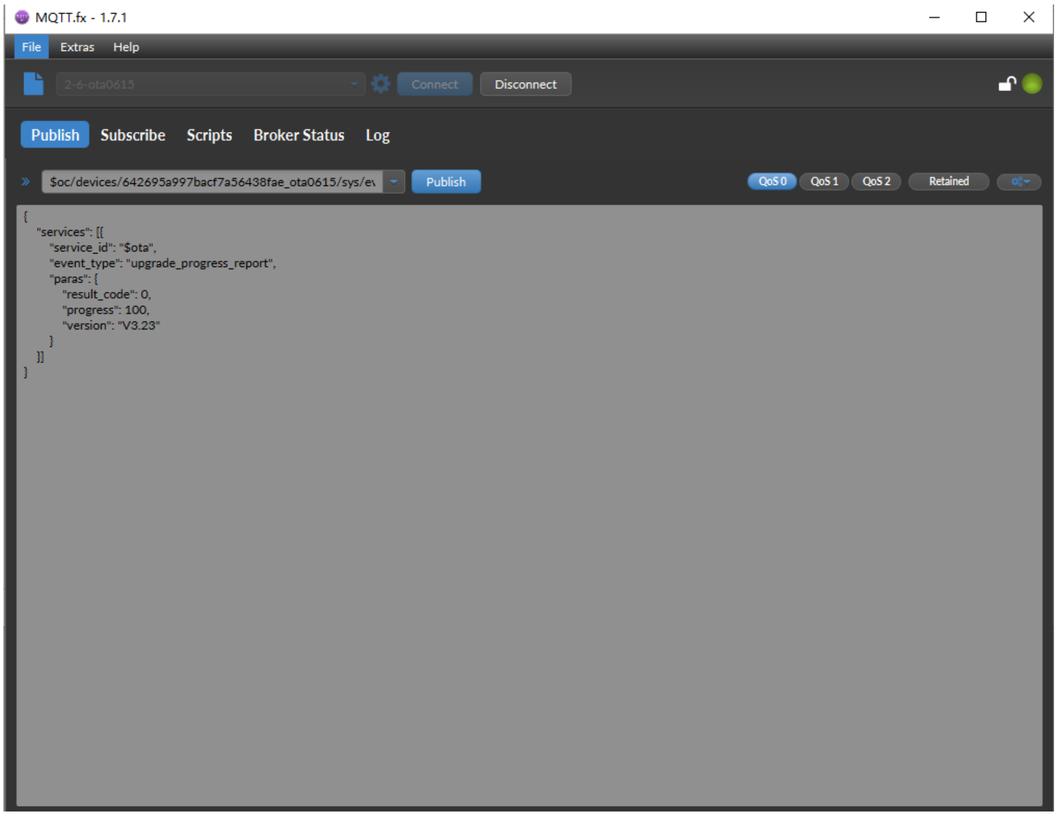
□	Status	Device ID	Upgrade Description	Operation
□	● Running	642695a997bacf7a56438fae_ot...	downloading progress is 50%	Retry | Stop

10 Total Records: 1 ‹ 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.

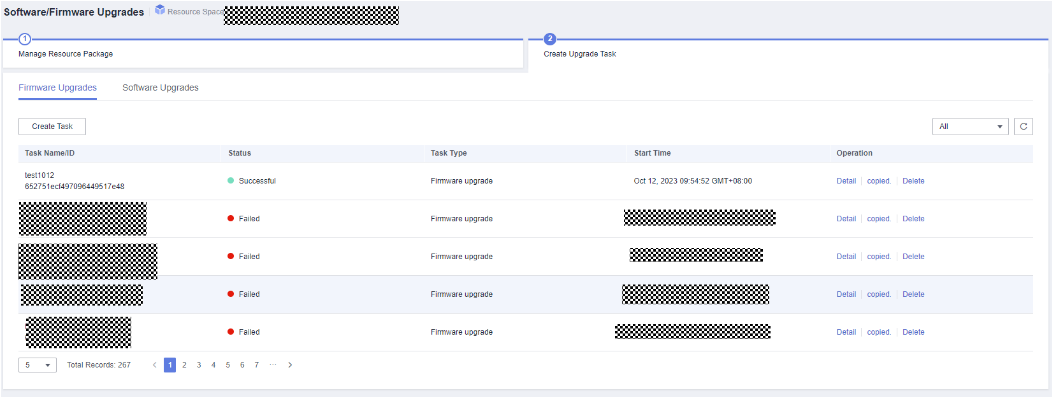
Figure 2-53 Reporting the upgrade progress (100%)



$oc/devices/642695a997bacf7a56438fae_ota0615/sys/ev

```
{
  "services": [{
    "service_id": "$ota",
    "event_type": "upgrade_progress_report",
    "paras": {
      "result_code": 0,
      "progress": 100,
      "version": "V3.23"
    }
  }]
}
```

Figure 2-54 Querying the upgrade status



[redacted]

Task Name/ID	Status	Task Type	Start Time	Operation
test1012 652751ecf497096449517e48	● Successful	Firmware upgrade	Oct 12, 2023 09:54:52 GMT+08:00	Detail | copied. | Delete
[redacted]	● Failed	Firmware upgrade	[redacted]	Detail | copied. | Delete
[redacted]	● Failed	Firmware upgrade	[redacted]	Detail | copied. | Delete
[redacted]	● Failed	Firmware upgrade	[redacted]	Detail | copied. | Delete
[redacted]	● Failed	Firmware upgrade	[redacted]	Detail | copied. | Delete

----End

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

This topic 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

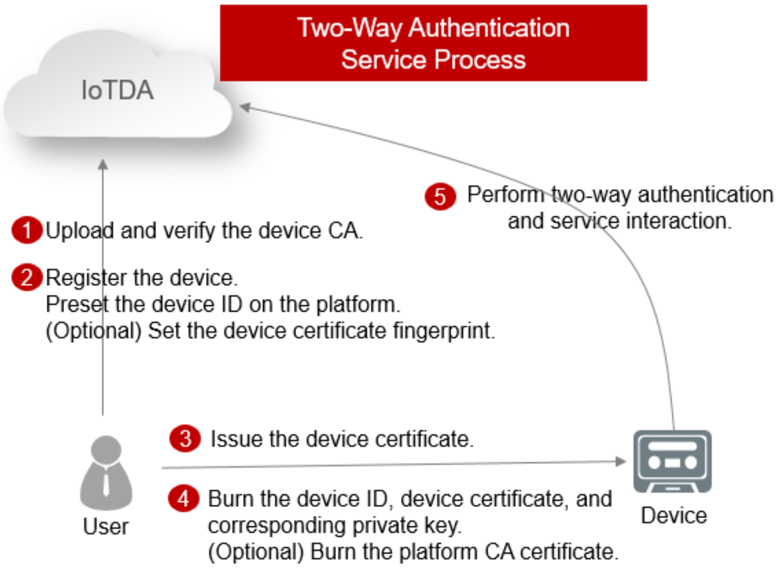
- You have registered a Huawei Cloud account. If you have not registered, click **here** to complete the registration.
- You have subscribed to the IoTDA service. If you have not subscribed to the service, go to the **IoTDA** service page, and click **Access Console** to subscribe to the service.

Constraints

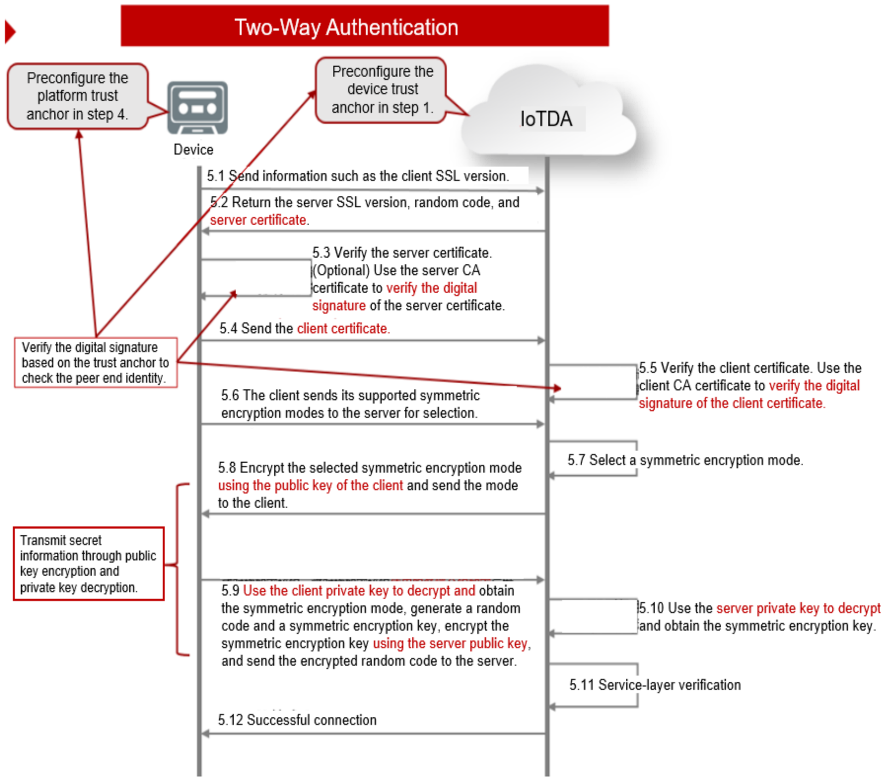
- Only MQTT devices can use X.509 certificates for identity authentication.
- You can upload a maximum of 100 device CA certificates.

X.509-based Authentication

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


Two-Way Authentication Service Process
IoTDA
1 Upload and verify the device CA.
2 Register the device.
Preset the device ID on the platform.
(Optional) Set the device certificate fingerprint.
5 Perform two-way authentication and service interaction.
3 Issue the device certificate.
4 Burn the device ID, device certificate, and corresponding private key.
(Optional) Burn the platform CA certificate.
User
Device

- Two-way certificate authentication:


Two-Way Authentication
Preconfigure the platform trust anchor in step 4.
Device
Preconfigure the device trust anchor in step 1.
IoTDA
5.1 Send information such as the client SSL version.
5.2 Return the server SSL version, random code, and server certificate.
5.3 Verify the server certificate. (Optional) Use the server CA certificate to verify the digital signature of the server certificate.
5.4 Send the client certificate.
Verify the digital signature based on the trust anchor to check the peer end identity.
5.5 Verify the client certificate. Use the client CA certificate to verify the digital signature of the client certificate.
5.6 The client sends its supported symmetric encryption modes to the server for selection.
5.7 Select a symmetric encryption mode.
5.8 Encrypt the selected symmetric encryption mode using the public key of the client and send the mode to the client.
Transmit secret information through public key encryption and private key decryption.
5.9 Use the client private key to decrypt and obtain the symmetric encryption mode, generate a random code and a symmetric encryption key, encrypt the symmetric encryption key using the server public key, and send the encrypted random code to the server.
5.10 Use the server private key to decrypt and obtain the symmetric encryption key.
5.11 Service-layer verification
5.12 Successful connection

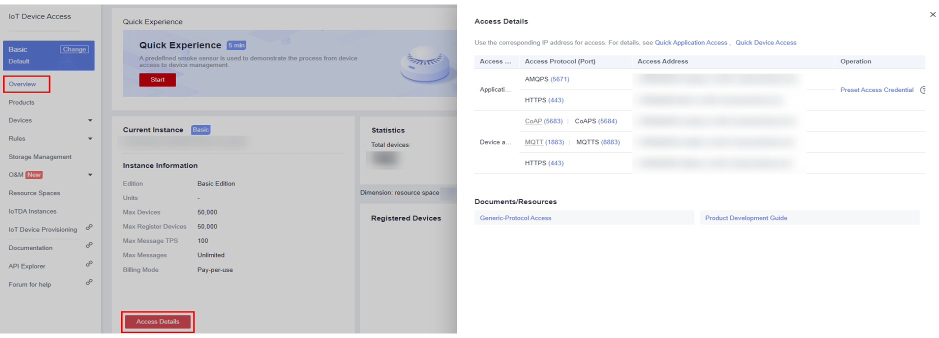
Obtaining Device Access Information

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 view the device access information and record domain names and ports.

Figure 2-55 Platform access addresses



Instance Information

Edition	Basic Edition
Units	-
Max Devices	50,000
Max Register Devices	50,000
Max Message TPS	100
Max Messages	Unlimited
Billing Mode	Pay-per-use

Access Details

Use the corresponding IP address for access. For details, see Quick Application Access , Quick Device Access

Access ...	Access Protocol (Port)	Access Address	Operation
Applicati...	AMQPS (5671)	[redacted]	Preset Access Credential
	HTTPS (443)	[redacted]	
Device a...	CoAP (5683) | CoAPS (5684)	[redacted]	
	MQTT (1883) | MQTTS (8883)	[redacted]	
	HTTPS (443)	[redacted]	

Documents/Resources

Generic-Protocol Access Product Development Guide

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

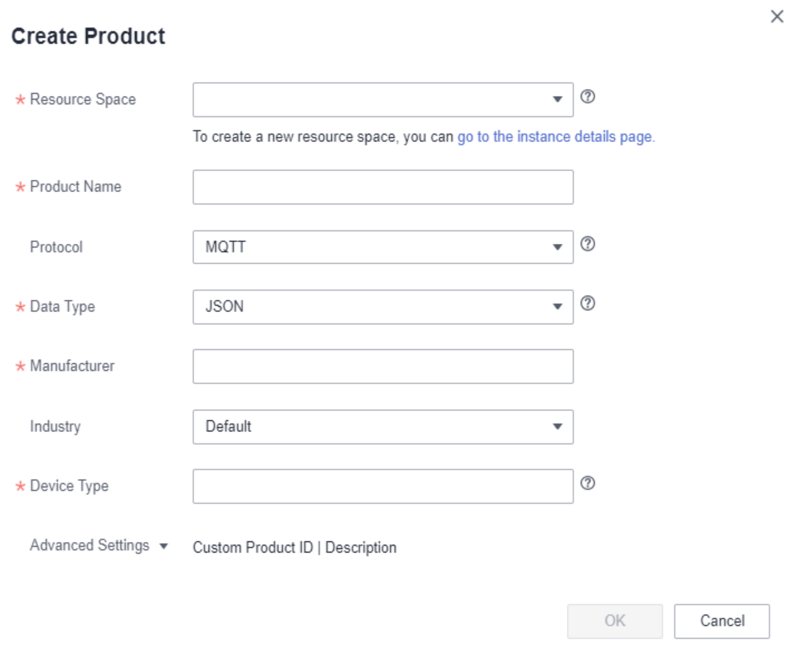
Creating a Product

Step 1 Log in to the **console**, choose **Products** in the navigation pane, and click **Create Product** in the upper right corner.

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. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create it first.
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacture r	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Industry	Set this parameter based on the site requirements. If the product model preset on the platform is used, set this parameter based on the industry to which the product model belongs. If not, select **Default**.
Device Type	Set this parameter based on the site requirements. If the product model preset on the platform is used, the device type is automatically matched.
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.
Description	Provide a description for the product. Set this parameter based on the site requirements.

Figure 2-56 Creating a product



Create Product

* Resource Space

To create a new resource space, you can go to the instance details page.

* Product Name

Protocol: MQTT

* Data Type: JSON

* Manufacturer

Industry: Default

* Device Type

Advanced Settings: Custom Product ID | Description

----End

Developing a Product Model

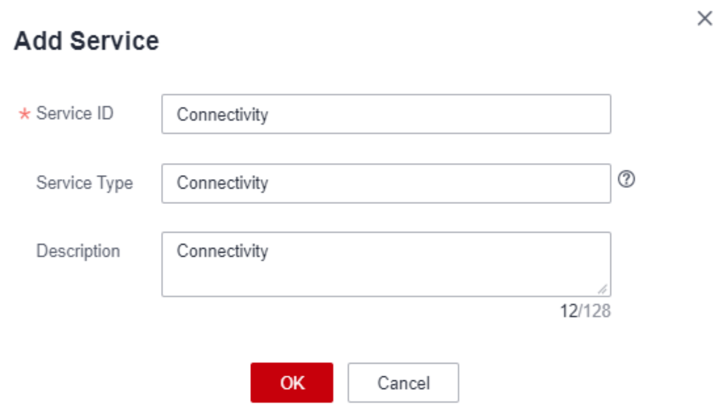
Step 1 Click the created product. The product details page is displayed.

Step 2 On the **Model Definition** tab page of the product details page, click **Custom 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 2-57 Adding a service



Add Service

* Service ID: Connectivity

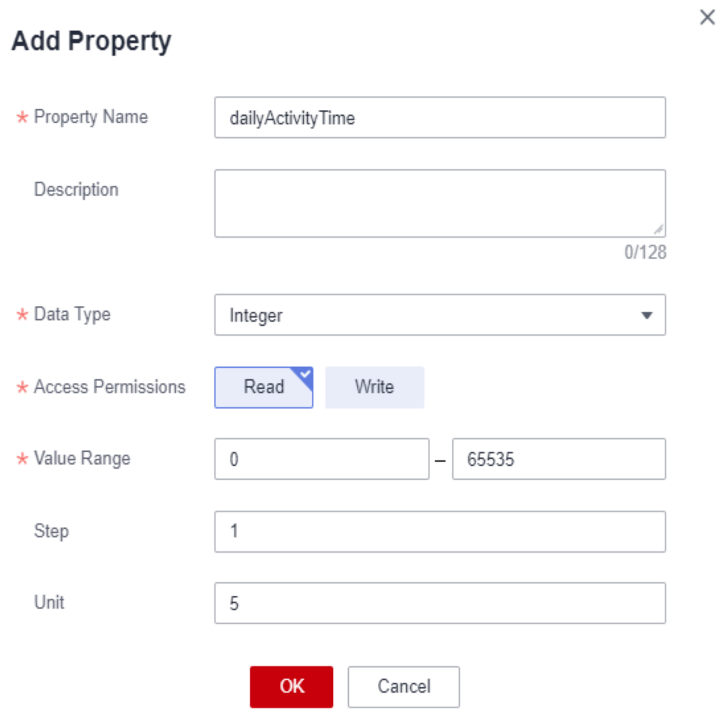
Service Type: Connectivity

Description: Connectivity

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 0–65535.
   - **Step**: Enter **1**.
   - **Unit**: Enter **s**.

Figure 2-58 Adding a property



Add Property

* Property Name: dailyActivityTime

Description:

0/128

* Data Type: Integer

* Access Permissions: Read (selected) Write

* Value Range: 0 – 65535

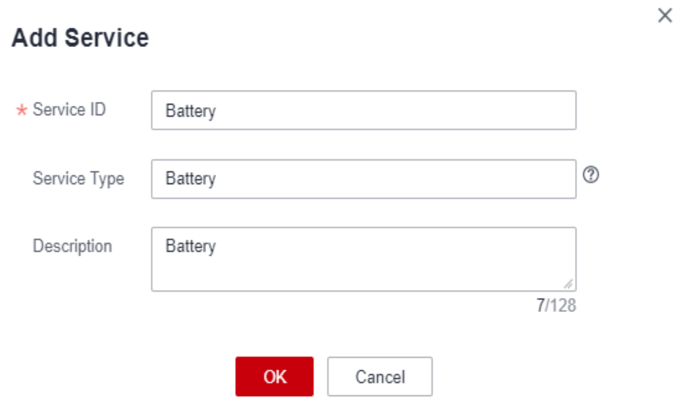
Step: 1

Unit: 5

Step 4 Add the **Battery** service.

1. On the **Model Definition** 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 2-59 Adding a service



Add Service

* Service ID: Battery

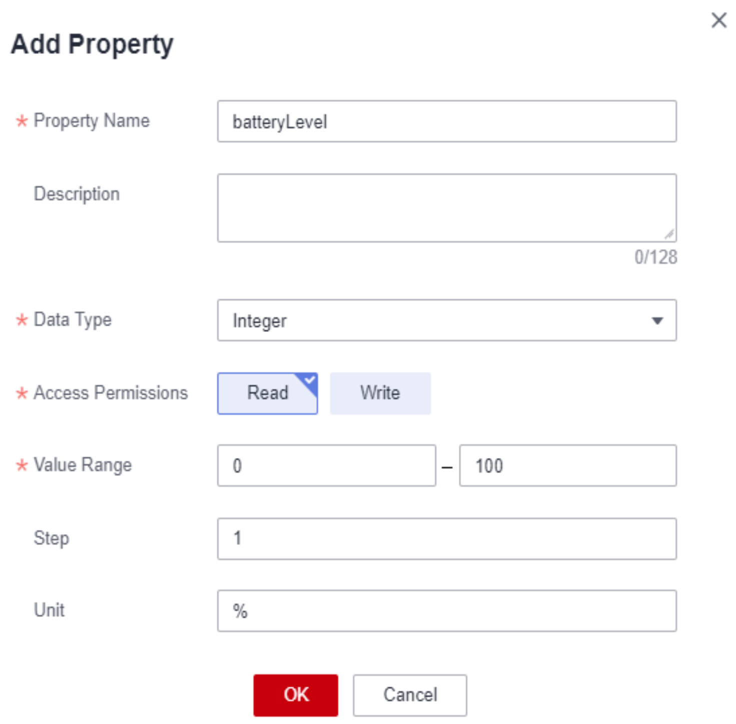
Service Type: Battery

Description: Battery

7/128

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 2-60 Adding a property



Add Property

* Property Name: batteryLevel

Description: 0/128

* Data Type: Integer

* Access Permissions: Read, Write

* Value Range: 0 – 100

Step: 1

Unit: %

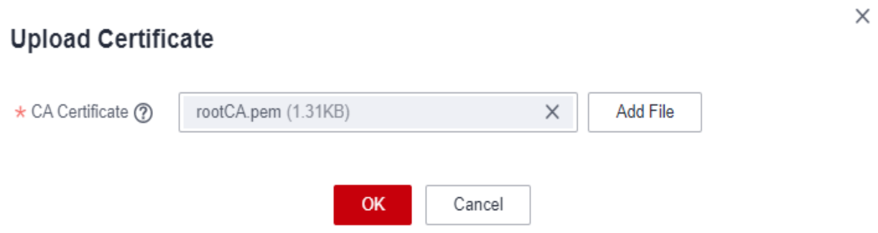
----End

Uploading a Device CA certificate

Step 1 In the left navigation pane, choose **Devices** > **Device CA Certificates**, and click **Upload Certificate** in the upper right corner.

Step 2 In the displayed dialog box, click **Select File** to add a file, and then click **OK**.

Figure 2-61 Uploading a certificate



Upload Certificate

* CA Certificate: rootCA.pem (1.31KB)

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.

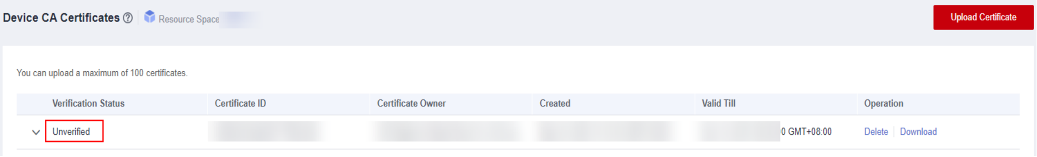
1. Download and install **OpenSSL**.
2. Open the CLI as user **admin**.
3. Run **cd c:\openssl\bin** (replace **c:\openssl\bin** with the actual OpenSSL installation directory) to access the OpenSSL view.
4. Generate a public/private key pair.
   ```
   openssl genrsa -out rootCA.key 2048
   ```
5. Use 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 2-62 Device CA certificate



Device CA Certificates ⓘ Resource Space [redacted]

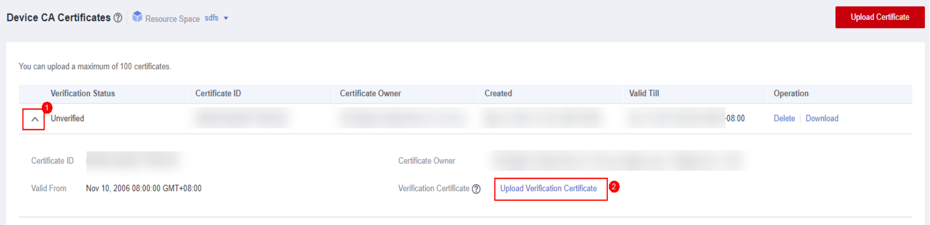
You can upload a maximum of 100 certificates.

Verification Status	Certificate ID	Certificate Owner	Created	Valid Till	Operation
Unverified	[redacted]	[redacted]	[redacted]	[redacted]0 GMT+08:00	Delete | Download

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 2-63 Uploading a verification certificate



Device CA Certificates ⓘ Resource Space sdfs ▾

You can upload a maximum of 100 certificates.

Verification Status	Certificate ID	Certificate Owner	Created	Valid Till	Operation
Unverified	[redacted]	[redacted]	[redacted]	[redacted]-08:00	Delete | Download

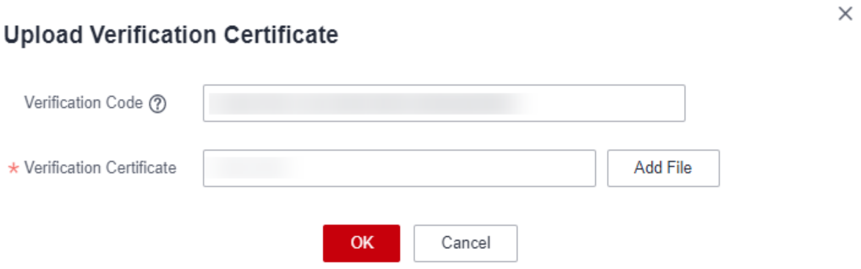
Certificate ID [redacted]

Certificate Owner [redacted]

Valid From Nov 10, 2006 08:00:00 GMT+08:00

Verification Certificate ⓘ Upload Verification Certificate

Figure 2-64 Obtaining a verification code



Upload Verification Certificate

Verification Code ⓘ [redacted]

* Verification Certificate [redacted]

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

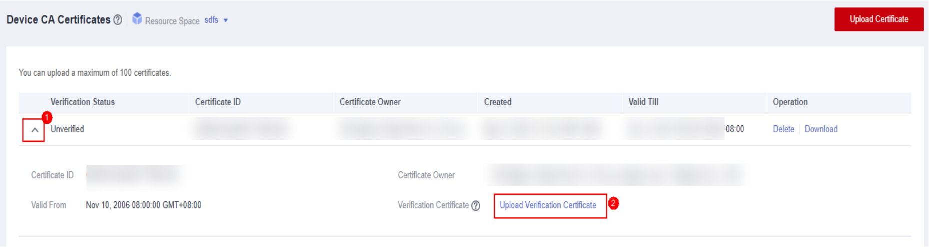
Step 4 Use the CSR to create a verification certificate.

```
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 2-65 Uploading a verification certificate



Device CA Certificates ⓘ Resource Space sdfs ▾ Upload Certificate

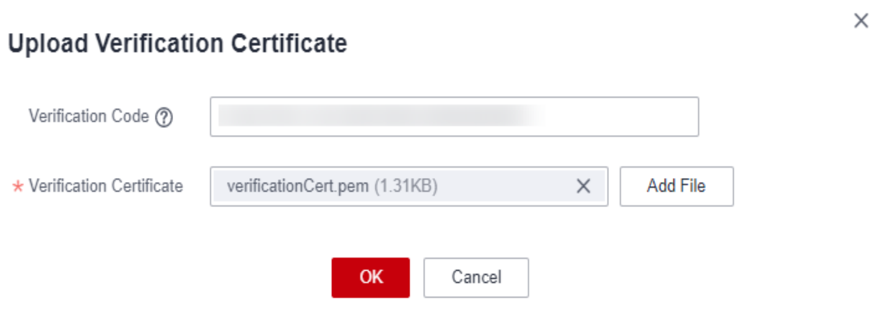
You can upload a maximum of 100 certificates.

Verification Status	Certificate ID	Certificate Owner	Created	Valid Till	Operation
1 ^ Unverified	[redacted]	[redacted]	[redacted]	[redacted] -08:00	Delete | Download

Certificate ID [redacted]
Certificate Owner [redacted]
Valid From Nov 10, 2006 08:00:00 GMT+08:00
Verification Certificate ⓘ Upload Verification Certificate 2

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

Figure 2-66 Completing verification



Upload Verification Certificate ×

Verification Code ⓘ [redacted]

* Verification Certificate verificationCert.pem (1.31KB) × Add File

OK Cancel

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
   ```

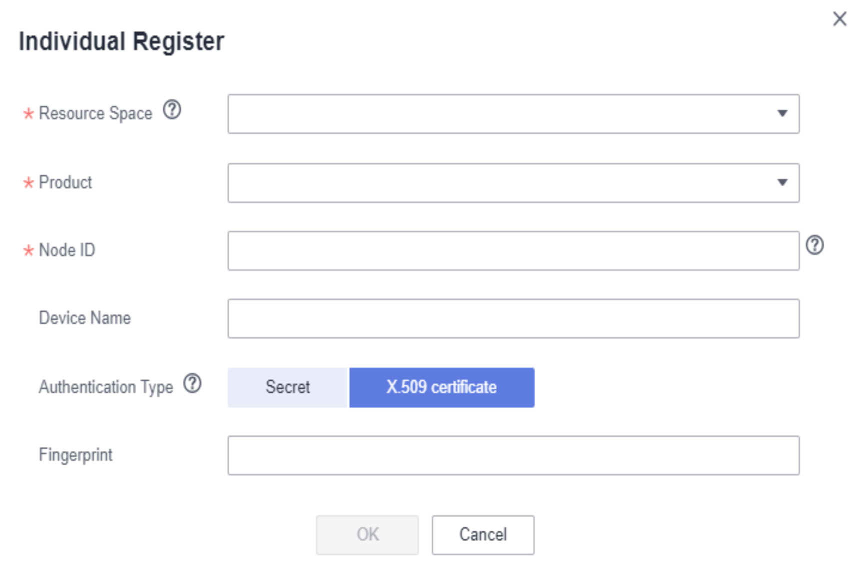
   Obtain the generated device certificate **deviceCert.pem** from the **bin** folder in the OpenSSL installation directory.

Registering a Device Authenticated by an X.509 Certificate

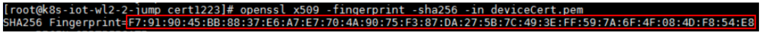
Step 1 Log in to the IoTDA console.

Step 2 In the navigation pane, choose **Devices** > **All Devices**, click **Individual Register** in the upper right corner, set parameters based on the table below, and click **OK**.

Figure 2-67 Registering a 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, create a product 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	**X.509 certificate**: The device uses an X.509 certificate for identity verification.

Fingerprint	This parameter is displayed when **Authentication Type** is set to **X.509 certificate**. Import the fingerprint corresponding to the preset device certificate on the device side. You can run **openssl x509 -fingerprint -sha256 -in deviceCert.pem** in the OpenSSL view to query the fingerprint. **Note: Delete the colon (:) from the obtained fingerprint when filling it.**
	
	`[root@k8s-iot-wl2-2-jump cert1223]# openssl x509 -fingerprint -sha256 -in deviceCert.pem`
	`SHA256 Fingerprint=F7:91:90:45:BB:88:37:E6:A7:E7:70:4A:90:75:F3:87:DA:27:5B:7C:49:3E:FF:59:7A:6F:4F:08:4D:F8:54:E8`

----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

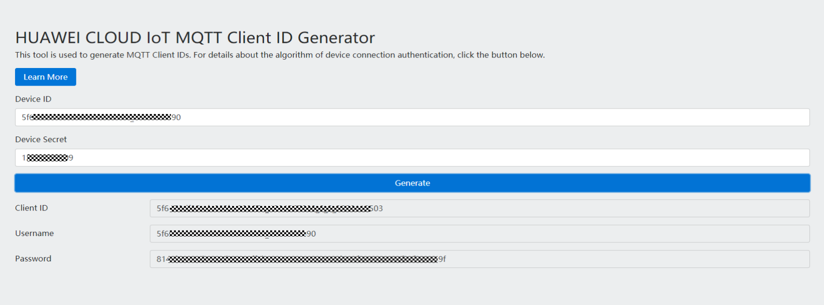
- Install the latest MQTT.fx tool. **Download** it.
- MQTT.fx 1.7.0 and earlier versions have problems in processing topics containing $. Use the latest version for test.

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 2-68 Obtaining ClientId



HUAWEI CLOUD IoT MQTT Client ID Generator

This tool is used to generate MQTT Client IDs. For details about the algorithm of device connection authentication, click the button below.

Device ID: 5f[redacted]90

Device Secret: 1[redacted]9

Client ID: 5f6[redacted]503

Username: 5f6[redacted]90

Password: 814[redacted]9f

Parameter	Mandatory	Type	Description
ClientId	Yes	String(256)	The value of this parameter consists of a device ID, device type, password signature type, and timestamp. They are separated by underscores (_). ● 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 (_). ● Device type: The value is fixed at **0**, indicating a device ID. ● Password signature type: The length is 1 byte, and the value can be **0** or **1**. – **0**: The timestamp is not verified using the HMAC-SHA256 algorithm. – **1**: The timestamp is verified using the HMAC-SHA256 algorithm. ● 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 **2018072417**.
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.

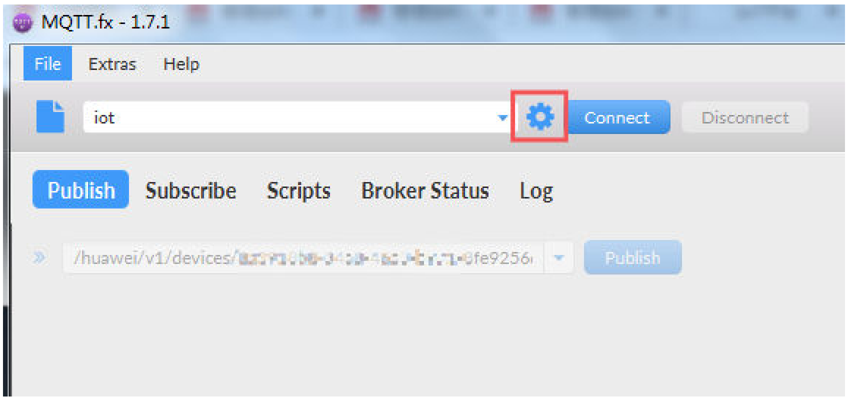
The generated client ID is in the format ***Device ID*_0_0_*Timestamp***. By default, the timestamp is not verified.

- 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 2-69 Settings

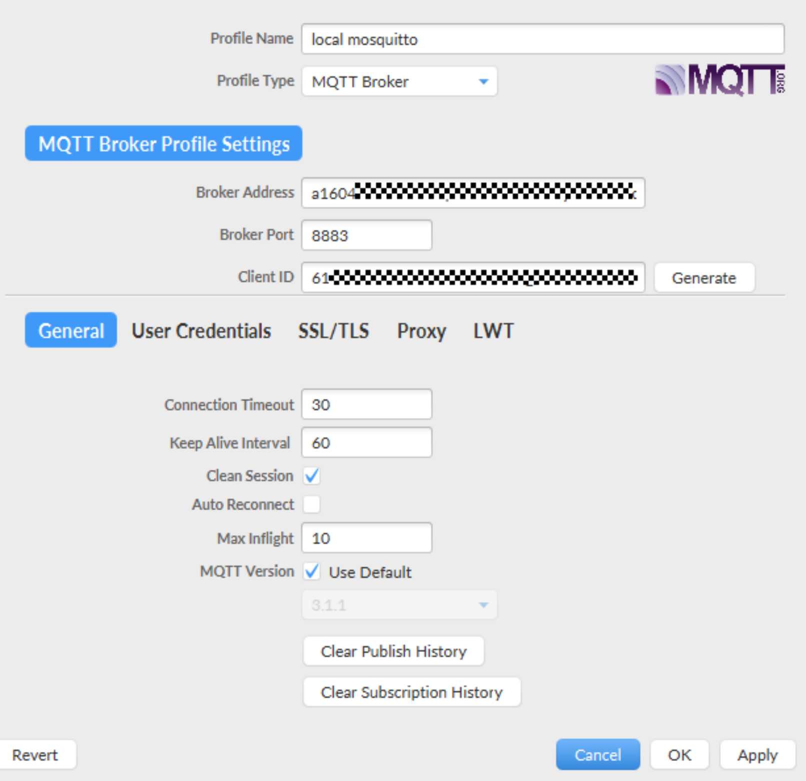


iot

/huawei/v1/devices/[redacted]fe9256

Step 4 Enter **Connection Profile** information.

Figure 2-70 Using default settings for parameters on the General tab page

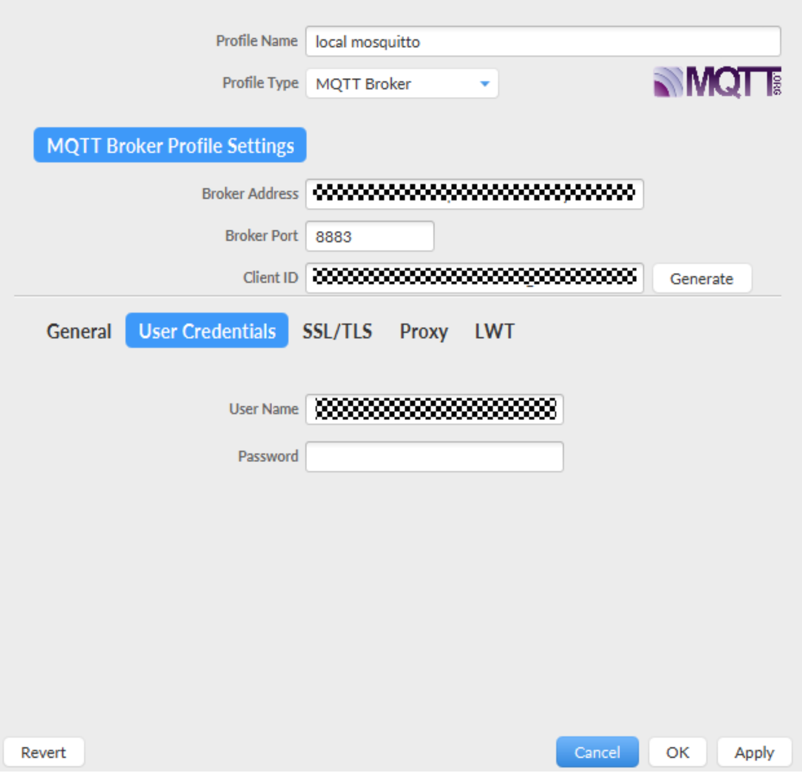


Profile Name: local mosquitto
Profile Type: MQTT Broker
Broker Address: a1604[redacted]
Broker Port: 8883
Client ID: 61[redacted]
Connection Timeout: 30
Keep Alive Interval: 60
Clean Session: ☑
Auto Reconnect: □
Max Inflight: 10
MQTT Version: ☑ Use Default
3.1.1

Parameter	Description
Broker Address	Enter the **device access address** (domain name) obtained from the IoTDA console. If the device cannot be connected using a domain name, enter the IP address obtained in 2.
Broker Port	Enter **8883**.
Cliend ID	Enter the device ClientId obtained in 2.

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

Figure 2-71 Entering the device ID



Profile Name: local mosquitto

Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: [redacted]

Broker Port: 8883

Client ID: [redacted]

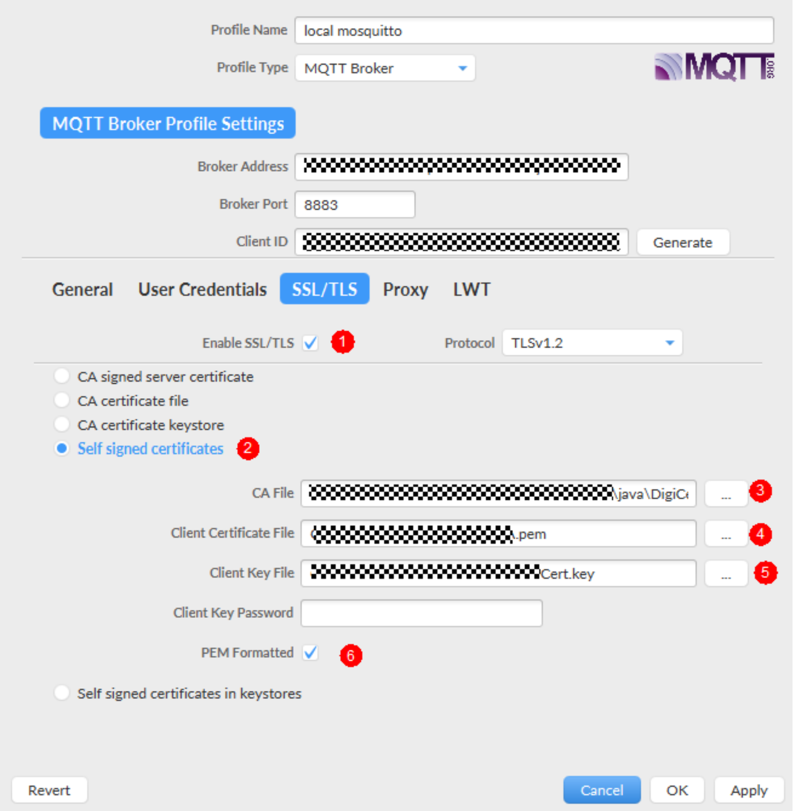
User Name: [redacted]

Password:

Parameter	Description
User Name	Enter the DeviceId 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 2-72 Setting SSL/TLS parameters



Profile Name: local mosquitto

Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: [redacted]

Broker Port: 8883

Client ID: [redacted]

Enable SSL/TLS ☑ (1)   Protocol: TLSv1.2

○ CA signed server certificate

○ CA certificate file

○ CA certificate keystore

● Self signed certificates (2)

CA File: [redacted]\java\DigiC (3)

Client Certificate File: [redacted].pem (4)

Client Key File: [redacted]Cert.key (5)

Client Key Password:

PEM Formatted ☑ (6)

○ Self signed certificates in keystores

NOTE

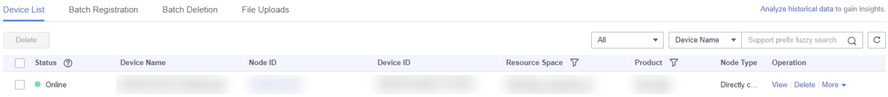
CA File: corresponding CA certificate. Download the certificate from **Obtaining Resources** and load the PEM certificate.

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 2-73 Device status



Status	Device Name	Node ID	Device ID	Resource Space	Product	Node Type	Operation
● Online	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	Directly c...	View | Delete | More

----End

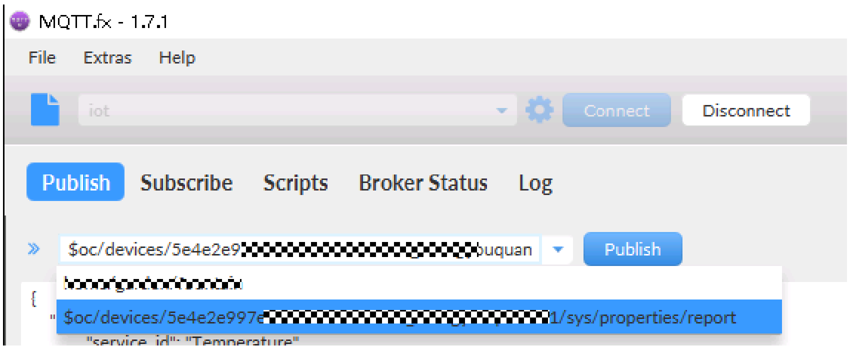
Reporting Data

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 2-74 Entering the API address



$oc/devices/5e4e2e9[redacted]ouquan

[redacted]

$oc/devices/5e4e2e997[redacted]1/sys/properties/report

{
"service_id": "Temperature"

Step 2 Enter the data to report.

Request parameters

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

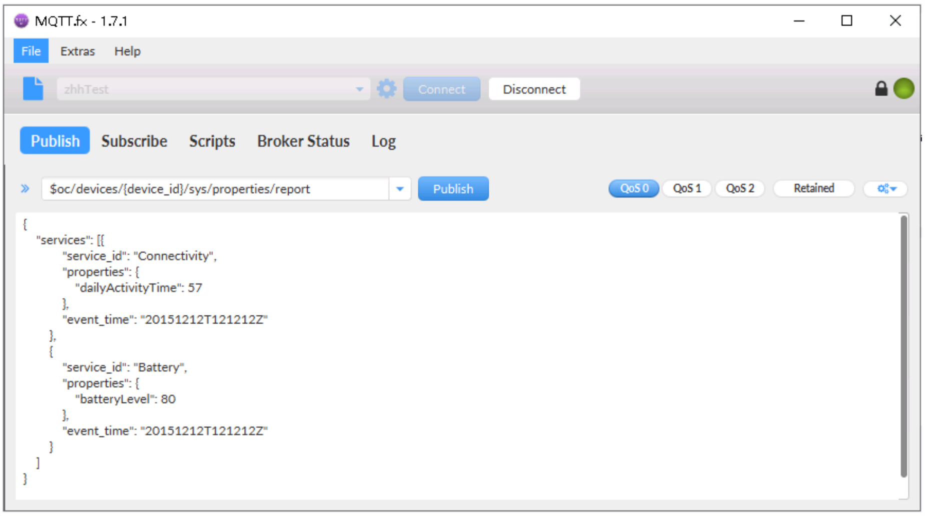
ServiceProperty structure

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 yyyyMMddTHHmmssZ, for example, **20161219T114920Z**. If this parameter is not carried in the reported data or is in incorrect format, the time when IoTDA receives the data is used.

Example request

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

Figure 2-75 Example request



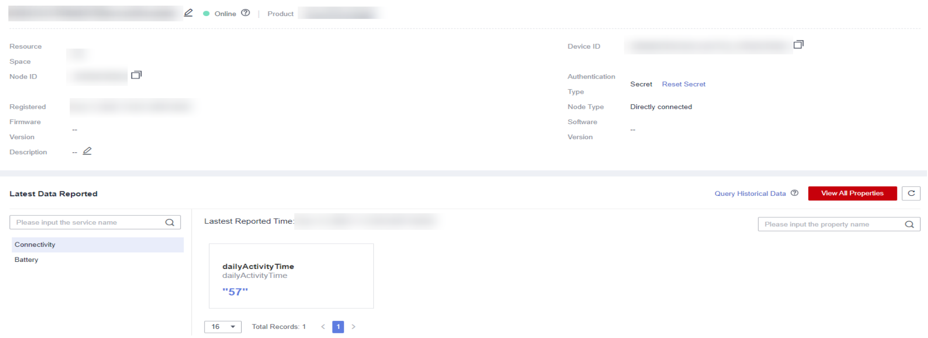
zhhTest

$oc/devices/{device_id}/sys/properties/report

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

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

Figure 2-76 Reporting data



[redacted] Online | Product [redacted]

Resource Space [redacted]

Node ID [redacted]

Registered [redacted]

Firmware Version --

Description --

Device ID [redacted]

Authentication Type Secret Reset Secret

Node Type Directly connected

Software Version --

Latest Data Reported

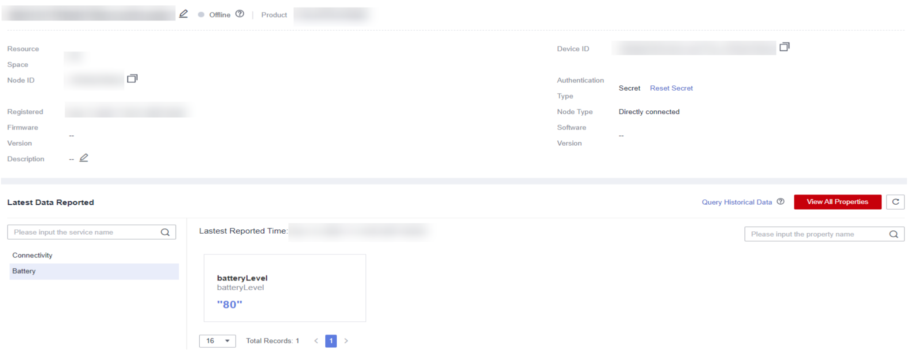
Connectivity

Battery

Lastest Reported Time: [redacted]

dailyActivityTime
dailyActivityTime
"57"

16 Total Records: 1 1



[redacted] Offline | Product [redacted]

Resource Space [redacted]

Node ID [redacted]

Registered [redacted]

Firmware Version --

Description --

Device ID [redacted]

Authentication Type Secret Reset Secret

Node Type Directly connected

Software Version --

Latest Data Reported

Query Historical Data

View All Properties

Please input the service name

Connectivity

Battery

Lastest Reported Time: [redacted]

Please input the property name

batteryLevel
batteryLevel
"80"

16 Total Records: 1 < 1 >

----End

2.11 Connecting an OpenHarmony 3.0 Device to IoTDA

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

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.

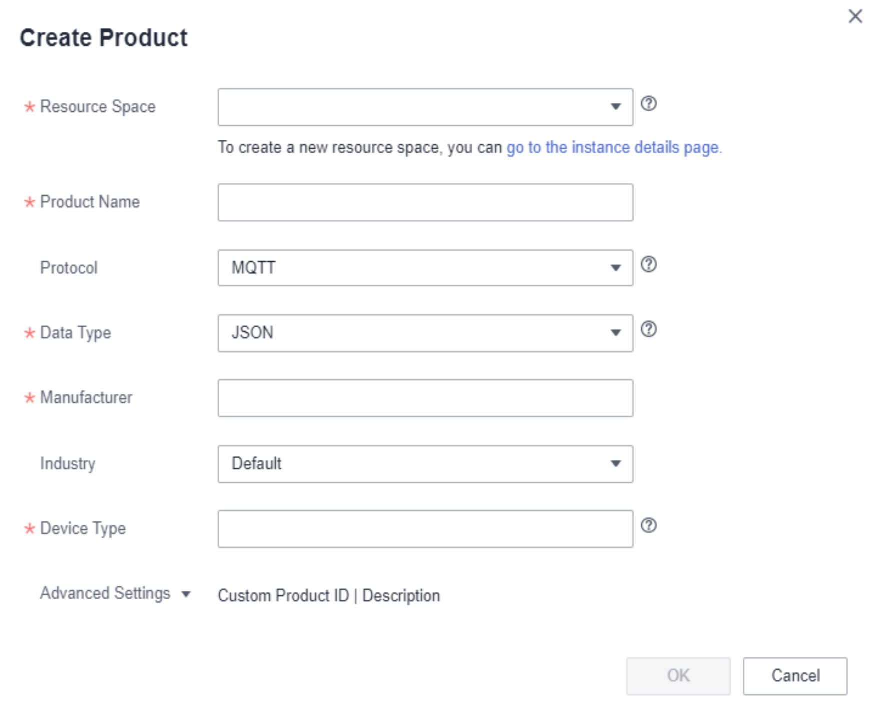
Creating a Product

Step 1 Log in to the **console**, choose **Products** in the navigation pane, and click **Create Product** in the upper right corner.

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. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create it first.
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Industry	Set this parameter based on the site requirements. If the product model preset on the platform is used, set this parameter based on the industry to which the product model belongs. If not, select **Default**.
Device Type	Set this parameter based on the site requirements. If the product model preset on the platform is used, the device type is automatically matched.
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.
Description	Provide a description for the product. Set this parameter based on the site requirements.

Figure 2-77 Creating a product



Create Product

* Resource Space
To create a new resource space, you can go to the instance details page.

* Product Name

Protocol: MQTT

* Data Type: JSON

* Manufacturer

Industry: Default

* Device Type

Advanced Settings ▾ Custom Product ID | Description

----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 of the product details page, upload a model file **BearPi_Agriculture.zip**.

NOTE

1. In the product list, click the name of a product to access its details. On the product details page displayed, you can view basic product information, such as the product ID, product name, device type, data format, manufacturer name, resource space, and protocol type. The product ID is automatically generated by the platform. Other information is defined by users during **product creation**.
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.

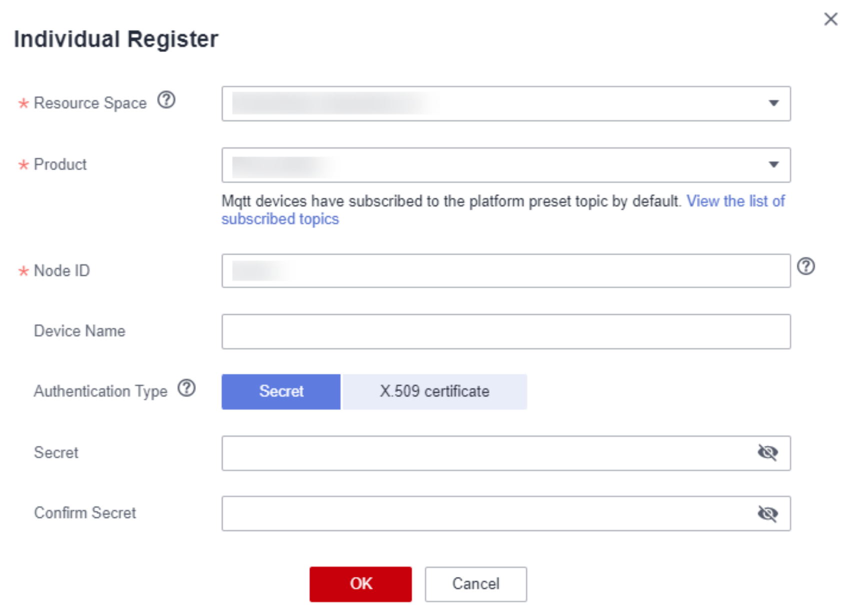
----End

Registering a Device

Step 1 Log in to the **IoTDA console**.

Step 2 In the navigation pane, choose **Devices** > **All Devices**, click **Individual Register** in the upper right corner, set parameters based on the table below, and click **OK**.

Figure 2-78 Registering an individual device



Individual Register

* Resource Space
* Product

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID

Device Name

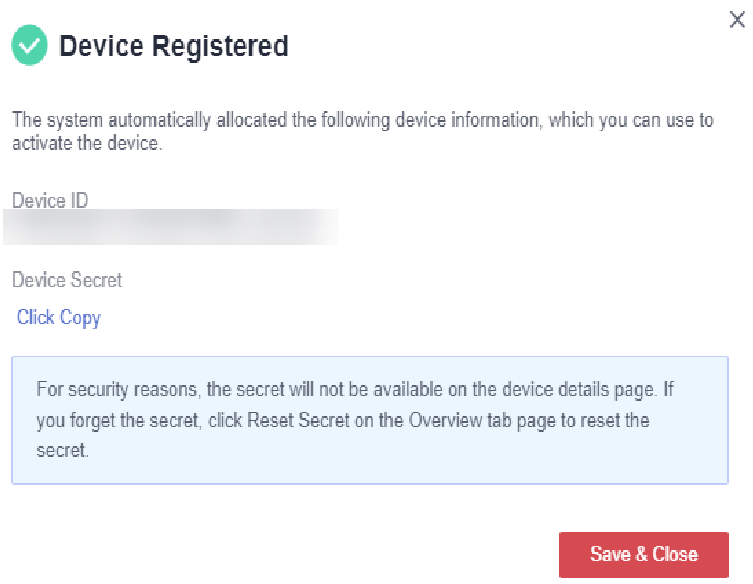
Authentication Type: Secret | X.509 certificate

Secret

Confirm Secret

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, **create a product** 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	Select **Secret**.
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

Figure 2-79 Obtaining the device secret



Device Registered

The system automatically allocated the following device information, which you can use to activate the device.

Device ID

[redacted]

Device Secret

For security reasons, the secret will not be available on the device details page. If you forget the secret, click Reset Secret on the Overview tab page to reset the secret.

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

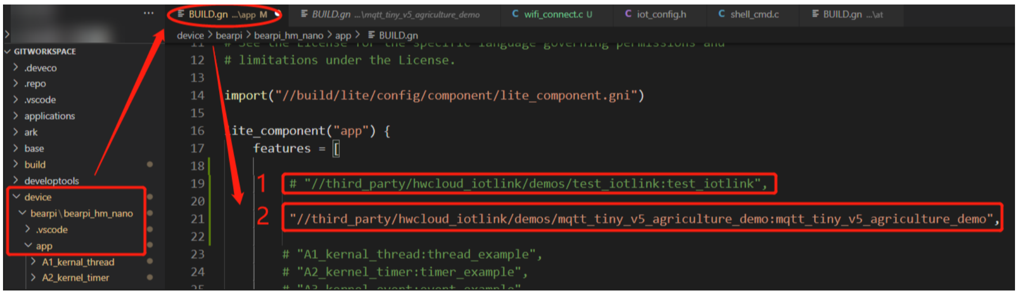
Step 1 Download the source code **oh3.0_hwcloud_iotlink-master.zip**.

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 2-80 Selecting a demo

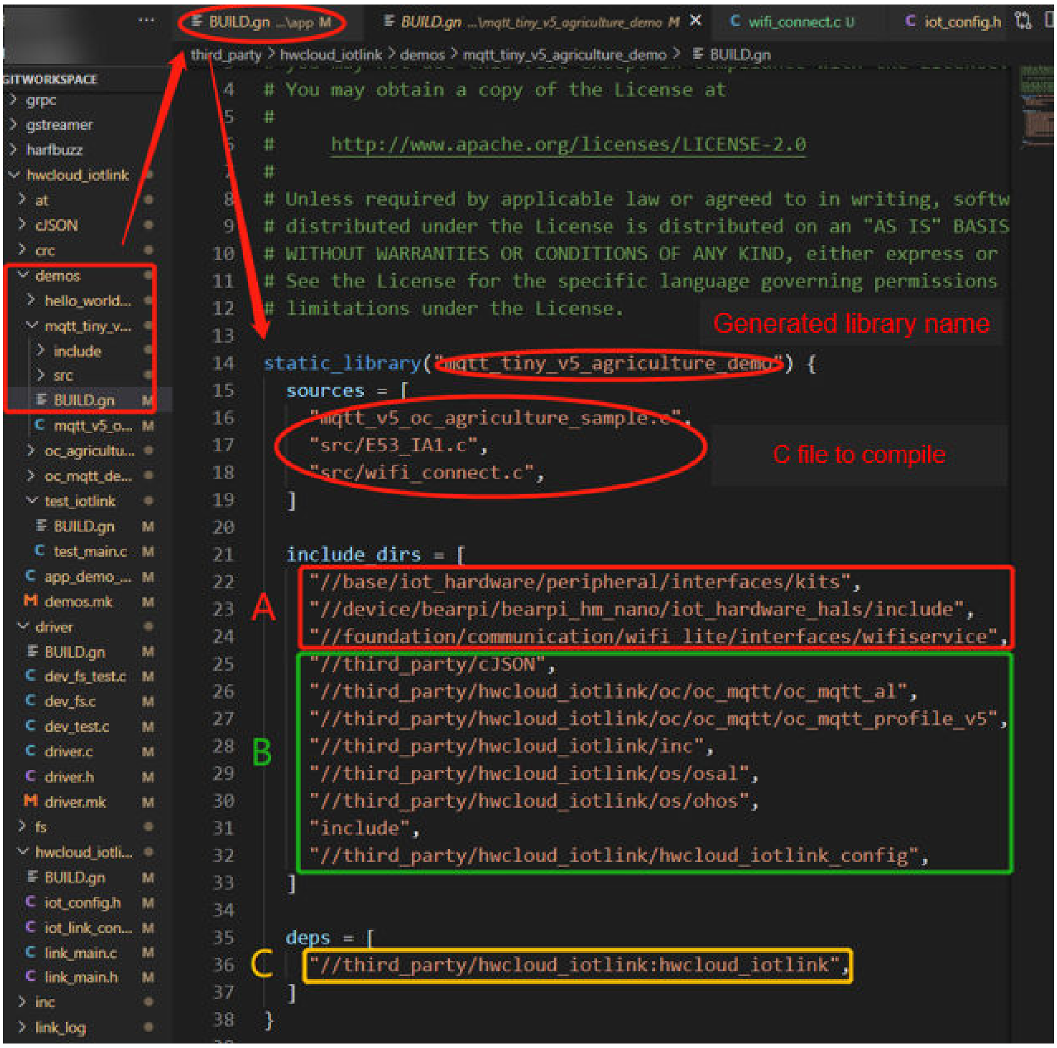


```
# limitations under the License.

import("//build/lite/config/component/lite_component.gni")

ite_component("app") {
    features = [

        # "//third_party/hwcloud_iotlink/demos/test_iotlink:test_iotlink",

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

        # "A1_kernal_thread:thread_example",
        # "A2_kernel_timer:timer_example",
```

NOTE

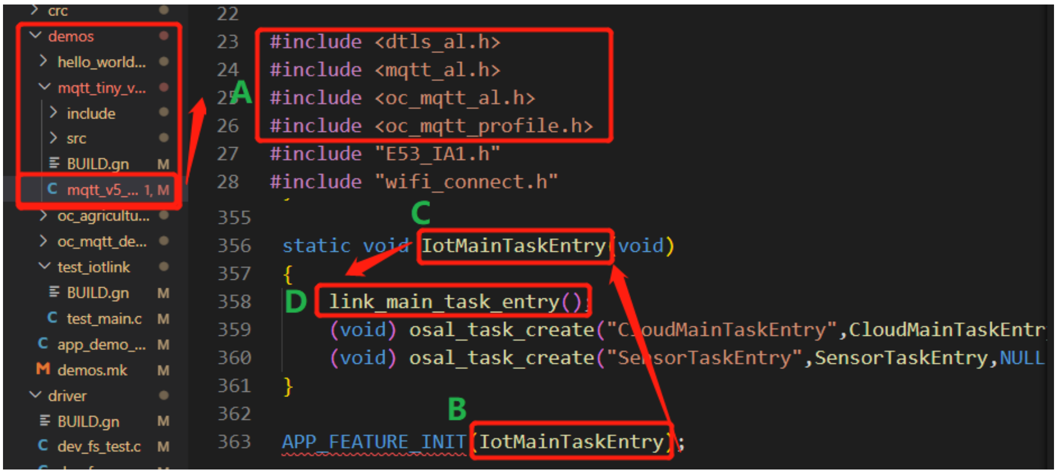
2 in Figure 6 is the demo of connecting an MQTT device to Huawei Cloud. Check the **BUILD.gn** file, as shown in **Figure 2-81**. **A** contains the library files related to the development board hardware and Wi-Fi. **B** contains the library files required for connecting an MQTT device to Huawei Cloud, 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 2-81 Code compilation file



```
# You may obtain a copy of the License at
#
#     http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, softw
# distributed under the License is distributed on an "AS IS" BASIS
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# See the License for the specific language governing permissions
# limitations under the License.

static_library("mqtt_tiny_v5_agriculture_demo") {
  sources = [
    "mqtt_v5_oc_agriculture_sample.c",
    "src/E53_IA1.c",
    "src/wifi_connect.c",
  ]

  include_dirs = [
    "//base/iot_hardware/peripheral/interfaces/kits",
    "//device/bearpi/bearpi_hm_nano/iot_hardware_hals/include",
    "//foundation/communication/wifi_lite/interfaces/wifiservice",
    "//third_party/cJSON",
    "//third_party/hwcloud_iotlink/oc/oc_mqtt/oc_mqtt_al",
    "//third_party/hwcloud_iotlink/oc/oc_mqtt/oc_mqtt_profile_v5",
    "//third_party/hwcloud_iotlink/inc",
    "//third_party/hwcloud_iotlink/os/osal",
    "//third_party/hwcloud_iotlink/os/ohos",
    "include",
    "//third_party/hwcloud_iotlink/hwcloud_iotlink_config",
  ]

  deps = [
    "//third_party/hwcloud_iotlink:hwcloud_iotlink",
  ]
}
```

Figure 2-82 Main function file of the demo



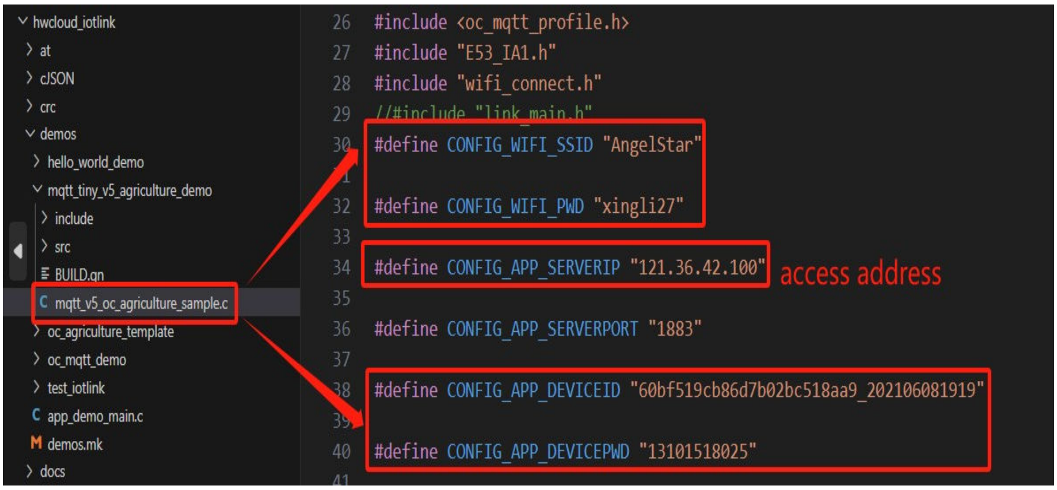
```
#include <dtls_al.h>
#include <mqtt_al.h>
#include <oc_mqtt_al.h>
#include <oc_mqtt_profile.h>
#include "E53_IA1.h"
#include "wifi_connect.h"

 static void IotMainTaskEntry(void)
 {
     link_main_task_entry();
     (void) osal_task_create("CloudMainTaskEntry",CloudMainTaskEntr
     (void) osal_task_create("SensorTaskEntry",SensorTaskEntry,NULL
 }

 APP_FEATURE_INIT(IotMainTaskEntry);
```

NOTE

A in **Figure 2-82** 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 2-83 Modifying parameters



```
#include <oc_mqtt_profile.h>
#include "E53_IA1.h"
#include "wifi_connect.h"
//#include "link_main.h"
#define CONFIG_WIFI_SSID "AngelStar"

#define CONFIG_WIFI_PWD "xingli27"

#define CONFIG_APP_SERVERIP "121.36.42.100"

#define CONFIG_APP_SERVERPORT "1883"

#define CONFIG_APP_DEVICEID "60bf519cb86d7b02bc518aa9_202106081919"

#define CONFIG_APP_DEVICEPWD "13101518025"
```

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 key 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.

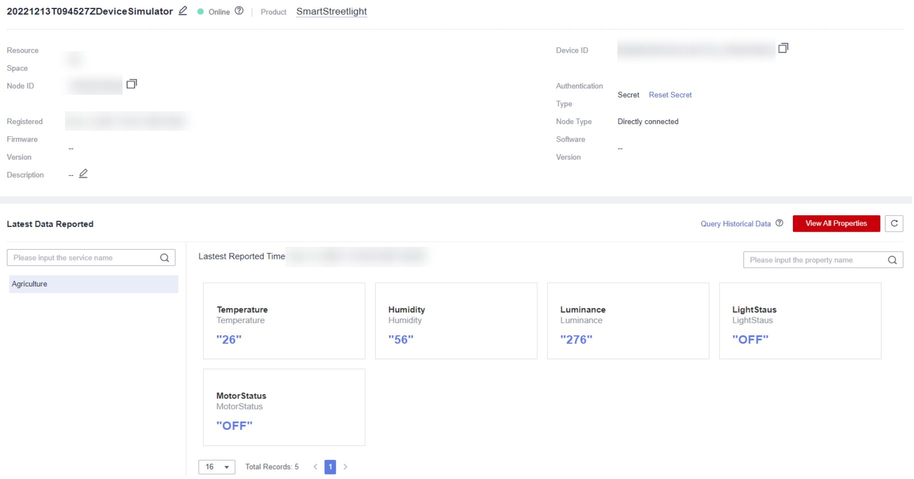
----End

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 2-84 Data reporting



20221213T094527ZDeviceSimulator Online | Product SmartStreetlight

Resource Space [redacted]

Node ID [redacted]

Registered [redacted]

Firmware Version --

Description --

Device ID [redacted]

Authentication Type Secret Reset Secret

Node Type Directly connected

Software Version --

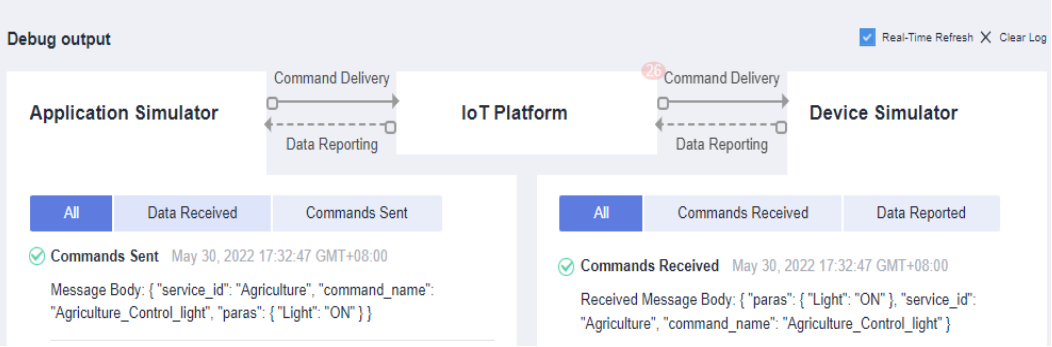
Latest Data Reported

Lastest Reported Time [redacted]

Temperature	Humidity	Luminance	LightStaus	MotorStatus
"26"	"56"	"276"	"OFF"	"OFF"

Total Records: 5

Figure 2-85 Command delivery



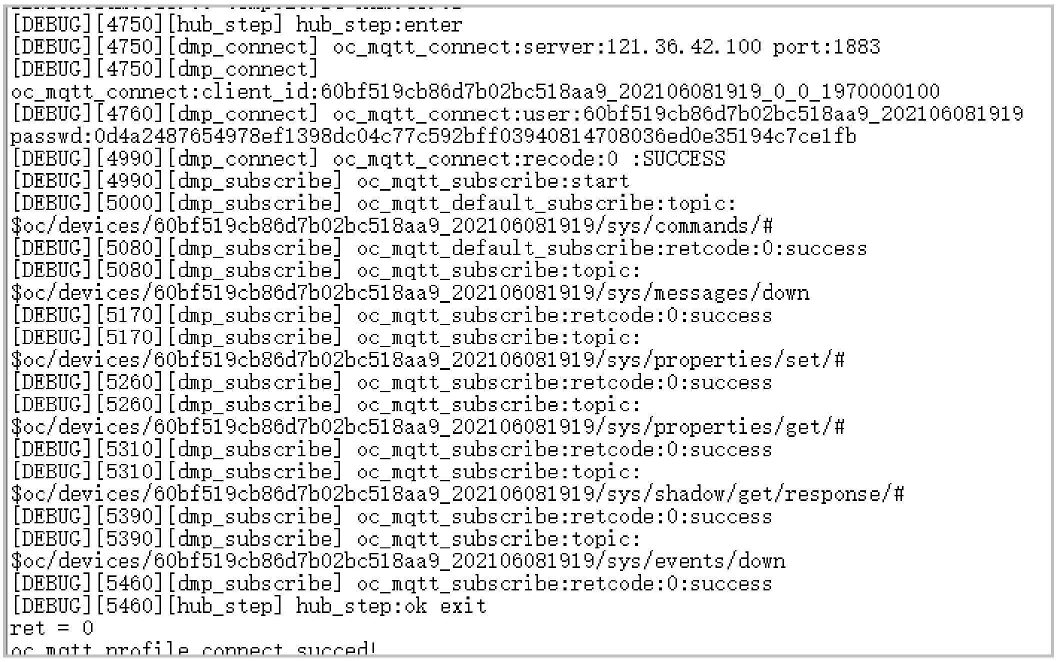
✓ Commands Sent May 30, 2022 17:32:47 GMT+08:00

Message Body: { "service_id": "Agriculture", "command_name": "Agriculture_Control_light", "paras": { "Light": "ON" } }

✓ Commands Received May 30, 2022 17:32:47 GMT+08:00

Received Message Body: { "paras": { "Light": "ON" }, "service_id": "Agriculture", "command_name": "Agriculture_Control_light" }

Figure 2-86 Log information



```
[DEBUG][4750][hub_step] hub_step:enter
[DEBUG][4750][dmp_connect] oc_mqtt_connect:server:121.36.42.100 port:1883
[DEBUG][4750][dmp_connect]
oc_mqtt_connect:client_id:60bf519cb86d7b02bc518aa9_202106081919_0_0_1970000100
[DEBUG][4760][dmp_connect] oc_mqtt_connect:user:60bf519cb86d7b02bc518aa9_202106081919
passwd:0d4a2487654978ef1398dc04c77c592bff039408147080036ed0e35194c7ce1fb
[DEBUG][4990][dmp_connect] oc_mqtt_connect:recode:0 :SUCCESS
[DEBUG][4990][dmp_subscribe] oc_mqtt_subscribe:start
[DEBUG][5000][dmp_subscribe] oc_mqtt_default_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/commands/#
[DEBUG][5080][dmp_subscribe] oc_mqtt_default_subscribe:retcode:0:success
[DEBUG][5080][dmp_subscribe] oc_mqtt_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/messages/down
[DEBUG][5170][dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG][5170][dmp_subscribe] oc_mqtt_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/properties/set/#
[DEBUG][5260][dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG][5260][dmp_subscribe] oc_mqtt_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/properties/get/#
[DEBUG][5310][dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG][5310][dmp_subscribe] oc_mqtt_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/shadow/get/response/#
[DEBUG][5390][dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG][5390][dmp_subscribe] oc_mqtt_subscribe:topic:
$oc/devices/60bf519cb86d7b02bc518aa9_202106081919/sys/events/down
[DEBUG][5460][dmp_subscribe] oc_mqtt_subscribe:retcode:0:success
[DEBUG][5460][hub_step] hub_step:ok exit
ret = 0
oc_mqtt_profile_connect_succed!
```

2.12 Using a Custom Topic for Communication

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. For details, see **Register Huawei ID**.

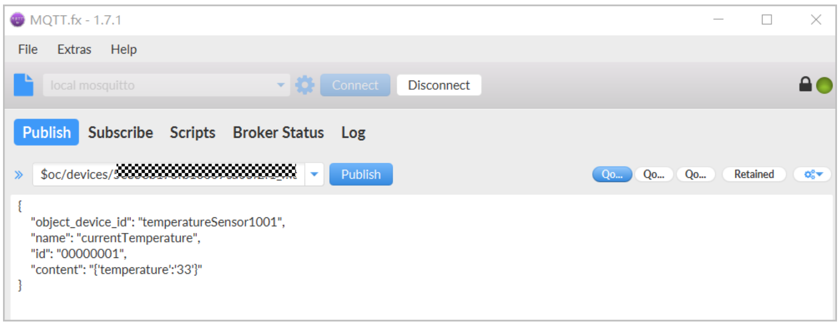
- You have subscribed to the IoTDA service.
- You have created an MQTT product, developed a product model, and registered a device on the IoTDA **console**. For details, see **Products**, **Developing a Product Model Online**, **Registering an Individual Device**, or **Registering a Batch of Devices**.
- 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** product page and click **Access Console**.
2. In the navigation pane, choose **Devices** > **All Devices**, find the corresponding device, and click **View** to access its details.
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**.



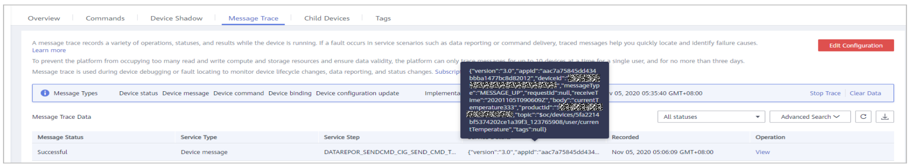
$oc/devices/[redacted]

```
{
    "object_device_id": "temperatureSensor1001",
    "name": "currentTemperature",
    "id": "00000001",
    "content": "['temperature':'33']"
}
```

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.



A message trace records a variety of operations, statuses, and results while the device is running. If a fault occurs in service scenarios such as data reporting or command delivery, traced messages help you quickly locate and identify failure causes. Learn more

To prevent the platform from occupying too many read and write compute and storage resources and ensure data validity, the platform can only trace messages for up to 10 devices at a time for a single user, and for no more than three days.

Message trace is used during device debugging or fault locating to monitor device lifecycle changes, data reporting, and status changes. Subscrip

Message Types Device status Device message Device command Device binding Device configuration update Implementa v 05, 2020 05:35:40 GMT+08:00

```
{"version":"3.0","appId":"aac7a75845dd434bbba1477bc8d82012","deviceId":"[redacted]","messageType":"MESSAGE_UP","requestId":null,"receiveTime":"20201105T090609Z","body":"currentTemperature333","productId":"[redacted]","topic":"$oc/devices/5fa2214bf5374202ce1a39f3_123765908/user/currentTemperature","tags":null}
```

Message Trace Data

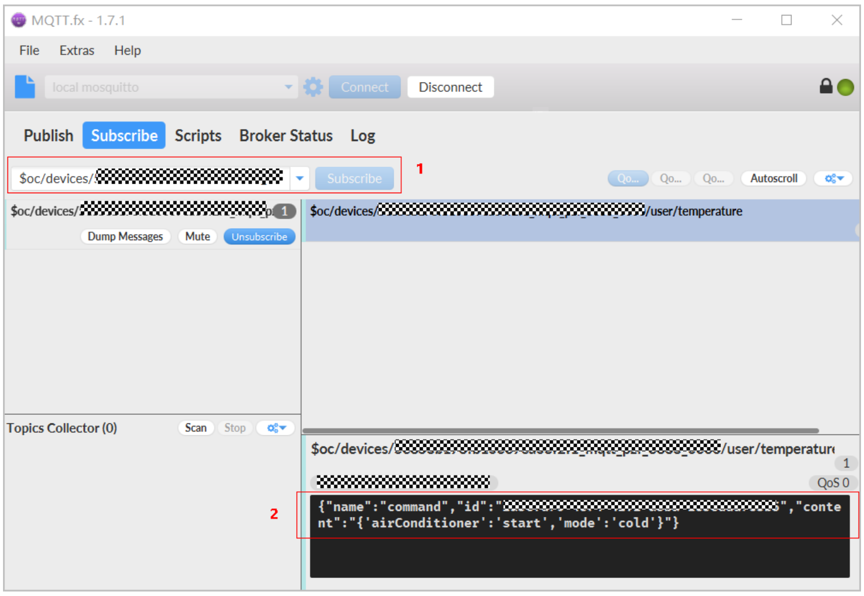
Message Status	Service Type	Service Step		Recorded	Operation
Successful	Device message	DATAREPOR_SENDCMD_CIG_SEND_CMD_T...	{"version":"3.0","appId":"aac7a75845dd434...	Nov 05, 2020 05:06:09 GMT+08:00	View

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.



$oc/devices/[redacted]

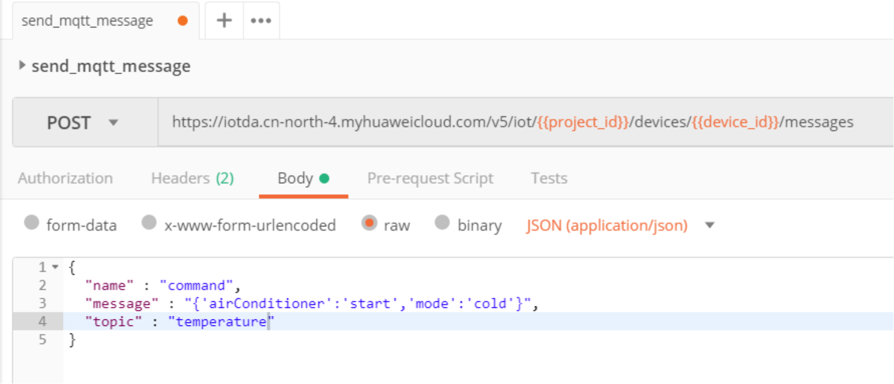
$oc/devices/[redacted]/user/temperature

```
{"name":"command","id":"[redacted]","content":"{'airConditioner':'start','mode':'cold'}"}
```

📖 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.

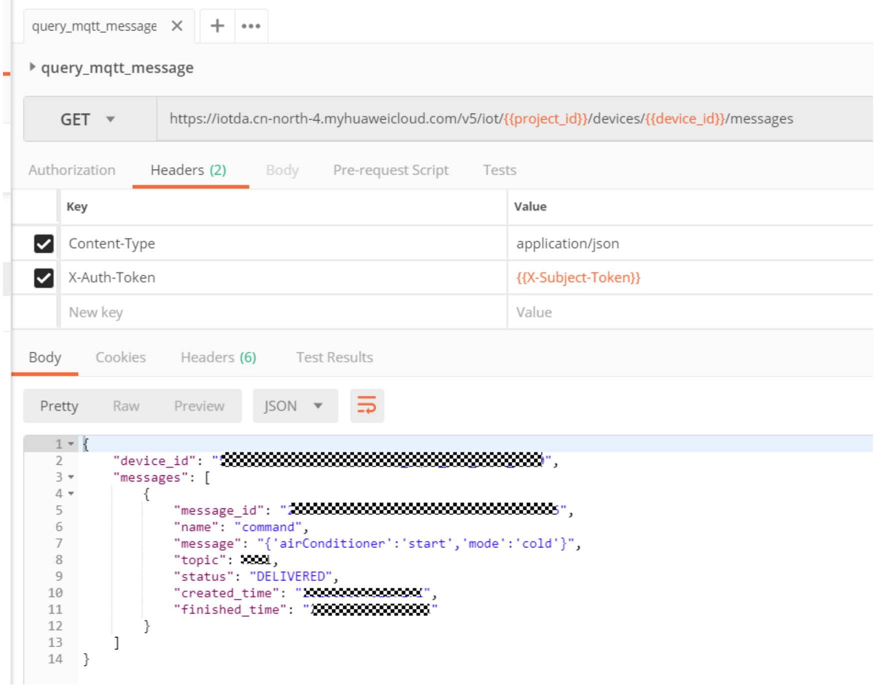
2. Use the Postman to simulate the application to call the API **Delivering a Message to a Device** to deliver a command for starting the indoor air conditioner.



POST https://iotda.cn-north-4.myhuaweicloud.com/v5/iot/{{project_id}}/devices/{{device_id}}/messages

```
{
  "name" : "command",
  "message" : "{'airConditioner':'start','mode':'cold'}",
  "topic" : "temperature"
}
```

3. Call the API **Querying Device Messages** to check whether the command is delivered. If the command is delivered, the indoor air conditioner will be started.



query_mqtt_message

GET https://iotda.cn-north-4.myhuaweicloud.com/v5/iot/{{project_id}}/devices/{{device_id}}/messages

	Key	Value
☑	Content-Type	application/json
☑	X-Auth-Token	{{X-Subject-Token}}
	New key	Value

```
{
    "device_id": "[redacted]",
    "messages": [
        {
            "message_id": "[redacted]",
            "name": "command",
            "message": "{'airConditioner':'start','mode':'cold'}",
            "topic": [redacted],
            "status": "DELIVERED",
            "created_time": "[redacted]",
            "finished_time": "[redacted]"
        }
    ]
}
```

2.13 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 topic 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 2-9 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. If you have not registered, click **here** to complete the registration.
- You have subscribed to the IoTDA service. If not, access **IoTDA** and buy an SU2 unit (10,000 online devices and 100 TPS upstream and downstream messages).

Preparations

- Install the Java running environment on the JMeter executor. Visit the **Java website**, and download and install the Java running 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.

Procedure

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** in the upper right corner.

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

Table 2-10 Parameters

Basic Information	
Resource Space	The platform automatically allocates the created product to the default resource space. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create one.
Product Name	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value. The name can contain letters, numbers, underscores (_), and hyphens (-).
Industry	Select the industry to which the product model belongs.
Device Type	If the product model preset on the platform is used, the device type is automatically matched.
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.
Description	Provide a description for the product. Set this parameter based on the site requirements.

----End

Registering a Batch of Devices

Step 1 Visit the **IoTDA** product 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**.

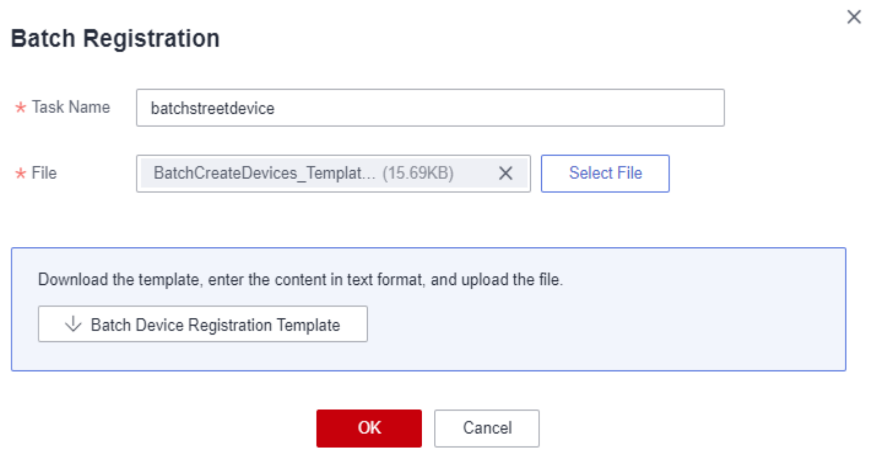
Table 2-11 Parameters

Parameter	Description
node_id	Device identifier. Set this parameter in ascending order, for example, **10001**, **10002**, and **10003**.
product_id	Product ID generated in **Creating a Product**.

Parameter	Description
app_id	Resource space. For details about how to obtain the resource space, see **Resource Spaces**.
device_name	Device name, which can be the same as the value of **node_id**.
secret	Device secret. You can hardcode a secret for the performance test.

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

Figure 2-87 Registering a batch of devices



Batch Registration

* Task Name: batchstreetdevice

* File: BatchCreateDevices_Templat... (15.69KB)

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

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.

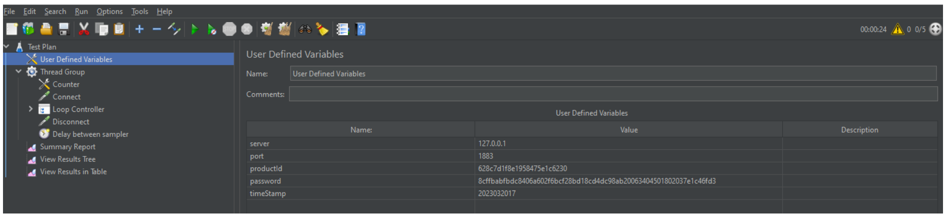
Step 3 In the JMeter directory on the left, choose **User Defined Variables**. On the **User Defined Variables** page, configure the following parameters:

Table 2-12 Parameters

Parameter	Description
server	MQTT server address. For details about how to obtain the value, see **Obtaining Resources**.
port	MQTT port number. Set this parameter to **8883**.
productId	Product ID generated in **Creating a Product**.

Parameter	Description
password	MQTT connection password, which is the value of **secret** encrypted using the HMAC-SHA256 algorithm with the timestamp as the key. **secret** is the secret entered during **batch device registration**. You can use this **tool** to obtain the encrypted value.
timeStamp	Timestamp used for encrypting the password. The time format is YYYYMMDDHH.

Figure 2-88 Example



User Defined Variables

Name: User Defined Variables

Comments:

User Defined Variables

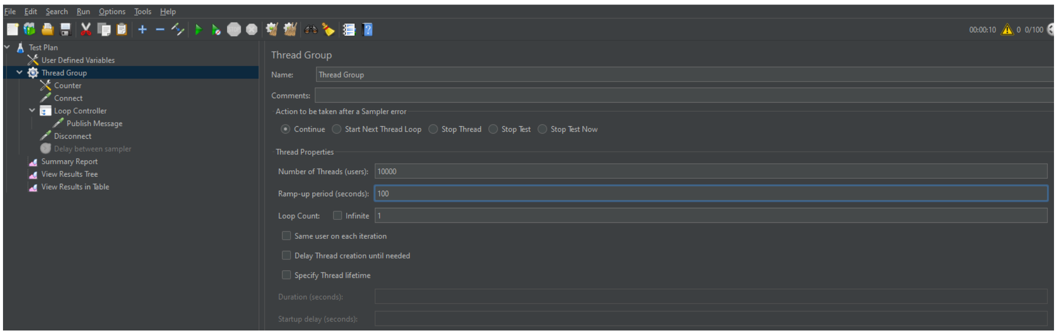
Name:	Value	Description
server	127.0.0.1	
port	1883	
productId	628c7d1f8e1958475e1c6230	
password	8cffbabfbdc8406a602f6bcf28bd18cd4dc98ab20063404501802037e1c46fd3	
timeStamp	2023032017	

----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 2-89 Configuring devices



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

Number of Threads (users): 10000

Ramp-up period (seconds): 100

Loop Count: □ Infinite 1

□ Same user on each iteration

□ Delay Thread creation until needed

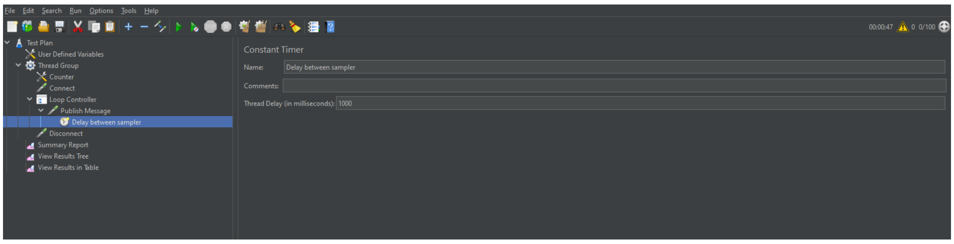
□ Specify Thread lifetime

Duration (seconds):

Startup delay (seconds):

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

Figure 2-90 Configuring devices



Constant Timer

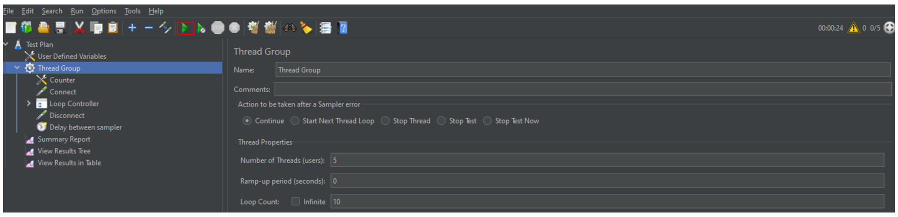
Name: Delay between sampler

Comments:

Thread Delay (in milliseconds): 1000

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

Figure 2-91 Performance test



Thread Group

Name: Thread Group

Comments:

Action to be taken after a Sampler error

● Continue ○ Start Next Thread Loop ○ Stop Thread ○ Stop Test ○ Stop Test Now

Thread Properties

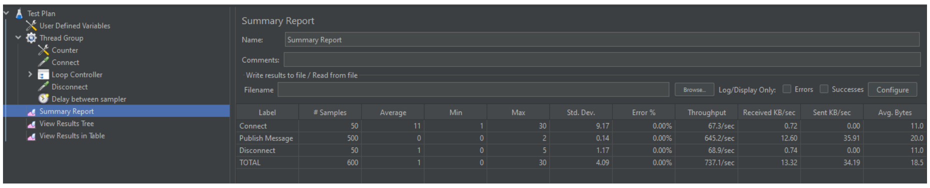
Number of Threads (users): 5

Ramp-up period (seconds): 0

Loop Count: □ Infinite 10

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 2-92 Performance test



Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename Browse... Log/Display Only: □ Errors □ Successes Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	Received KB/sec	Sent KB/sec	Avg. Bytes
Connect	50	11	1	30	9.17	0.00%	67.3/sec	0.72	0.00	11.0
Publish Message	500	0	0	2	0.14	0.00%	645.2/sec	12.60	35.91	20.0
Disconnect	50	1	0	5	1.17	0.00%	68.9/sec	0.74	0.00	11.0
TOTAL	600	1	0	30	4.09	0.00%	737.1/sec	13.32	34.19	18.5

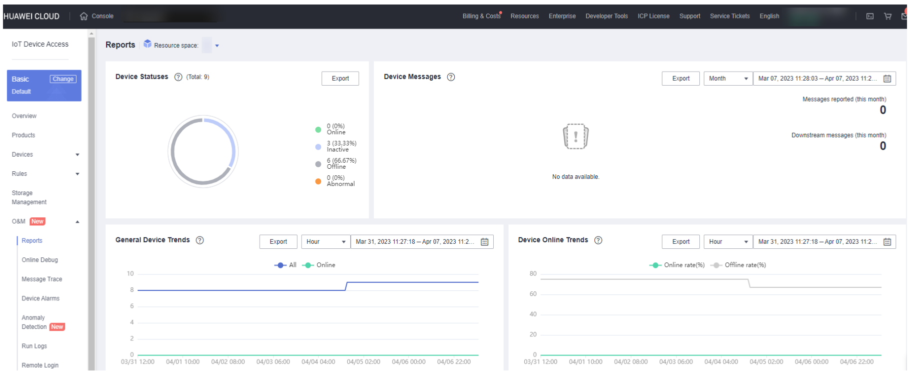
Step 5 After the JMeter test plan is debugged, import the test plan to CodeArts PerfTest for distributed deployment to meet requirements of higher-level performance tests.

----End

Viewing the Pressure Test Result

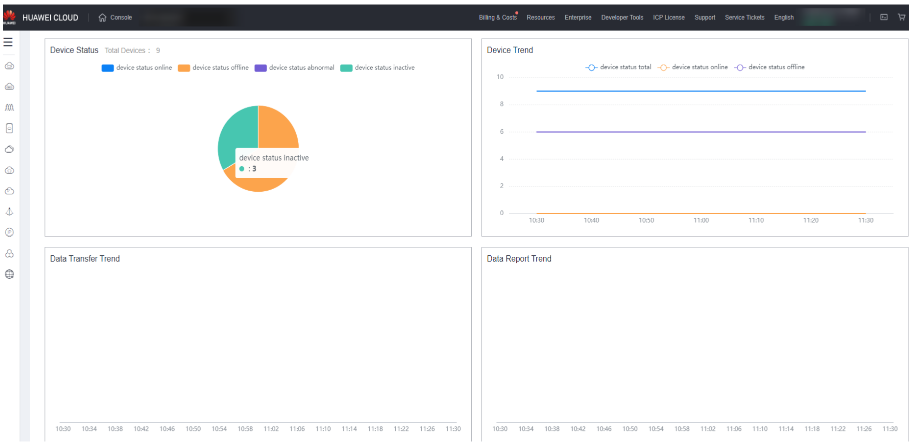
1. Log in to the IoTDA console and choose **O&M** > **Reports** in the navigation pane to view statistical metrics of the platform.

   Figure 2-93 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 2-94 Viewing metrics



3 Device Linkage

3.1 Triggering Alarms and Sending Email or SMS Notifications

Scenarios

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.

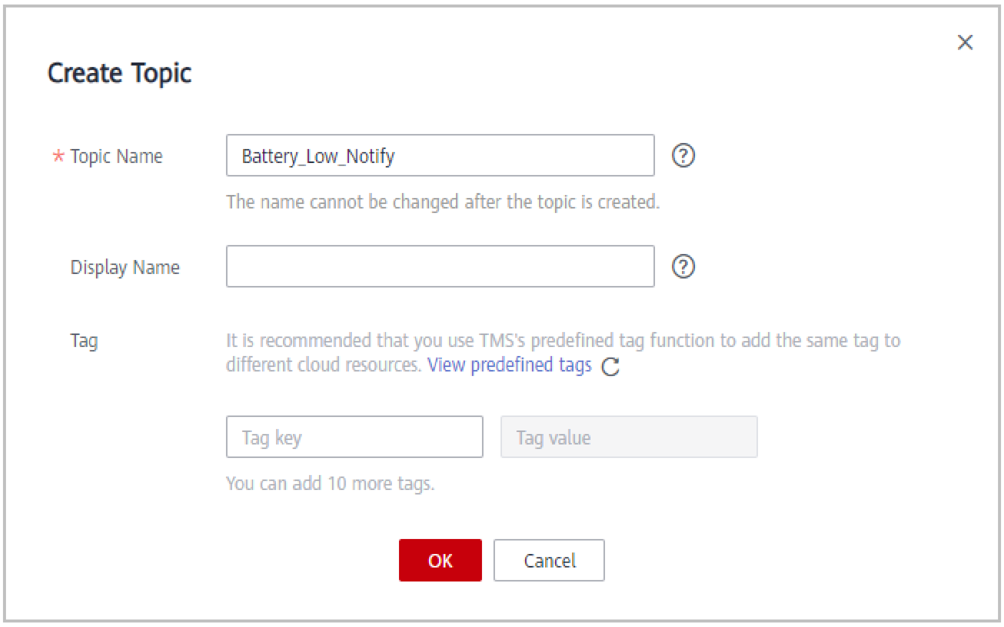
In this example, when the battery level reported by the device is lower than 20%, the IoT platform reports an alarm and sends an email or SMS notification to the specified mobile number.

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 visit **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, **Battery_Low_Notify**, and click **OK**.

Figure 3-1 Parameters for creating a topic



Create Topic

* Topic Name: Battery_Low_Notify

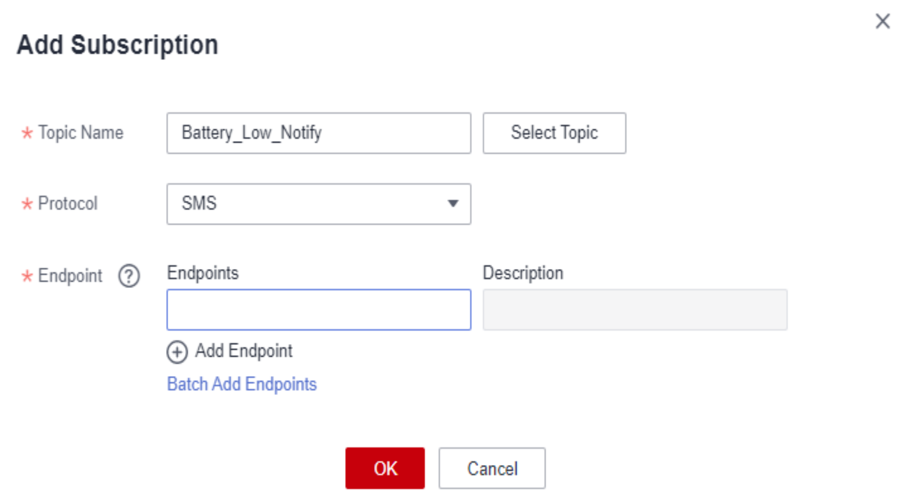
The name cannot be changed after the topic is created.

Display Name:

Tag: It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. View predefined tags

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

Figure 3-2 Parameters for adding a subscription



Add Subscription

* Topic Name: Battery_Low_Notify

* Protocol: SMS

* Endpoint

Table 3-1 Parameters for adding a subscription

Parameter	Description
Topic Name	Select the topic created in 4.
Protocol	● To send an email notification, select **Email**. ● To send an SMS notification, select **SMS**.

Parameter	Description
Endpoint	● If **Protocol** is set to **Email**, enter the email address for receiving notifications. ● If **Protocol** is set to **SMS**, enter the mobile number for receiving notifications. 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**.
2. In the navigation pane, choose **Products**.

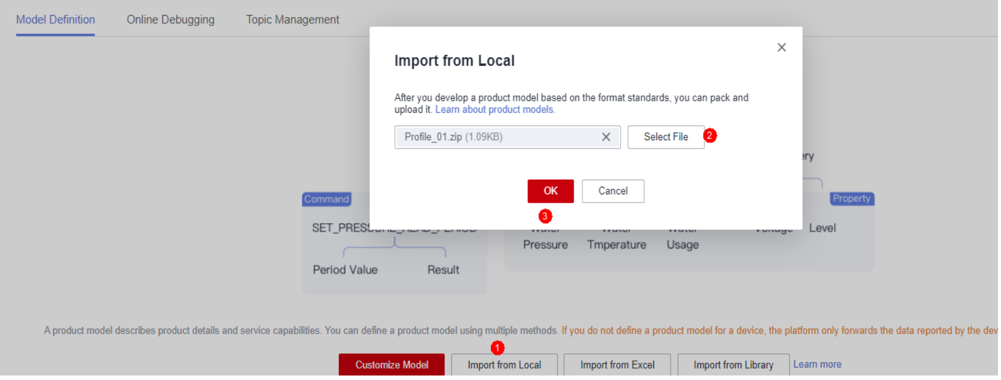
   Note: The product model and device used in this document are only examples. You can use your own product model and device.
3. Click **Create Product** in the upper right corner to create a product using MQTT. Set the parameters and click **Create**.

Table 3-2 Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, **MQTT_Device**.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value.
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 **Model Definition** tab page, click **Import from Local**. In the dialog box displayed, load the local product model and click **OK**.

Figure 3-3 Uploading a model file



Import from Local

After you develop a product model based on the format standards, you can pack and upload it. Learn about product models.

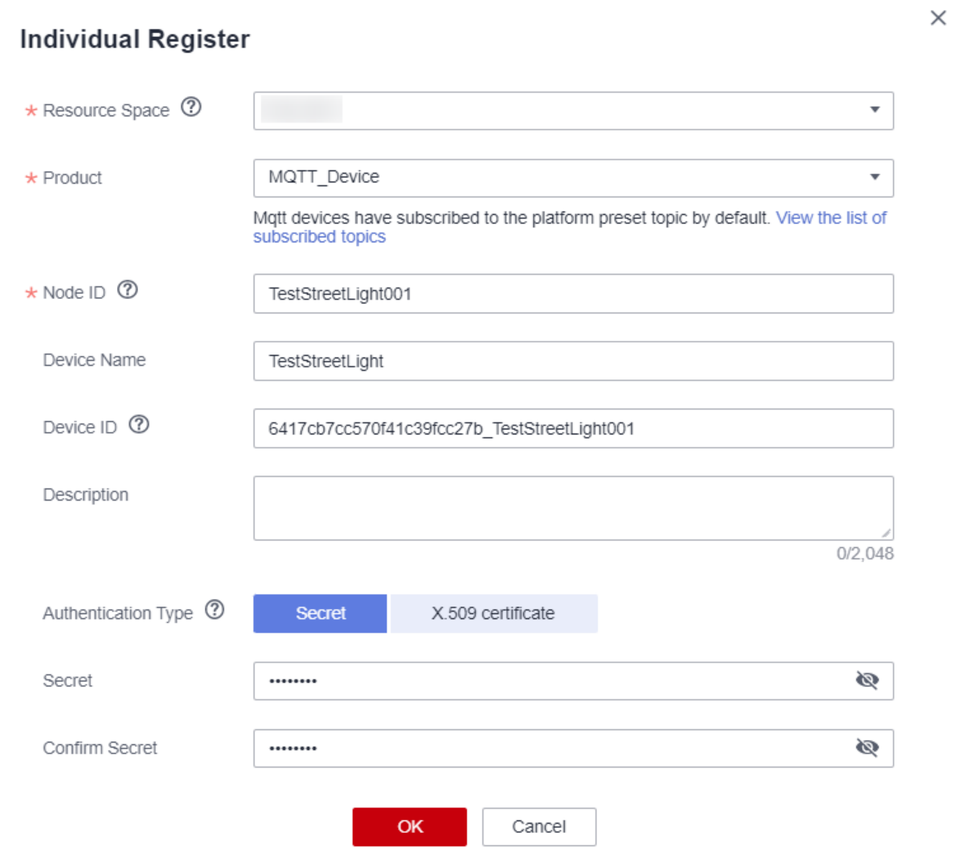
Profile_01.zip (1.09KB) | Select File

OK | Cancel

A product model describes product details and service capabilities. You can define a product model using multiple methods. If you do not define a product model for a device, the platform only forwards the data reported by the dev

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

Figure 3-4 Parameters for registering a device



Individual Register

* Resource Space: [redacted]

* Product: MQTT_Device

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: TestStreetLight001

Device Name: TestStreetLight

Device ID: 6417cb7cc570f41c39fcc27b_TestStreetLight001

Description:

0/2,048

Authentication Type: Secret | X.509 certificate

Secret: ••••••••

Confirm Secret: ••••••••

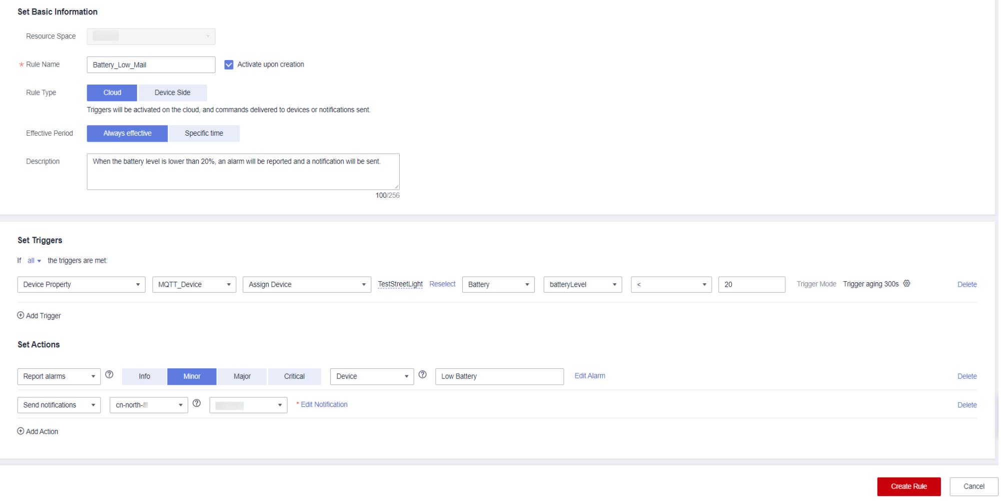
Table 3-3 Parameters for registering a device

Parameter	Description
Product	Select the product created in 3.

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

8. In the navigation pane, choose **Rules** > **Device Linkage**, and click **Create Rule** in the upper right corner. (Before creating a rule, select the resource space to which the rule will belong.)
9. 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 3-5 Creating a linkage rule



Set Basic Information

Resource Space: [redacted]

* Rule Name: Battery_Low_Mail ☑ Activate upon creation

Rule Type: Cloud | Device Side

Triggers will be activated on the cloud, and commands delivered to devices or notifications sent.

Effective Period: Always effective | Specific time

Description: When the battery level is lower than 20%, an alarm will be reported and a notification will be sent.

100/256

Set Triggers

If all the triggers are met:

Device Property | MQTT_Device | Assign Device | TestStreetLight Reselect | Battery | batteryLevel | < | 20 | Trigger Mode Trigger aging 300s | Delete

⊕ Add Trigger

Set Actions

Report alarms | Info | Minor | Major | Critical | Device | Low Battery | Edit Alarm | Delete

Send notifications | cn-north-[redacted] | [redacted] | * Edit Notification | Delete

⊕ Add Action

Table 3-4 Parameters for creating a linkage rule

Parameter	Description
Rule Name	Enter a name, such as **Battery_Low_Mail** or **Battery_Low_SMS**.
Activate upon creation	Select **Activate upon creation**.
Effective Period	Select **Always effective**.

Parameter	Description
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	1. Click **Add Trigger**. 2. Select **Device Property**. 3. Select the product added in 3, select **Assign Device**, and then select the device added in 7. 4. Select **Battery** for **Select service**, **batteryLevel** for **Select property**, **<** as the operation, and enter **20**. Click **Trigger Mode**. In the dialog box displayed, set **Trigger Strategy** to **Repetition suppression** and **Data Validity Period (s)** to **3600**, and click **OK**.
Set Actions	Add an alarm. 1. Click **Add Action**. 2. Select **Report alarms**. 3. Set the alarm severity to **Minor**, alarm isolation level to **Device**, **Alarm Name** to **Low battery level**, and **Description** to **The battery level is lower than 20%. Check and replace the battery in time.** Click **OK**. Add a notification. 1. Click **Add Action**. 2. Select **Send notifications**. 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.) 4. Select the topic created when configuring SMN for **Topic Name**. - If **Protocol** corresponding to the topic is **Email**, set **Message Title** to an email title, for example, **[Huawei IoT Platform] Low Battery Warning**, and set **Message Content** to information similar to **You have a device with less than 20% charge, please log in to the Huawei IoT Platform for details.** - If **Protocol** corresponding to the topic is **SMS**, left **Message Title** unspecified, and set **Message Content** to information similar to **You have a device with less than 20% charge, please log in to the Huawei IoT Platform for details.**

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 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.

3.2 Automatic Device Shutdown Upon High Temperature

Scenarios

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.

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.

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

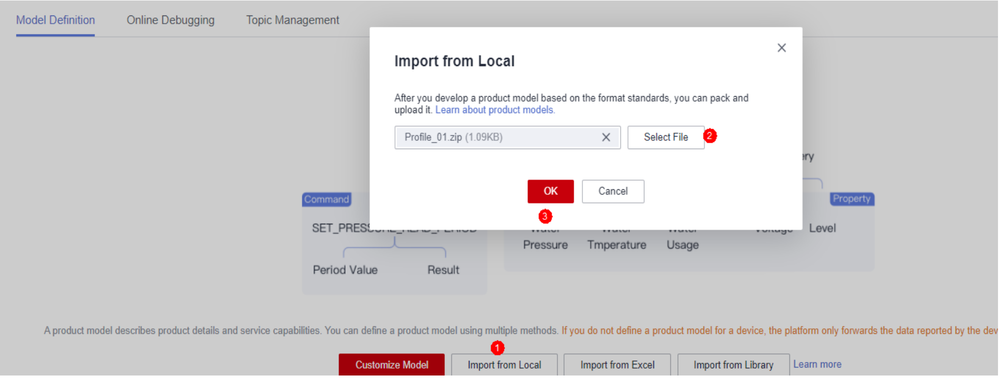
   Note: The product model and device used in this document are only examples. You can use your own product model and device.
3. Click **Create Product** in the upper right corner to create a product using MQTT. Set the parameters and click **Create**.

Table 3-5 Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, **MQTT_Device**.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
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 **Model Definition** tab page, click **Import from Local**. In the dialog box displayed, load the local product model and click **OK**.

Figure 3-6 Uploading a model file



Import from Local

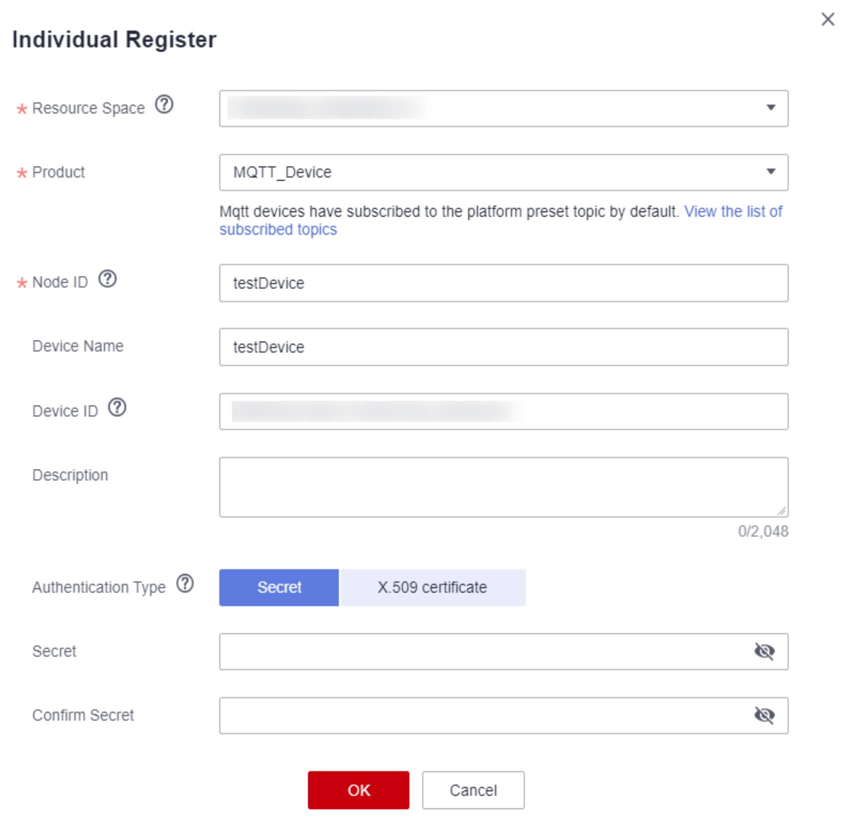
After you develop a product model based on the format standards, you can pack and upload it. Learn about product models.

Profile_01.zip (1.09KB)

A product model describes product details and service capabilities. You can define a product model using multiple methods. If you do not define a product model for a device, the platform only forwards the data reported by the dev

6. In the navigation pane, choose **Devices** > **All Devices**. On the page displayed, click **Individual Register** in the upper right corner. On the page displayed, set device registration parameters and click **OK**.

Figure 3-7 Registering a device



Individual Register

* Resource Space
* Product: MQTT_Device

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: testDevice

Device Name: testDevice

Device ID

Description

0/2,048

Authentication Type: Secret | X.509 certificate

Secret

Confirm Secret

Table 3-6 Parameters for registering a device

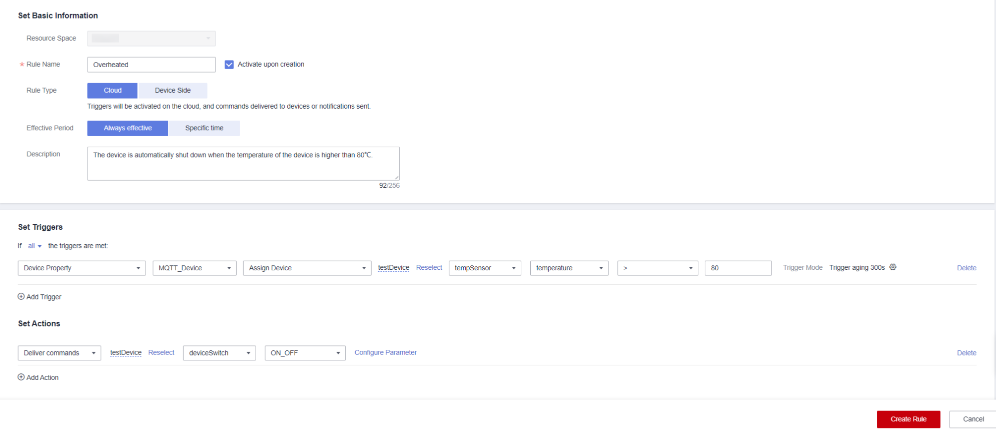
Parameter	Description
Product	Select the product created in 3.
Node ID	Set this parameter to the IMEI, MAC address, or serial number of the device. If the device is not a physical one, set this parameter to a custom character string that contains letters and digits.
Device Name	Customize the value.
Authentication Type	Select **Secret**.
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** in the upper right corner. (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 3-8 Creating a linkage rule



Set Basic Information

Resource Space [redacted]

* Rule Name: Overheated ☑ Activate upon creation

Rule Type: Cloud (selected) | Device Side

Triggers will be activated on the cloud, and commands delivered to devices or notifications sent.

Effective Period: Always effective (selected) | Specific time

Description: The device is automatically shut down when the temperature of the device is higher than 80℃.

92/256

Set Triggers

If all the triggers are met:

Device Property | MQTT_Device | Assign Device | testDevice Reselect | tempSensor | temperature | > | 80 | Trigger Mode Trigger aging 300s | Delete

⊕ Add Trigger

Set Actions

Deliver commands | testDevice Reselect | deviceSwitch | ON_OFF | Configure Parameter | Delete

⊕ Add Action

Table 3-7 Parameters for creating a linkage rule

Parameter	Description
Rule Name	Specify the name of the rule to be created, for example, **Overheated**.
Activate upon creation	Select **Activate upon creation**.
Effective Period	Select **Always effective**.
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	1. Click **Add Trigger**. 2. Select **Device Property**. 3. Select the product added in **3**, select **Assign Device**, and then select the device added in **6**. 4. Select **tempSensor** for **Select service**, **temperature** for **Select property**, **>** as the operation, and enter **80**. Click **Trigger Mode**. In the dialog box displayed, set **Trigger Strategy** to **Repetition suppression** and **Data Validity Period (s)** to **300**, and click **OK**.

Parameter	Description
Set Actions	1. Click **Add Action**. 2. Select **Deliver commands**, and select the device created in 6. 3. Select **deviceSwitch** for **Select service**, and **ON_OFF** for **Select command**. Click **Configure Parameter**. In the dialog box displayed, set **power** to **OFF**, and click **OK**.

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 {***deviceId***} with the actual device ID) and report data whose temperature is greater than 80. For details, see **Developing an MQTT-based 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.

3.3 Automatically Opening the Window upon High Gas Concentration

Scenarios

IoTDA can instruct a wireless window opener to open the window through a device linkage rule. The scenario is as follows: 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.

Creating a Gas Monitoring Product

Step 1 Visit the **IoTDA** product page and click **Access Console**.

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

Step 3 Click **Create Product** in the upper right corner to create a gas monitoring product, set the parameters, and click **OK**.

Table 3-8 Parameters for creating a product

Basic Information	
Product Name	Enter a value, for example, **gasdevice**.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Manufacturer	Customize the value.
Industry	Customize the values.
Device Type	

Step 4 On the **Model Definition** tab page, click **Custom 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 3-9 Parameters of the gas concentration monitoring product model

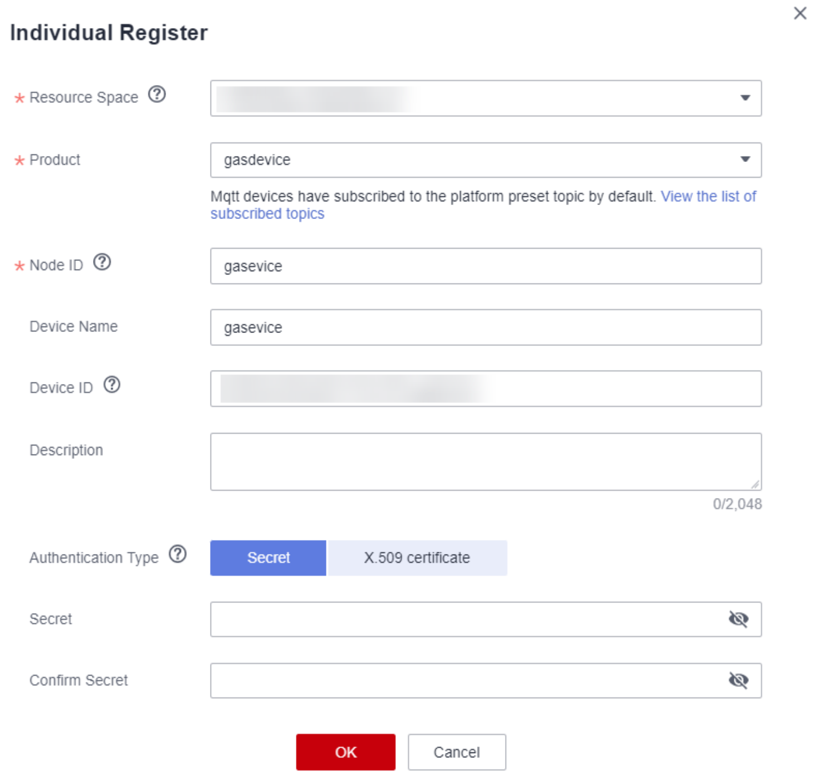
Service ID	Type	Description
gaslevel	Properties	Property Name: gaslevel Data Type: int Access Permissions: Read Value range: 0–100
windowswitch	Commands	Command Name: switch Parameter Name: switch Data Type: enum Enumerated Values: on,off

----End

Registering a Device

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

Figure 3-9 Registering a gas concentration monitoring device



Individual Register

* Resource Space: [redacted]

* Product: gasdevice

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: gasevice

Device Name: gasevice

Device ID: [redacted]

Description:

0/2,048

Authentication Type: Secret | X.509 certificate

Secret:

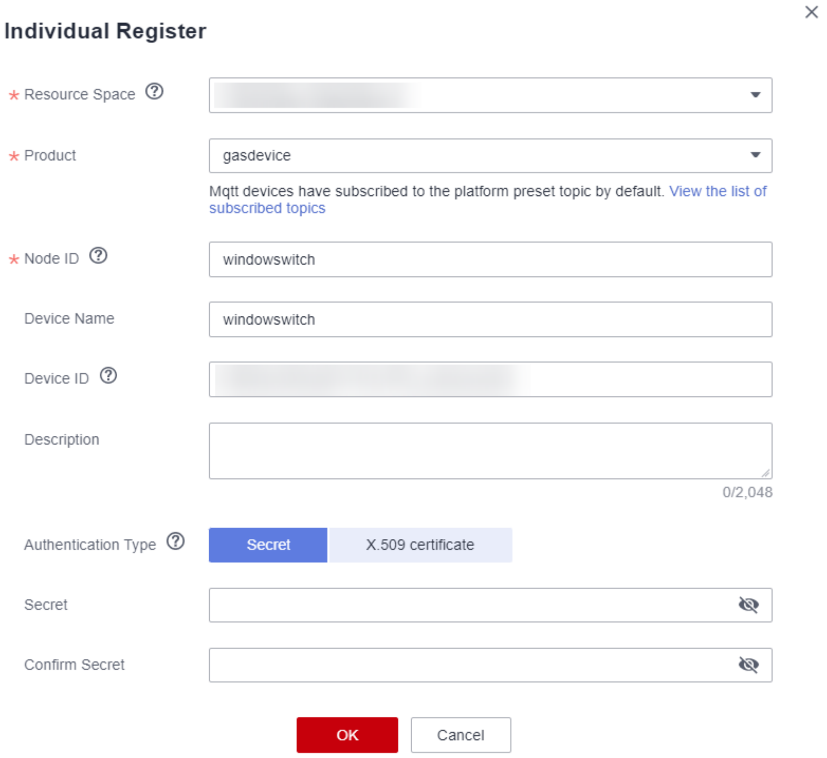
Confirm Secret:

Table 3-10 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 **Secret**.
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 page displayed, click **Individual Register** in the upper right corner. On the page displayed, set device registration parameters and click **OK**. Register a window opener device and record the device ID and secret.

Figure 3-10 Parameters for registering a window opener device



Individual Register

* Resource Space

* Product: gasdevice

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: windowswitch

Device Name: windowswitch

Device ID: [redacted]

Description: 0/2,048

Authentication Type: Secret (selected) | X.509 certificate

Secret

Confirm Secret

Table 3-11 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 **Secret**.
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

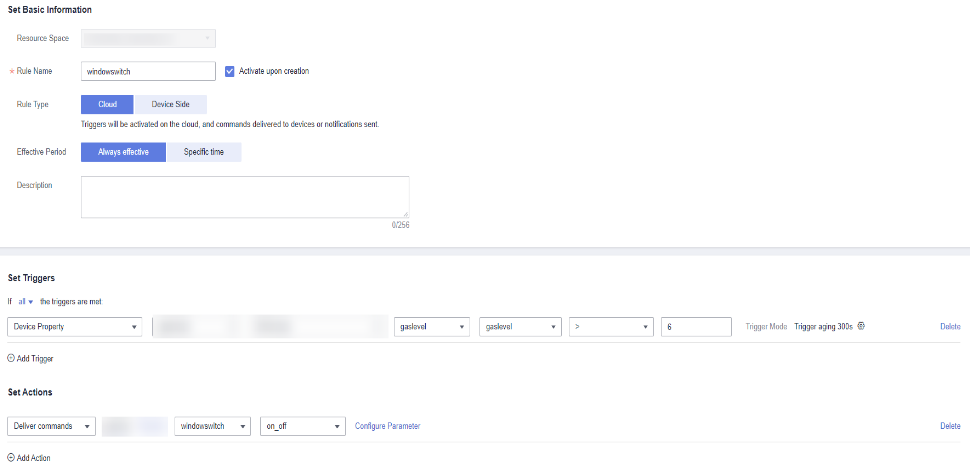
----End

Configuring a Device Linkage Rule

Step 1 In the navigation pane, choose **Rules** > **Device Linkage**, and click **Create Rule** in the upper right corner. (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 3-11 Creating a linkage rule



Set Basic Information

Resource Space

* Rule Name: windowswitch ☑ Activate upon creation

Rule Type: Cloud | Device Side

Triggers will be activated on the cloud, and commands delivered to devices or notifications sent.

Effective Period: Always effective | Specific time

Description: 0/256

Set Triggers

If all ▾ the triggers are met:

Device Property ▾ | gaslevel ▾ | gaslevel ▾ | > ▾ | 6 | Trigger Mode Trigger aging 300s | Delete

⊕ Add Trigger

Set Actions

Deliver commands ▾ | windowswitch ▾ | on_off ▾ | Configure Parameter | Delete

⊕ Add Action

Table 3-12 Parameters for creating a linkage rule

Parameter	Description
Rule Name	Specify the name of the rule to create, for example, **windowswitch**.
Activate upon creation	Select **Activate upon creation**.
Effective Period	Select **Always effective**.
Description	Enter a description of the rule, for example, "Automatically opens the window when the gas concentration is higher than 6".
Set Triggers	1. Click **Add Trigger**. 2. Select **Device Property**. 3. Select the product added in **3**, select **Assign Device**, and then select the device added in **1**. 4. Select **gaslevel** for **Select service**, **gaslevel** for **Select property**, **>** as the operation, and enter **6**. Click **Trigger Mode**. In the dialog box displayed, set **Trigger Strategy** to **Repetition suppression** and **Data Validity Period (s)** to **300**, and click **OK**.

Parameter	Description
Set Actions	1. Click **Add Action**. 2. Select **Deliver Commands**, and select the device created in 2. 3. Select **windowswitch** for **Select service**, and **on_off** for **Select command**. Click **Configure Parameter**. In the dialog box displayed, set **switch** to **on**, click **OK**.

----End

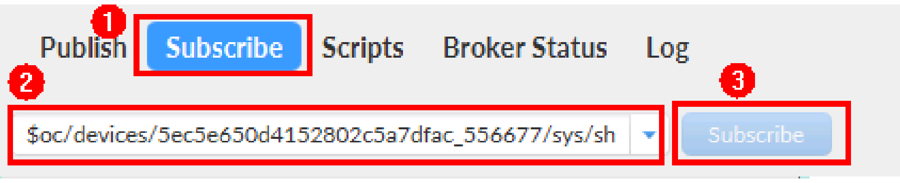
Verifying the Configurations

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 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. (The value must be the same as the device ID obtained in **2**.)
   c. Click **Subscribe** to deliver the subscription.

Figure 3-12 Creating an MQTT subscription



$oc/devices/5ec5e650d4152802c5a7dfac_556677/sys/sh

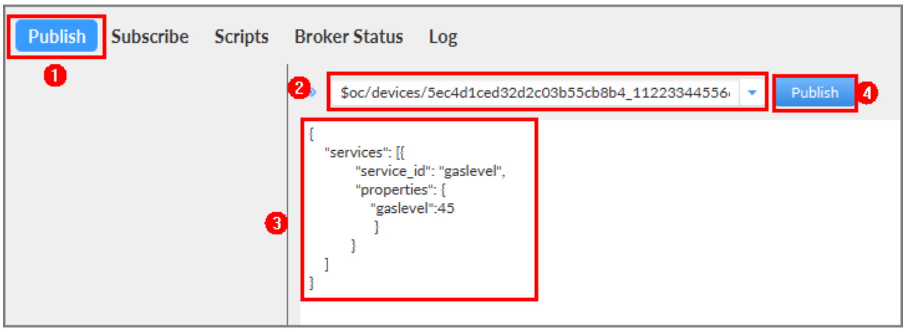
3. Switch to MQTT.fx that simulates the gas detector to report properties.
   a. Click the **Publish** tab.
   b. Enter **topic $oc/devices/{device_id}/sys/properties/report** for property reporting. ({deviceId} must be the same as the device ID obtained in **1**.)
   c. Report the property **gaslevel** with a value greater than 6.

   Example:

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

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

Figure 3-13 MQTT property reporting

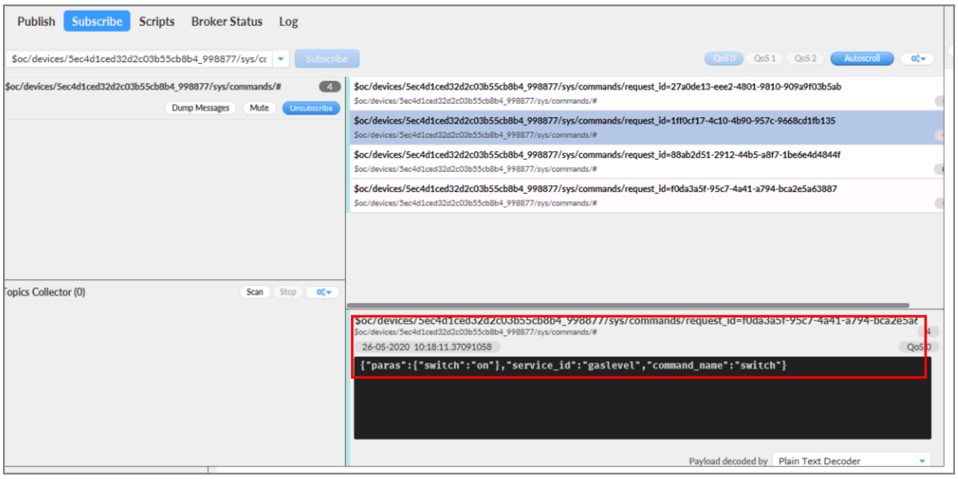


```
$oc/devices/5ec4d1ced32d2c03b55cb8b4_11223344556
```

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

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 3-14 Viewing delivered commands



```
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/cc
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
```

```
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/request_id=27a0de13-eee2-4801-9810-909a9f03b5ab
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/request_id=1ff0cf17-4c10-4b90-957c-9668cd1fb135
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/request_id=88ab2d51-2912-44b5-a8f7-1be6e4d4844f
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/request_id=f0da3a5f-95c7-4a41-a794-bca2e5a63887
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
```

```
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/request_id=f0da3a5f-95c7-4a41-a794-bca2e5ac
$oc/devices/5ec4d1ced32d2c03b55cb8b4_998877/sys/commands/#
26-05-2020 10:18:11.37091058
{"paras":{"switch":"on"},"service_id":"gaslevel","command_name":"switch"}
```

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.

3.4 Monitoring Device Status Changes and Sending Notifications

Scenarios

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:

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 five minutes, an alarm is reported by IoTDA. When the gateway is back to online for one minute, the alarm is cleared and an email or SMS is sent to a specified address or mobile number.

Interconnection Process

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.

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 in this topic are only examples. You can use your own product model and device.

Step 1 Visit the **IoTDA** product page and click **Access Console**.

Step 2 Choose **Products** in the navigation pane and click **Create Product** in the upper right corner to create an MQTT product. Set the parameters and click **OK**.

Table 3-13 Parameters for creating a product

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

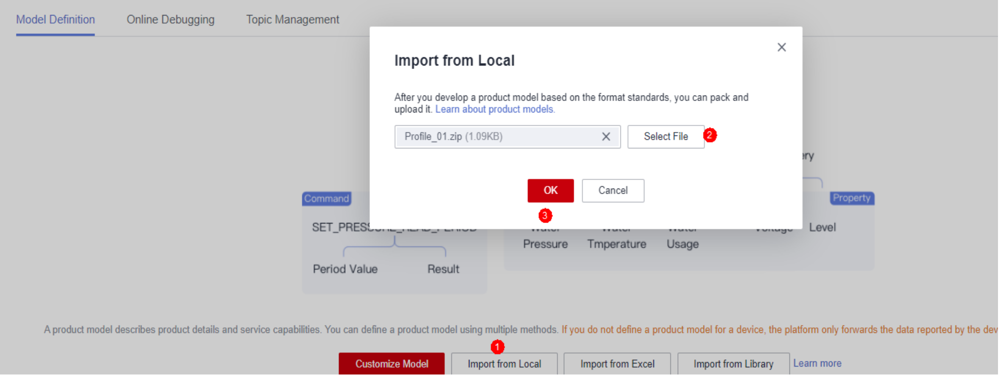
Device Type	

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 **Model Definition** tab page, click **Import from Local**. In the dialog box displayed, load the local product model and click **OK**.

Figure 3-15 Uploading a model file



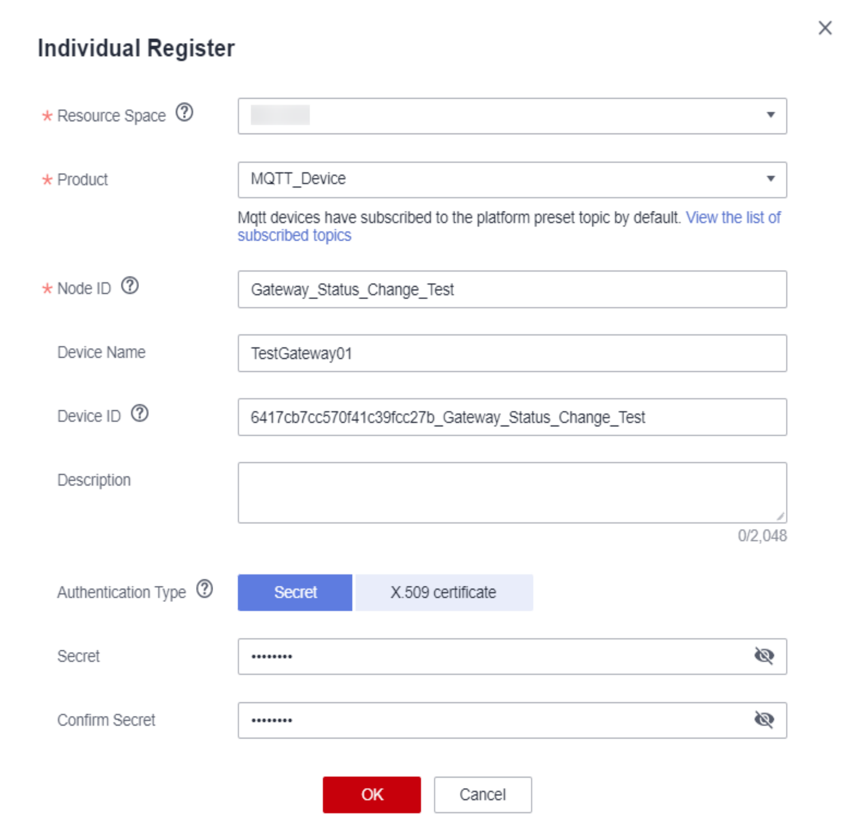
After you develop a product model based on the format standards, you can pack and upload it. Learn about product models.

A product model describes product details and service capabilities. You can define a product model using multiple methods. If you do not define a product model for a device, the platform only forwards the data reported by the dev

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

Copyright © Huawei Cloud Computing Technologies Co., Ltd.

Figure 3-16 Registering a device



Individual Register

* Resource Space

* Product: MQTT_Device

Mqtt devices have subscribed to the platform preset topic by default. View the list of subscribed topics

* Node ID: Gateway_Status_Change_Test

Device Name: TestGateway01

Device ID: 6417cb7cc570f41c39fcc27b_Gateway_Status_Change_Test

Description: 0/2,048

Authentication Type: Secret | X.509 certificate

Secret: ••••••••

Confirm Secret: ••••••••

Table 3-14 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 **Secret**.
Secret	Customize the secret used for device access. If the secret is left blank, the platform automatically generates a secret.

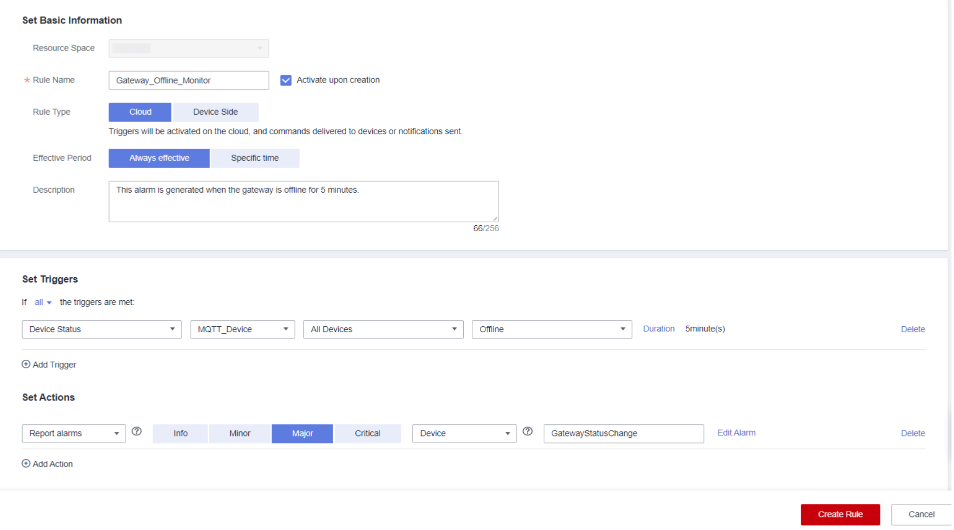
Step 7 In the navigation pane, choose **Rules** > **Device Linkage**, and click **Create Rule** in the upper right corner. (Before creating a rule, select the resource space to which the rule will belong.)

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 3-15 Parameters for creating a linkage rule

Parameter	Description
Rule Name	Customize a value, for example, **Gateway_Offline_Monitor**.
Activate upon creation	Select **Activate upon creation**.
Effective Period	Select **Always effective**.
Description	Describe the rule, for example, **This alarm is generated when the gateway is offline for 5 minutes.**.
Set Triggers	1. Click **Add Trigger**. 2. Select **Device Property**. 3. Select the **MQTT_Device** product created in **4**, select **All Devices**, and select **Offline** as the trigger status. 4. Set **Duration** to 5 minutes.
Set Actions	1. Click **Add Action**. 2. Select **Report alarms**. 3. Set the alarm severity to **Major**, alarm isolation level to **Device**, **Alarm Name** to **GatewayStatusChange**, and **Description** to **The gateway is offline. Check the issue and arrange personnel for maintenance in a timely manner.**, and click **OK**.

Figure 3-17 Configuring a linkage rule for reporting device going offline alarms



Set Basic Information

Resource Space

* Rule Name: Gateway_Offline_Monitor ☑ Activate upon creation

Rule Type: Cloud | Device Side

Triggers will be activated on the cloud, and commands delivered to devices or notifications sent.

Effective Period: Always effective | Specific time

Description: This alarm is generated when the gateway is offline for 5 minutes.

66/256

Set Triggers

If all the triggers are met:

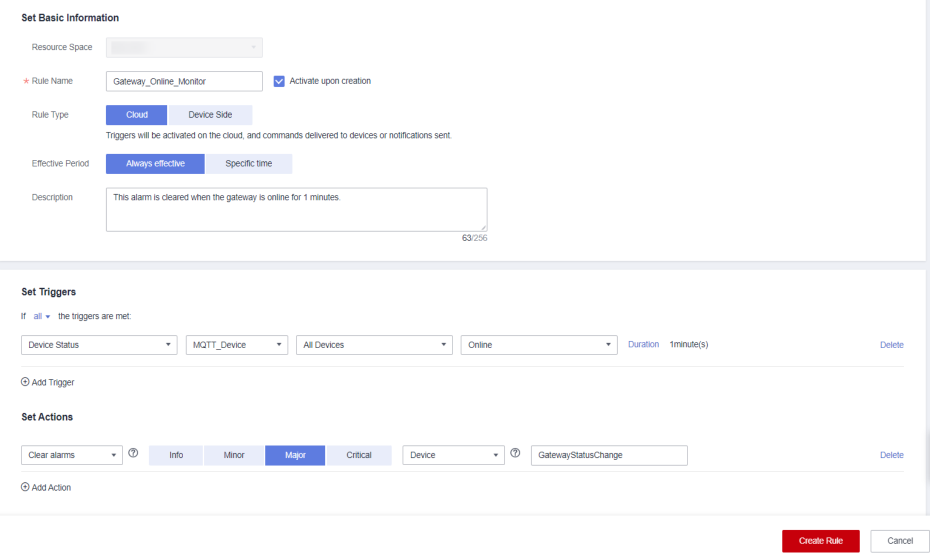
Device Status | MQTT_Device | All Devices | Offline | Duration 5minute(s) | Delete

Set Actions

Report alarms | Info | Minor | Major | Critical | Device | GatewayStatusChange | Edit Alarm | Delete

Step 9 Create a linkage rule for alarm clearance.

Figure 3-18 Creating a linkage rule for alarm clearance upon device going online



Set Basic Information

Resource Space

* Rule Name: Gateway_Online_Monitor ☑ Activate upon creation

Rule Type: **Cloud** | Device Side

Triggers will be activated on the cloud, and commands delivered to devices or notifications sent.

Effective Period: **Always effective** | Specific time

Description: This alarm is cleared when the gateway is online for 1 minutes.

63/256

Set Triggers

If all ▾ the triggers are met:

Device Status ▾ | MQTT_Device ▾ | All Devices ▾ | Online ▾ | Duration 1minute(s) | Delete

⊕ Add Trigger

Set Actions

Clear alarms ▾ ⓘ | Info | Minor | **Major** | Critical | Device ▾ ⓘ | GatewayStatusChange | Delete

⊕ Add Action

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.

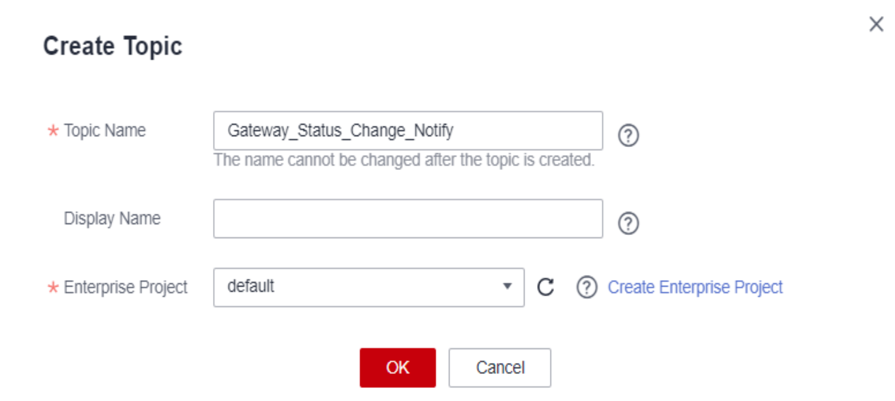
Step 1 Log in to Huawei Cloud and visit **SMN**.

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 the topic name, for example, **Gateway_Status_Change_Notify**, and click **OK**.

Figure 3-19 Creating a topic



Create Topic

* Topic Name: Gateway_Status_Change_Notify

The name cannot be changed after the topic is created.

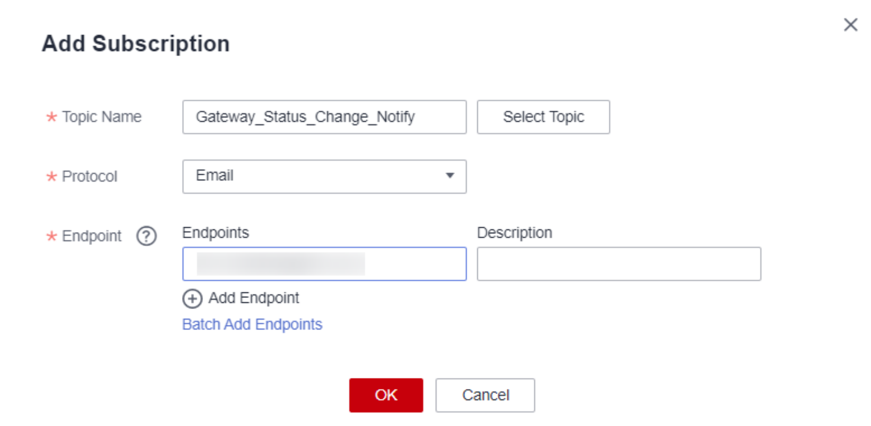
Display Name:

* Enterprise Project: default

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

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

Figure 3-20 Adding a subscription



Add Subscription

* Topic Name: Gateway_Status_Change_Notify

* Protocol: Email

* Endpoint: Endpoints [redacted] Description

Table 3-16 Parameters for adding a subscription

Parameter	Description
Topic Name	Select the topic created in **Step 4**.
Protocol	● To send an email notification, select **Email**. ● To send an SMS notification, select **SMS**.
Endpoint	● If **Protocol** is set to **Email**, enter the email address for receiving notifications. ● If **Protocol** is set to **SMS**, enter the mobile number for receiving notifications. To add multiple endpoints, place one endpoint in a line. A maximum of 10 lines can be entered.

----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.

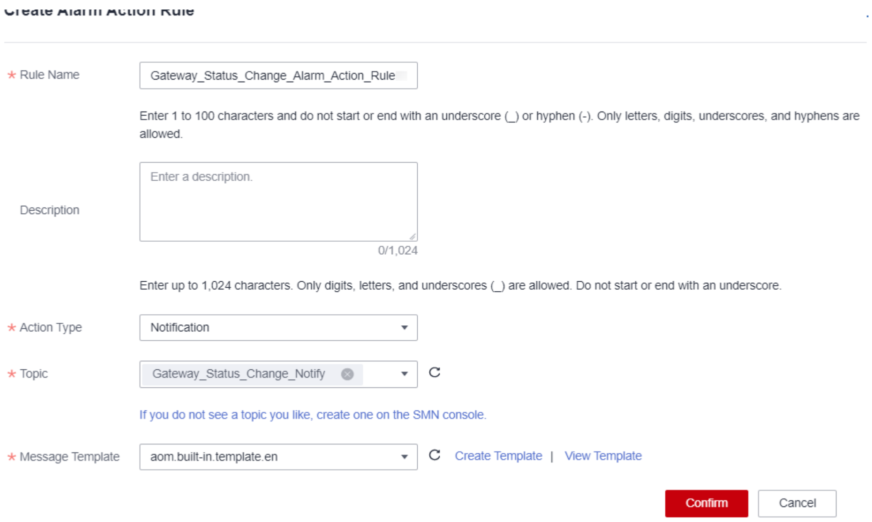
Step 1 Log in to Huawei Cloud and visit **AOM**.

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, **Gateway_Status_Change_Alarm_Action_Rule**, select the **Gateway_Status_Change_Notify** topic created in **Configuring SMN**, and click **Confirm**.

Figure 3-21 Configuring an alarm action rule



Create Alarm Action Rule

* Rule Name: Gateway_Status_Change_Alarm_Action_Rule

Enter 1 to 100 characters and do not start or end with an underscore (_) or hyphen (-). Only letters, digits, underscores, and hyphens are allowed.

Description: Enter a description.

0/1,024

Enter up to 1,024 characters. Only digits, letters, and underscores (_) are allowed. Do not start or end with an underscore.

* Action Type: Notification

* Topic: Gateway_Status_Change_Notify

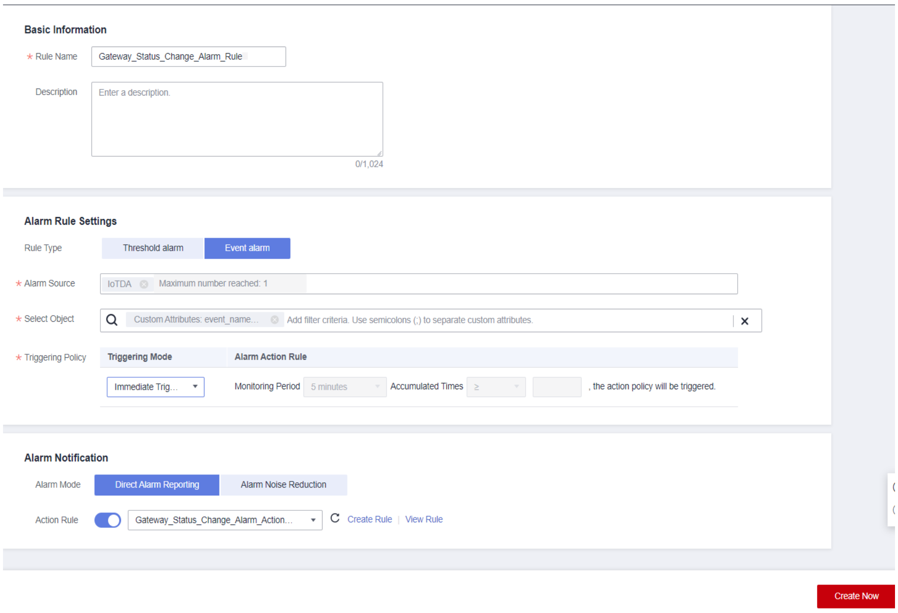
If you do not see a topic you like, create one on the SMN console.

* Message Template: aom.built-in.template.en

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 **Event alarm** for **Rule Type**, enter **IoTDA** in **Alarm Source**, set **Select Object** to **event_name=GatewayStatusChange** (GatewayStatusChange is the alarm name), and set **Triggering Policy** to **Immediate Triggering**, set **Alarm Mode** to **Direct Alarm Reporting**, set **Action Rule** to the action rule created in step **4**, and click **Create Now** in the lower right corner.

Figure 3-22 Configuring an alarm rule



Basic Information

* Rule Name: Gateway_Status_Change_Alarm_Rule

Description: Enter a description.

0/1,024

Alarm Rule Settings

Rule Type: Threshold alarm | Event alarm

* Alarm Source: IoTDA Maximum number reached: 1

* Select Object: Custom Attributes: event_name... Add filter criteria. Use semicolons (;) to separate custom attributes.

* Triggering Policy

Triggering Mode	Alarm Action Rule
Immediate Trig...	Monitoring Period 5 minutes Accumulated Times ≥ , the action policy will be triggered.

Alarm Notification

Alarm Mode: Direct Alarm Reporting | Alarm Noise Reduction

Action Rule: Gateway_Status_Change_Alarm_Action... Create Rule | View Rule

Create Now

----End

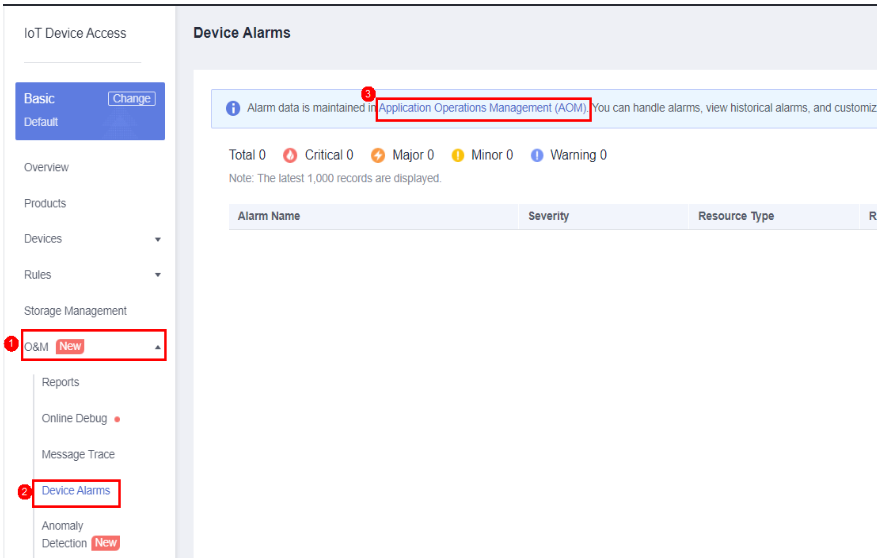
Verifying Configurations

- 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 Smart Street Light Online**.

Expected result:

1. After the device is offline for five 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 3-23 Accessing AOM



Alarm data is maintained in Application Operations Management (AOM). You can handle alarms, view historical alarms, and customiz

Total 0 Critical 0 Major 0 Minor 0 Warning 0

Note: The latest 1,000 records are displayed.

Alarm Name	Severity	Resource Type	R

- 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 one 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.