IoT Device Access

Service Overview

Issue 1.0

Date 2022-08-30

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions

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 Platform Overview..1

2 Quick Device Access..2

2.1 Quick Experience..2

2.2 Bidirectional Communications Between Virtual Devices and IoTDA..................................3

2.3 Sample Code for Bidirectional Communications...10

2.3.1 Quick Experience Based on Java Sample Code..10

2.3.2 Quick Experience Based on C Sample Code...21

3 Quick Application Access...31

1 Platform Overview

Understanding the Platform

IoT Device Access (IoTDA) is a basic service of the Huawei Cloud IoT platform. IoTDA provides functions such as device fleet access, bidirectional communications between devices and the cloud, batch device management, remote control and monitoring, Over-the-Air (OTA) upgrades, and device linkage rules. It can flexibly transfer device data to other Huawei Cloud services. Using IoTDA, you can quickly connect devices to the platform and integrate your applications. IoTDA integrates the functions of the original IoT Device Access and IoT Device Management services.

In short, IoTDA connects devices and applications to the platform for data interaction. You can manage applications and devices on the IoTDA console.

Experiencing IoTDA

This document describes how to connect devices to IoTDA and enable the communications between devices and IoTDA.

- **Quick Experience**

  This section helps you experience device data collection and command receiving using a Windows or Linux PC as a virtual device.

- **Bidirectional Communications Between Virtual Devices and IoTDA**

  This section helps you experience the basic functions and development process of the platform without downloading any software. You can use a virtual device on the platform to implement bidirectional communications. To explore more functions of the platform, see **Sample Code for Bidirectional Communications**.

- **Sample Code for Bidirectional Communications**

  This section is intended for developers. You can download sample code or develop code by yourself and run the code on your physical device or simulated device (Windows or Linux PC) to complete device access and experience advanced functions such as the rules and AMQP push.

2 Quick Device Access

2.1 Quick Experience

Scenarios

This section uses a virtual smart smoke detector as an example to describe how to connect a device to the platform, enable the device to report data, and deliver commands to the device.

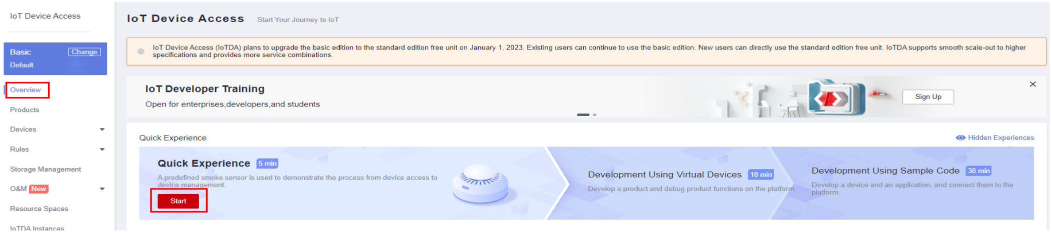
Prerequisites

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

Procedure

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

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



IoT Device Access (IoTDA) plans to upgrade the basic edition to the standard edition free unit on January 1, 2023. Existing users can continue to use the basic edition. New users can directly use the standard edition free unit. IoTDA supports smooth scale-out to higher specifications and provides more service combinations.

IoT Developer Training
Open for enterprises,developers,and students

Quick Experience 5 min
A predefined smoke sensor is used to demonstrate the process from device access to device management.

Development Using Virtual Devices 10 min
Develop a product and debug product functions on the platform.

Development Using Sample Code 30 min
Develop a device and an application, and connect them to the platform.

Step 3 Perform operations as prompted to complete the experience.

----End

Advanced Experience

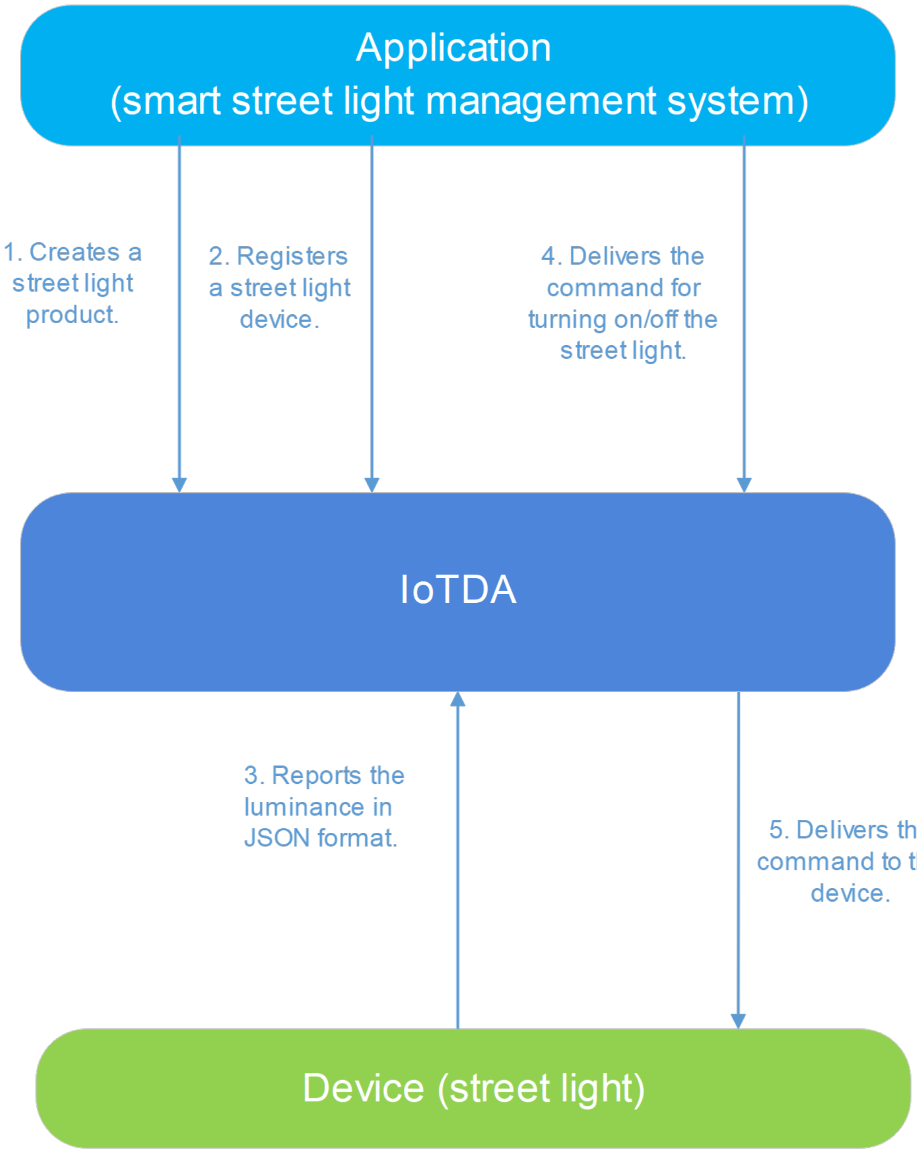
By following the instructions in this section, you may have understood how to connect a device to the platform and related concepts.

To better experience IoTDA, customize a product and use a virtual device and simulated application to experience the functions and development process of the platform. For details, see **Bidirectional Communications Between Virtual Devices and IoTDA**.

2.2 Bidirectional Communications Between Virtual Devices and IoTDA

Scenarios

This section uses a smart street light as an example to describe how to connect a device to the platform, enable the device to report luminance data, and deliver commands to the device.


Application
(smart street light management system)
1. Creates a street light product.
2. Registers a street light device.
4. Delivers the command for turning on/off the street light.
IoTDA
3. Reports the luminance in JSON format.
5. Delivers the command to the device.
Device (street light)

Prerequisites

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

Service Process

Based on the online debugging function of IoTDA, you can use a virtual device to experience device data reporting and platform command delivery for remote control.

The procedure is as follows:

Step 1 **Create an MQTT product.**

Step 2 **Develop a product model.** Define the luminance property reported by a street light to the platform and the command for remote control of the street light.

Step 3 **Register a virtual device** to experience data reporting.

Step 4 **Report data.** Report data in the device simulator area.

Step 5 **Deliver a command.** Deliver a command in the application simulator area.

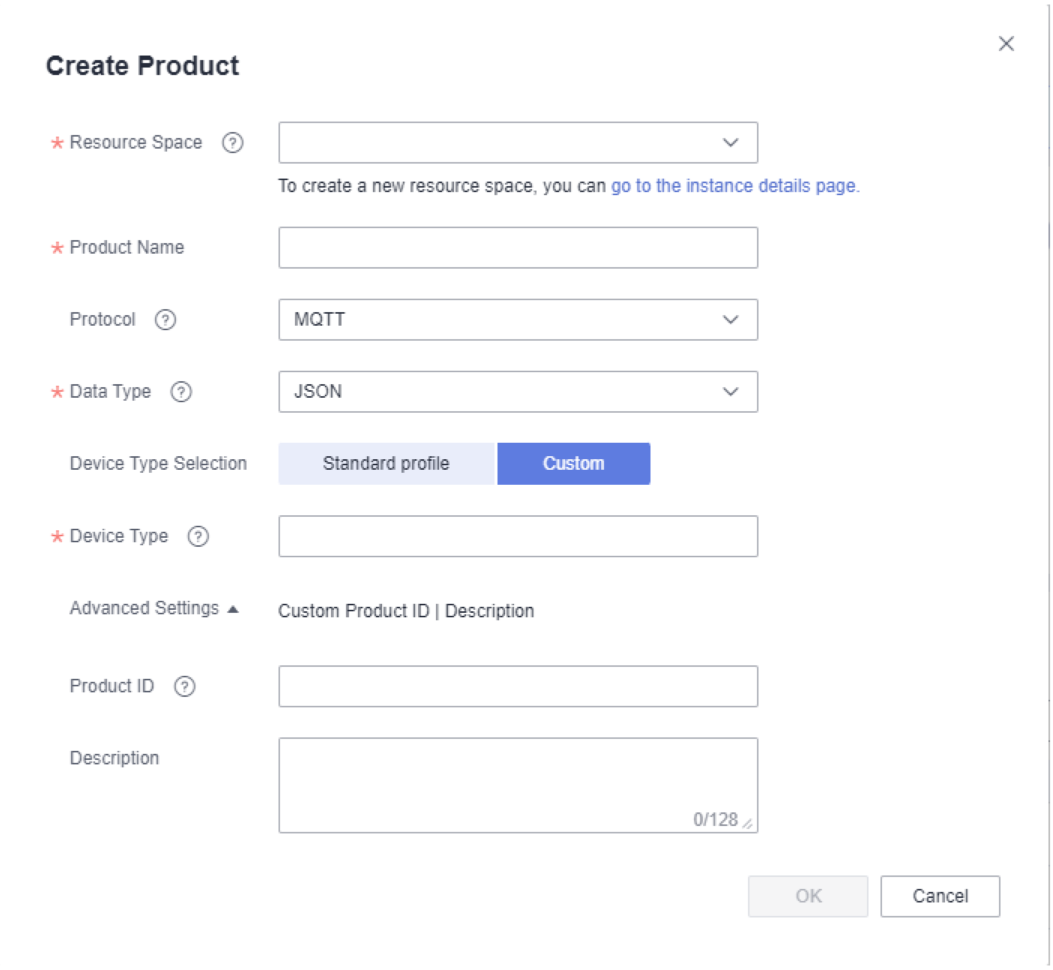
Creating a Product

A product is a collection of devices with the same capabilities or features.

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

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

Figure 2-1 Creating an MQTT 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

Device Type Selection: Standard profile | Custom

* Device Type

Advanced Settings ▲ Custom Product ID | Description

Product ID

Description 0/128

Basic Information

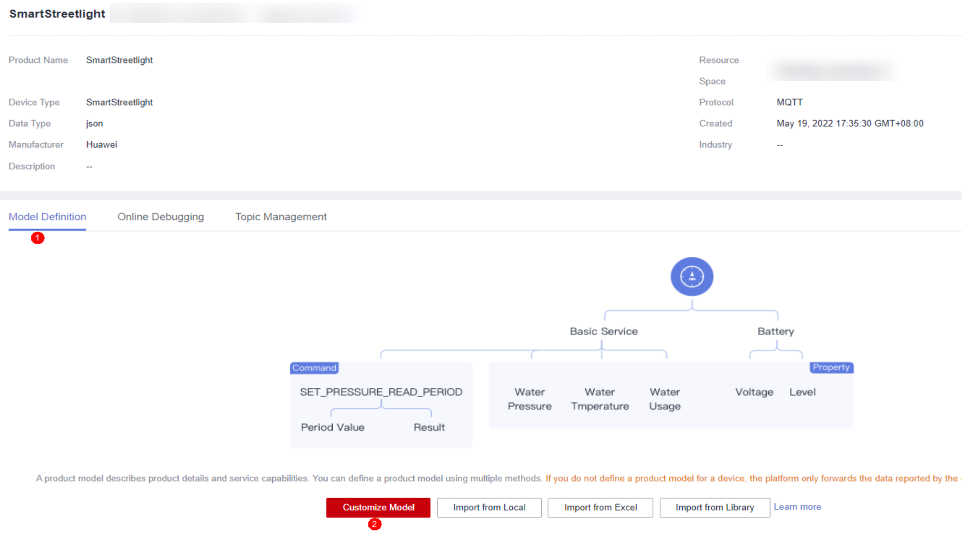
Resource Space	Select a resource space from the drop-down list box. If a **resource space** does not exist, create one.
Product Name	Customize a name, for example, **SmartStreetlight**. The value can contain up to 64 characters. Only letters, digits, and special characters (_?'#().,&%@!-) are allowed.
Protocol	Select **MQTT**.
Data Type	Select **JSON**.
Industry	Select **Default**.
Device Type	Enter **SmartStreetlight**.

----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 **Customize Model** to add services of the product.



SmartStreetlight

Product Name SmartStreetlight

Device Type SmartStreetlight

Data Type json

Manufacturer Huawei

Description --

Resource Space

Protocol MQTT

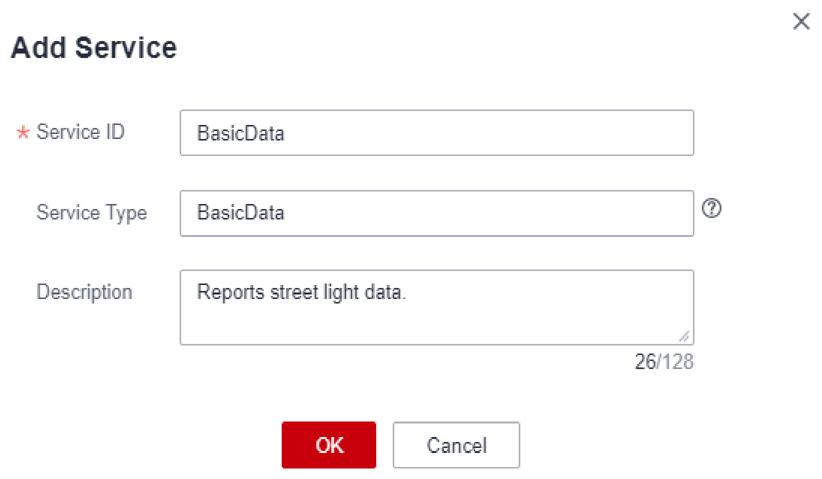
Created May 19, 2022 17:35:30 GMT+08:00

Industry --

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

Step 3 Add the **BasicData** service.

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



Add Service

* Service ID: BasicData

Service Type: BasicData

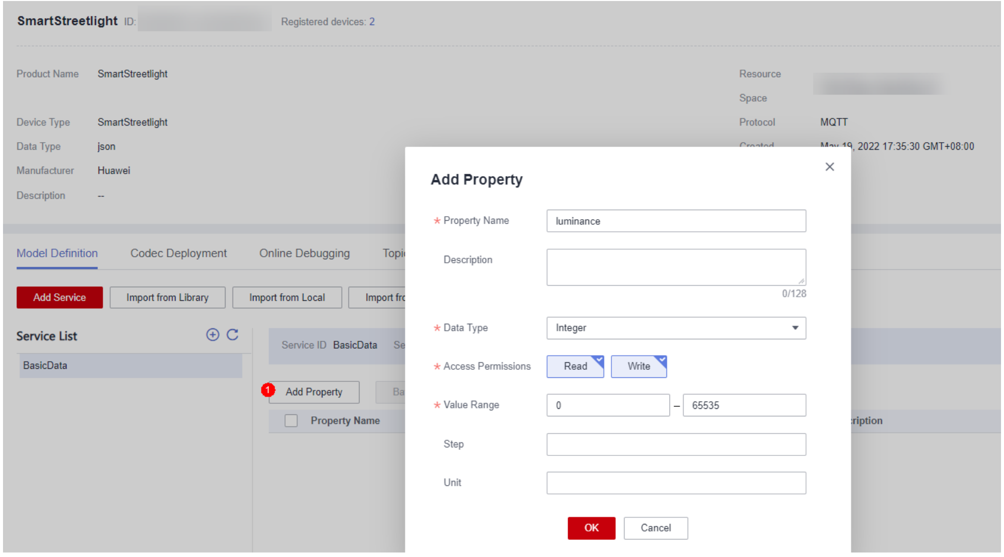
Description: Reports street light data.

26/128

OK Cancel

- **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. Click **Add Property**, enter related information, and click **OK**.



SmartStreetlight ID: Registered devices: 2

Product Name	SmartStreetlight	Resource Space	
Device Type	SmartStreetlight	Protocol	MQTT
Data Type	json	Created	May 19, 2022 17:35:30 GMT+08:00
Manufacturer	Huawei		
Description	--		

Add Property

* Property Name: luminance

Description:

0/128

* Data Type: Integer

* Access Permissions: Read Write

* Value Range: 0 – 65535

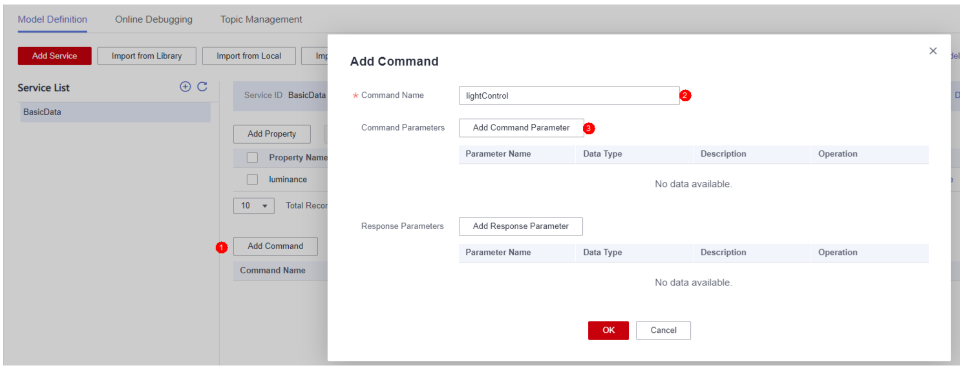
Step:

Unit:

OK Cancel

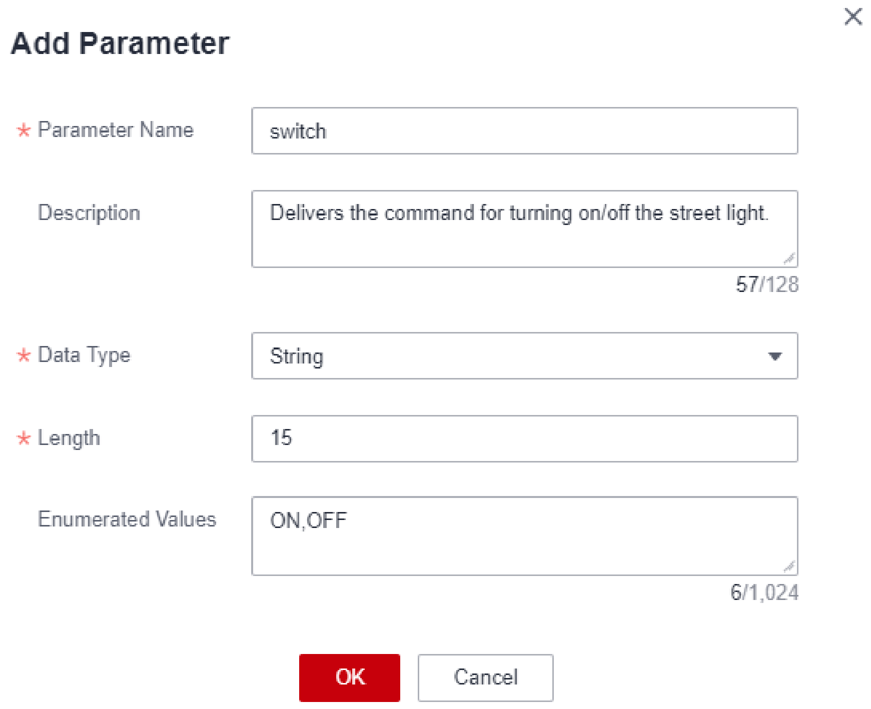
- **Property Name**: Enter **luminance**.
- **Data Type**: Select **Integer**.
- **Access Permissions**: Select **Read** and **Write**.
- **Value Range**: Set it to 0–65535.
- **Step**: Enter **0**.
- **Unit**: Leave it blank.

Step 4 Click **Add Command** and enter the command name **lightControl**.



On the **Add Command** page, click **Add Command Parameter**, enter related information, and click **OK**.

Figure 2-2 Adding the command parameter **switch**


Add Parameter

* Parameter Name: switch

Description: Delivers the command for turning on/off the street light. 57/128

* Data Type: String

* Length: 15

Enumerated Values: ON,OFF 6/1,024

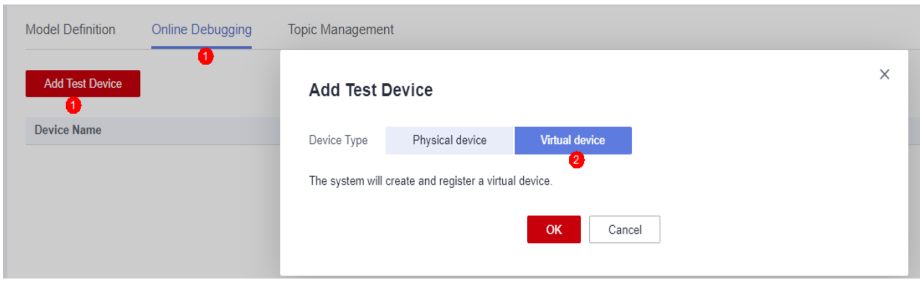
- **Parameter Name**: Enter **switch**.
- **Description**: Enter **Delivers the command for turning on/off the street light**.
- **Data Type**: Select **String**.
- **Length**: Enter **15**.
- **Enumerated Values**: Enter **ON,OFF**.

----End

Registering a Virtual Device

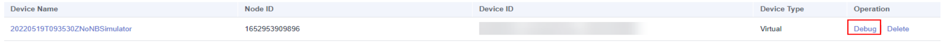
Step 1 Click the product created in **Creating a Product**. The product details page is displayed.

Step 2 On the **Online Debugging** tab page, click **Add Test Device**. In the displayed dialog box, select **Virtual device** and click **OK**.



The system will create and register a virtual device.

The virtual device name contains **Simulator**. Select the new virtual device and click **Debug** on the right. The debugging page is displayed, and the device status changes to online.

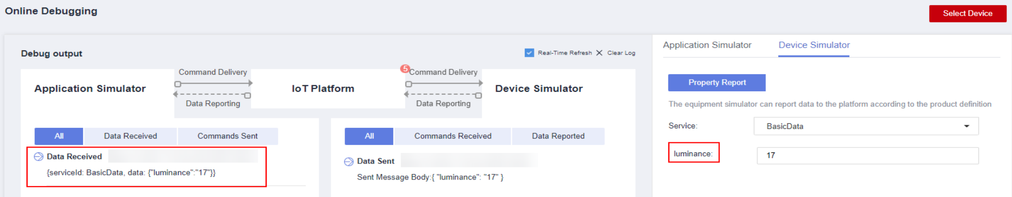


Device Name	Node ID	Device ID	Device Type	Operation
20220519T093530ZNoNBSimulator	1652953909896	[redacted]	Virtual	Debug Delete

----End

Reporting Data

In **Device Simulator**, enter a luminance value and click **Send**. View the reporting result in **Application Simulator**.



Data Received
{serviceId: BasicData, data: {"luminance":"17"}}

Data Sent
Sent Message Body:{ "luminance": "17" }

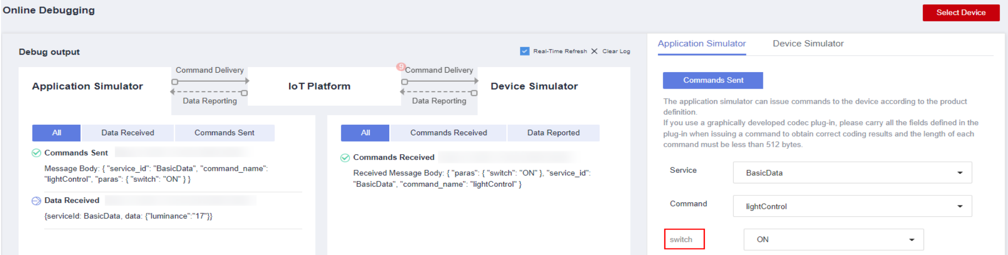
The equipment simulator can report data to the platform according to the product definition

Service: BasicData

luminance: 17

Delivering a Command

In **Application Simulator**, select the command, enter a parameter value, and deliver the command for remote control of the street light. View the received command in **Device Simulator**.



Commands Sent
Message Body: { "service_id": "BasicData", "command_name": "lightControl", "paras": { "switch": "ON" } }

Data Received
{serviceId: BasicData, data: {"luminance":"17"}}

Commands Received
Received Message Body: { "paras": { "switch": "ON" }, "service_id": "BasicData", "command_name": "lightControl" }

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: BasicData

Command: lightControl

switch: ON

Advanced Experience

By following the instructions in this section, you may have understood the basic functions and development process of the platform.

To better experience IoTDA, develop real applications and devices and connect them to the platform. For details, see **Sample Code for Bidirectional Communications**.

2.3 Sample Code for Bidirectional Communications

In this section, you can use sample code to connect a physical device or Window/ Linux PC to IoTDA, enable the device to report data to the platform and receive a control command delivered by the platform, and enable the application to receive data reported to the platform.

2.3.1 Quick Experience Based on Java Sample Code

Overview

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

Prerequisites

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

Uploading a Product Model

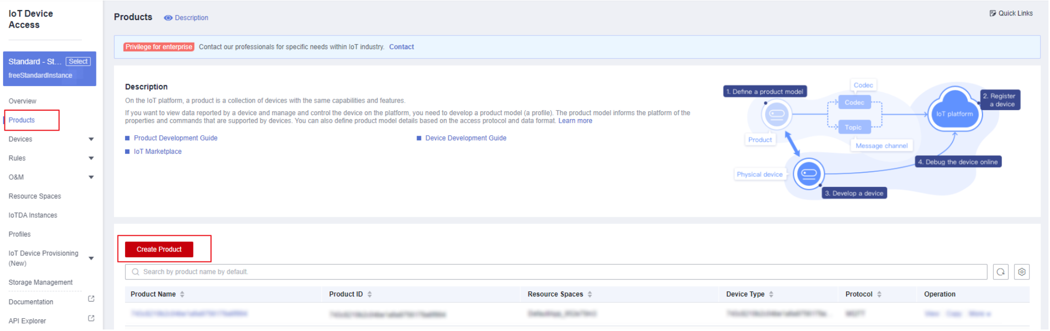
A product model is a JSON file that describes device capabilities. It defines basic device properties and message formats for data reporting and command delivery. Defining a product model is to construct an abstract model of a device in the platform to enable the platform to understand the device function.

Procedure

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

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

Figure 2-3 Creating a product



Contact our professionals for specific needs within IoT industry.

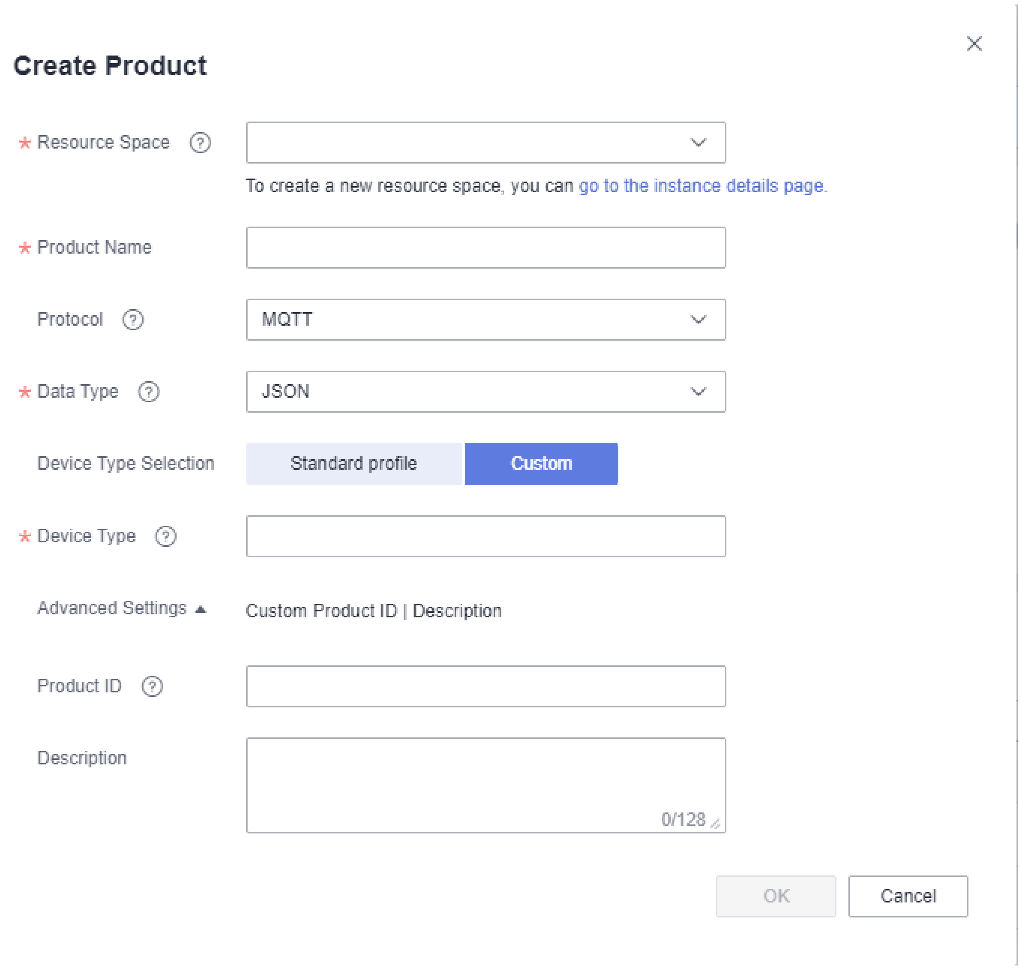
Description

On the IoT platform, a product is a collection of devices with the same capabilities and features.

If you want to view data reported by a device and manage and control the device on the platform, you need to develop a product model (a profile). The product model informs the platform of the properties and commands that are supported by devices. You can also define product model details based on the access protocol and data format. Learn more

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

Figure 2-4 Creating an MQTT product



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

Protocol: MQTT

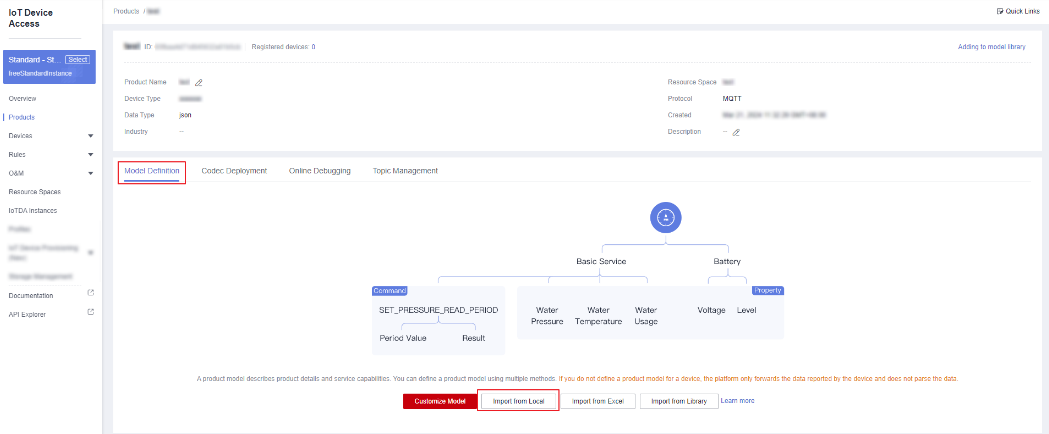
Data Type: JSON

Device Type Selection: Custom

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

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

Figure 2-5 Uploading an MQTT product model



ID: | Registered devices: 0

Product Name

Device Type

Data Type json

Industry --

Resource Space

Protocol MQTT

Created

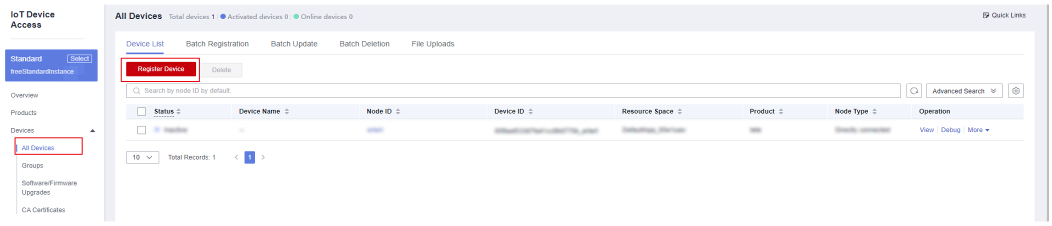
Description --

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 device and does not parse the data.

----End

Creating a Device

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

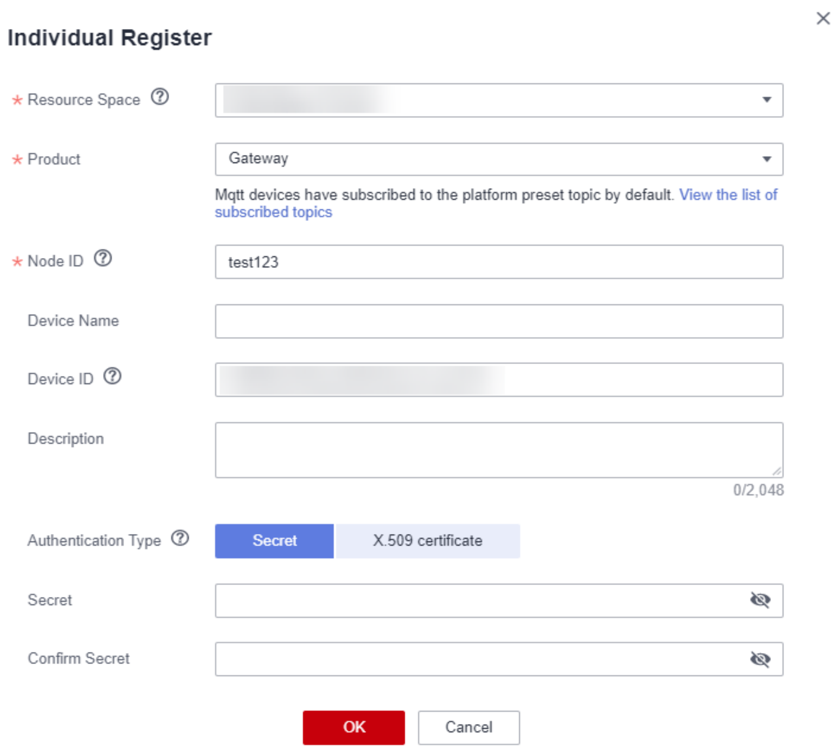


All Devices Total devices 1 | Activated devices 0 | Online devices 0

☐	Status	Device Name	Node ID	Device ID	Resource Space	Product	Node Type	Operation
☐		--						View | Debug | More

10 Total Records: 1 < 1 >

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



Individual Register

* Resource Space: [redacted]

* Product: Gateway

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

* Node ID: test123

Device Name:

Device ID: [redacted]

Description:

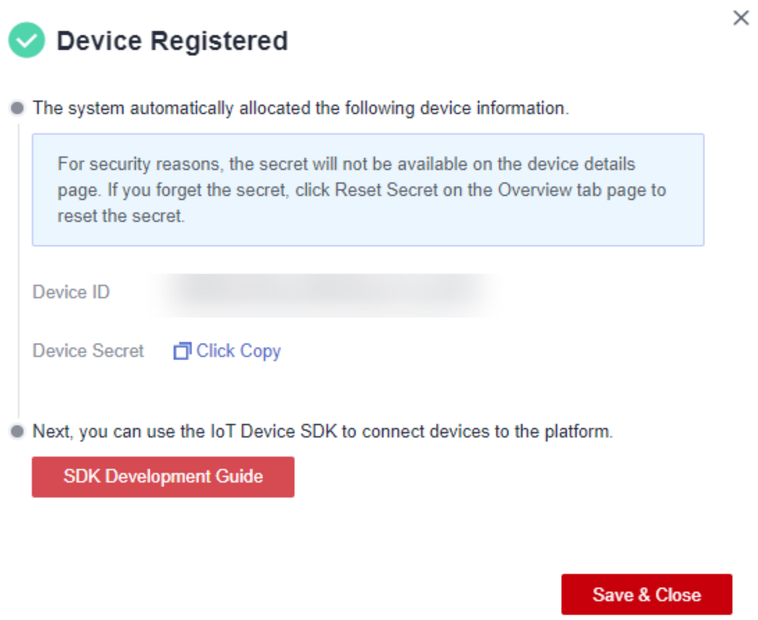
0/2,048

Authentication Type: Secret | X.509 certificate

Secret:

Confirm Secret:

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



Device Registered

- The system automatically allocated the following device information.

> 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.

Device ID: [redacted]

Device Secret: Click Copy

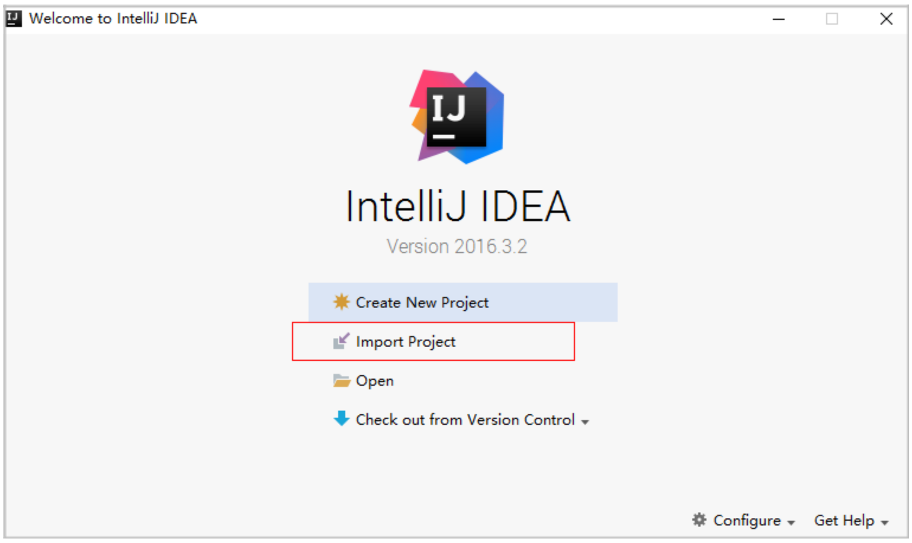
- Next, you can use the IoT Device SDK to connect devices to the platform.

----End

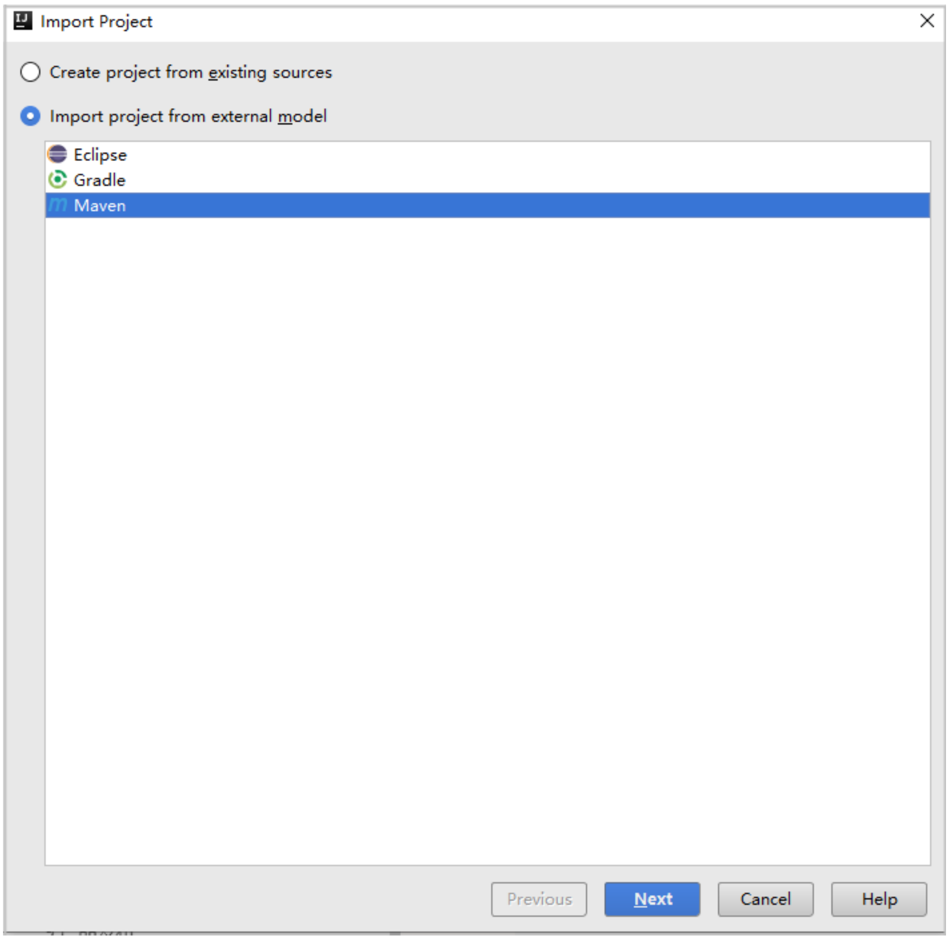
Importing Sample Code

Step 1 Download the **Java demo**.

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



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

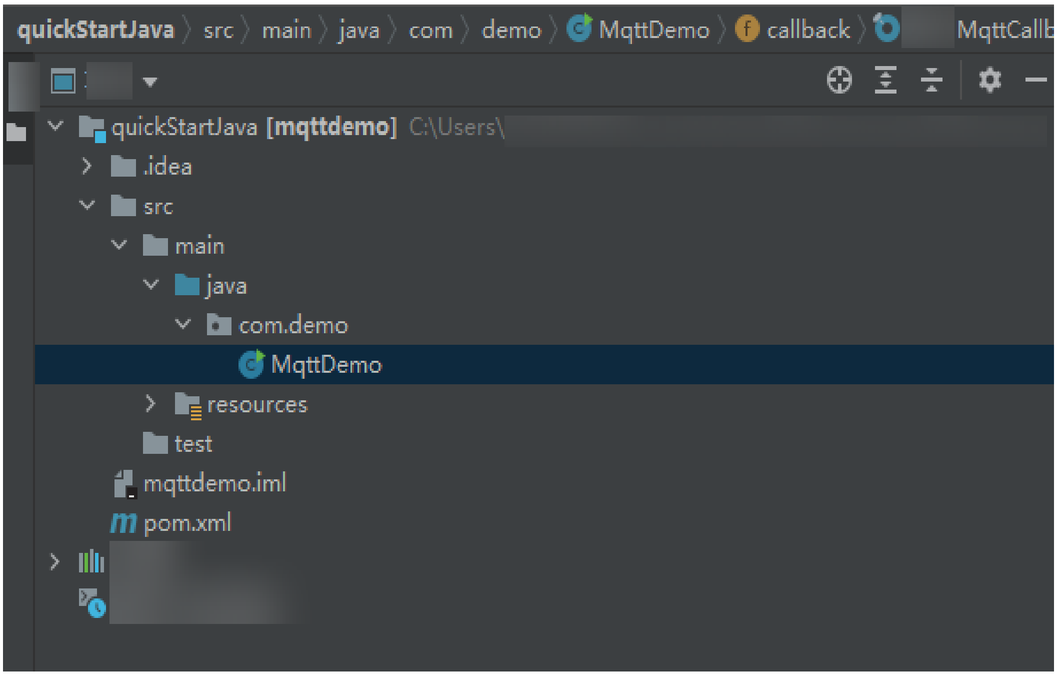


○ Create project from existing sources

● Import project from external model

- Eclipse
- Gradle
- Maven

Step 4 Import the sample code.



```
quickStartJava 〉 src 〉 main 〉 java 〉 com 〉 demo 〉 MqttDemo 〉 callback 〉 [redacted] MqttCallb

quickStartJava [mqttdemo] C:\Users\[redacted]
  .idea
  src
    main
      java
        com.demo
          MqttDemo
      resources
    test
  mqttdemo.iml
  pom.xml
[redacted]
```

----End

Establishing a Connection

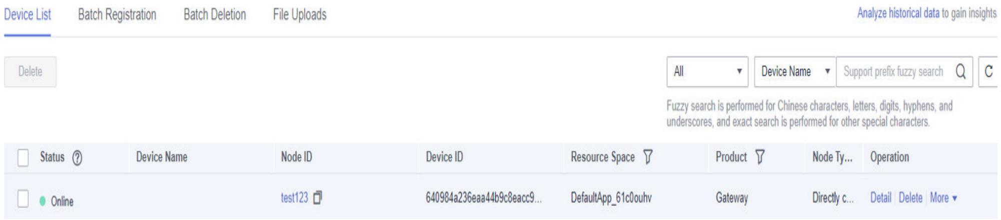
To connect a device or gateway to the platform, upload the device information to bind the device or gateway to the platform.

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

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

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

2. After the preceding information is modified, run the program. The device is displayed as online on the platform.



Fuzzy search is performed for Chinese characters, letters, digits, hyphens, and underscores, and exact search is performed for other special characters.

Status	Device Name	Node ID	Device ID	Resource Space	Product	Node Ty...	Operation
● Online		test123	640984a236eaa44b9c8eacc9...	DefaultApp_61c0ouhv	Gateway	Directly c...	Detail Delete More ▾

Reporting Properties

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

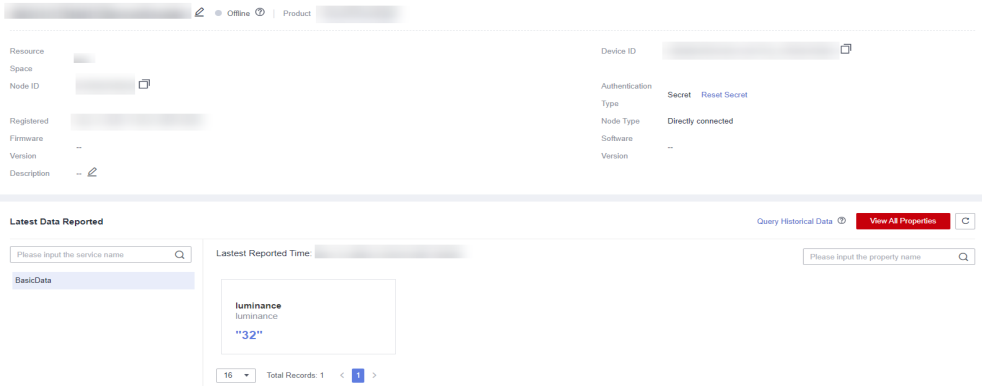
```
// Report JSON data. service_id must be the same as that defined in the product model.
String jsonMsg = "{\"services\":[{\"service_id\":\"BasicData\",\"properties\":{\"luminance\":32},\"eventTime
\":null}]}";
```

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

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

Viewing Reported Data

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



[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

Please input the service name

BasicData

Lastest Reported Time: [redacted]

Please input the property name

luminance
luminance
"32"

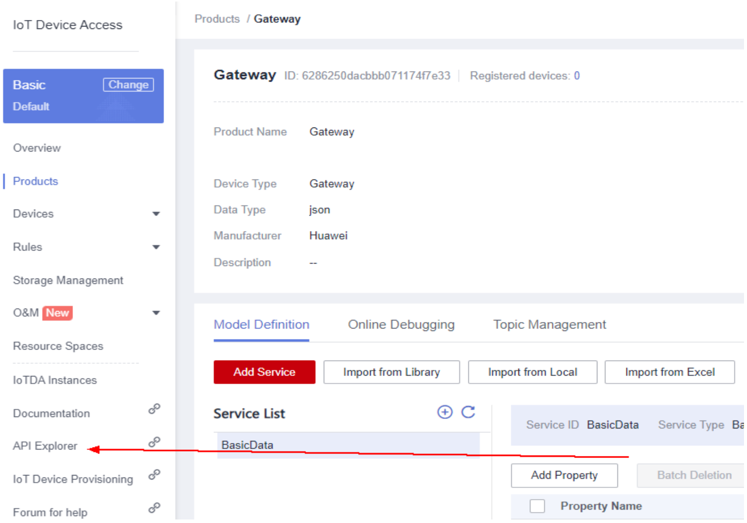
16 Total Records: 1 1

NOTE

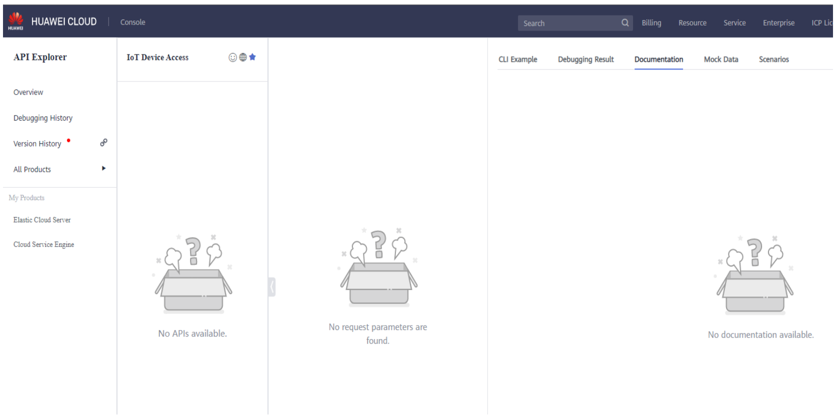
If no latest reported data is displayed on the device details page, modify the services and properties in the product model to ensure that the services and properties reported by the device are consistent with those in the product model. If they are inconsistent, the data reported by the device is not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

Delivering a Command

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

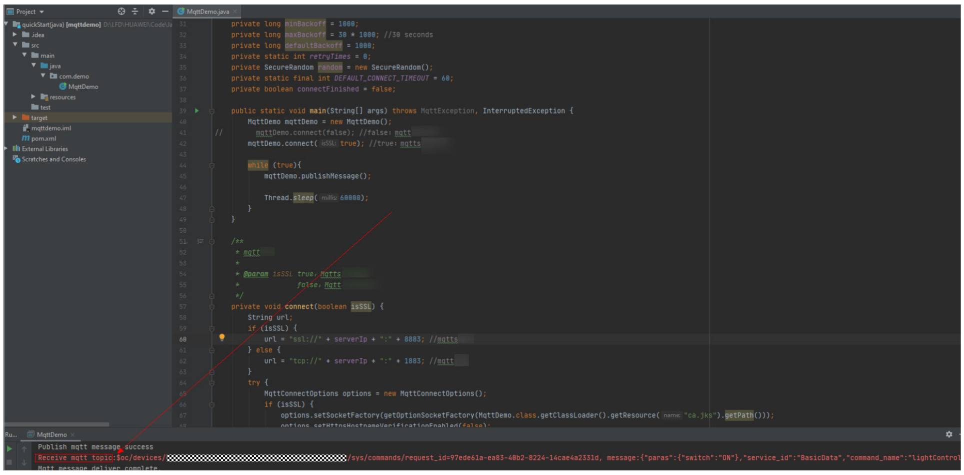


Step 2 Locate the row that contains the device command. For details about the delivered parameters, see the figure (consistent with those in the product model). Then, click **Debug** to send the command.



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

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



```
    private long minBackoff = 1000;
    private long maxBackoff = 30 * 1000; //30 seconds
    private long defaultBackoff = 1000;
    private static int retryTimes = 0;
    private SecureRandom random = new SecureRandom();
    private static final int DEFAULT_CONNECT_TIMEOUT = 60;
    private boolean connectFinished = false;

    public static void main(String[] args) throws MqttException, InterruptedException {
        MqttDemo mqttDemo = new MqttDemo();
//        mqttDemo.connect(false); //false: mqtt[redacted]
        mqttDemo.connect( isSSL: true); //true: mqtts[redacted]

        while (true){
            mqttDemo.publishMessage();

            Thread.sleep( millis: 60000);
        }
    }

    /**
     * mqtt[redacted]
     *
     * @param isSSL true: Mqtts[redacted]
     *              false: Mqtt[redacted]
     */
    private void connect(boolean isSSL) {
        String url;
        if (isSSL) {
            url = "ssl://" + serverIp + ":" + 8883; //mqtts[redacted]
        } else {
            url = "tcp://" + serverIp + ":" + 1883; //mqtt[redacted]
        }
        try {
            MqttConnectOptions options = new MqttConnectOptions();
            if (isSSL) {
                options.setSocketFactory(getOptionSocketFactory(MqttDemo.class.getClassLoader().getResource( name: "ca.jks").getPath()));
                options.setHttpsHostnameVerificationEnabled(false);
```

```
Publish mqtt message success
Receive mqtt topic:$oc/devices/[redacted]/sys/commands/request_id=97ede61a-ea83-40b2-8224-14cae4a2331d, message:{"paras":{"switch":"ON"},"service_id":"BasicData","command_name":"lightControl
Mqtt message deliver complete.
```

----End

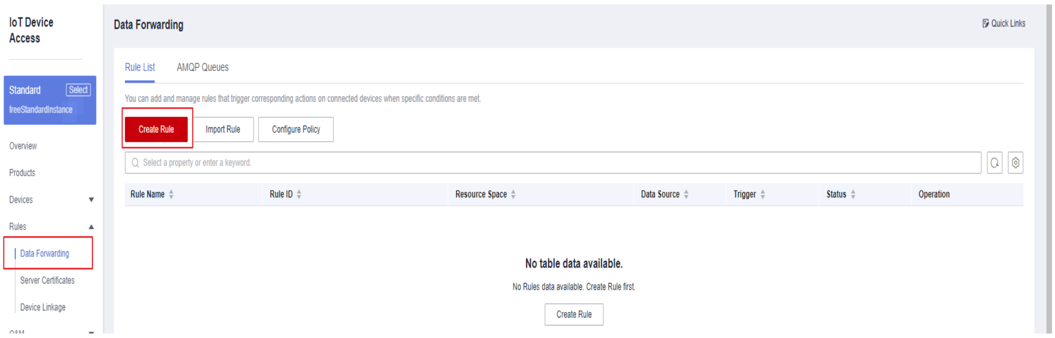
Obtaining Data Reported by a Device from the Cloud

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

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

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

Figure 2-6 Creating a data forwarding rule



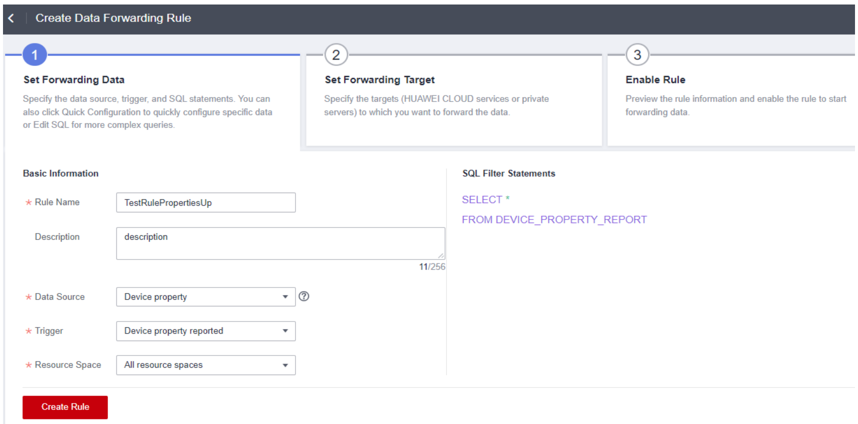
You can add and manage rules that trigger corresponding actions on connected devices when specific conditions are met.

Rule Name	Rule ID	Resource Space	Data Source	Trigger	Status	Operation

No table data available.

No Rules data available. Create Rule first.

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



Set Forwarding Data
Specify the data source, trigger, and SQL statements. You can also click Quick Configuration to quickly configure specific data or Edit SQL for more complex queries.

Set Forwarding Target
Specify the targets (HUAWEI CLOUD services or private servers) to which you want to forward the data.

Enable Rule
Preview the rule information and enable the rule to start forwarding data.

Basic Information

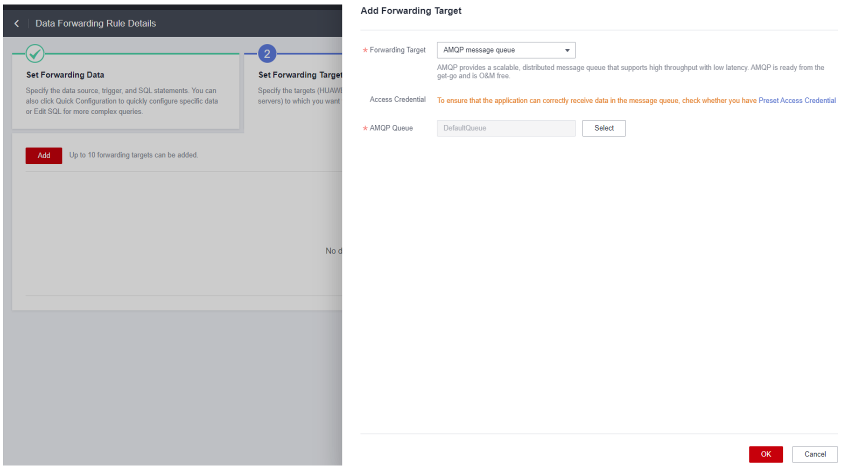
* Rule Name	TestRulePropertiesUp
Description	description 11/256
* Data Source	Device property
* Trigger	Device property reported
* Resource Space	All resource spaces

SQL Filter Statements

```
SELECT *
FROM DEVICE_PROPERTY_REPORT
```

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

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



Data Forwarding Rule Details
Set Forwarding Data
Specify the data source, trigger, and SQL statements. You can also click Quick Configuration to quickly configure specific data or Edit SQL for more complex queries.
2
Set Forwarding Target
Specify the targets (HUAWE
servers) to which you want
Add
Up to 10 forwarding targets can be added.
No d

Add Forwarding Target

* Forwarding Target: AMQP message queue

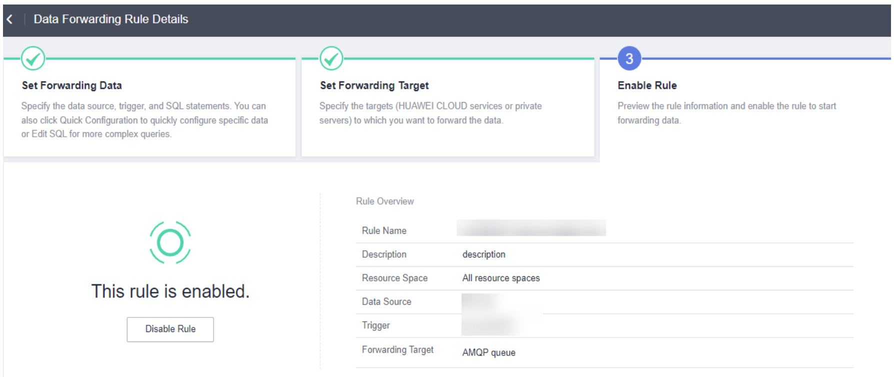
AMQP provides a scalable, distributed message queue that supports high throughput with low latency. AMQP is ready from the get-go and is O&M free.

Access Credential: To ensure that the application can correctly receive data in the message queue, check whether you have Preset Access Credential

* AMQP Queue: DefaultQueue Select

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

Step 5 Click **Enable Rule**.



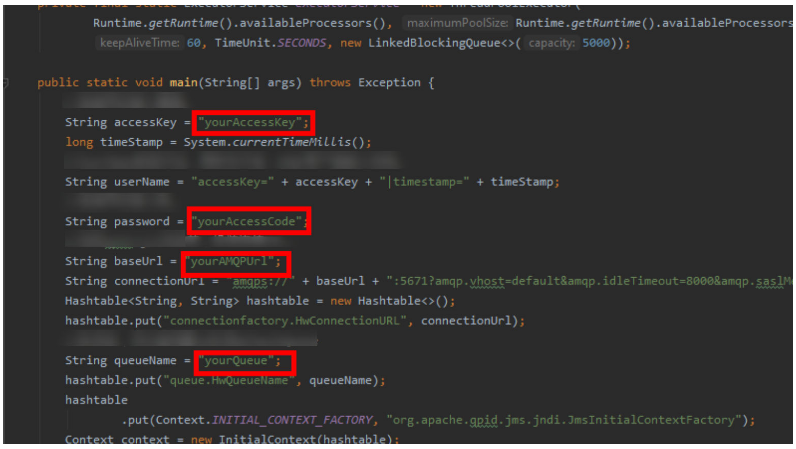
Data Forwarding Rule Details
Set Forwarding Data
Specify the data source, trigger, and SQL statements. You can also click Quick Configuration to quickly configure specific data or Edit SQL for more complex queries.
Set Forwarding Target
Specify the targets (HUAWEI CLOUD services or private servers) to which you want to forward the data.
3
Enable Rule
Preview the rule information and enable the rule to start forwarding data.

This rule is enabled.

Rule Overview

Rule Name	[redacted]
Description	description
Resource Space	All resource spaces
Data Source	[redacted]
Trigger	[redacted]
Forwarding Target	AMQP queue

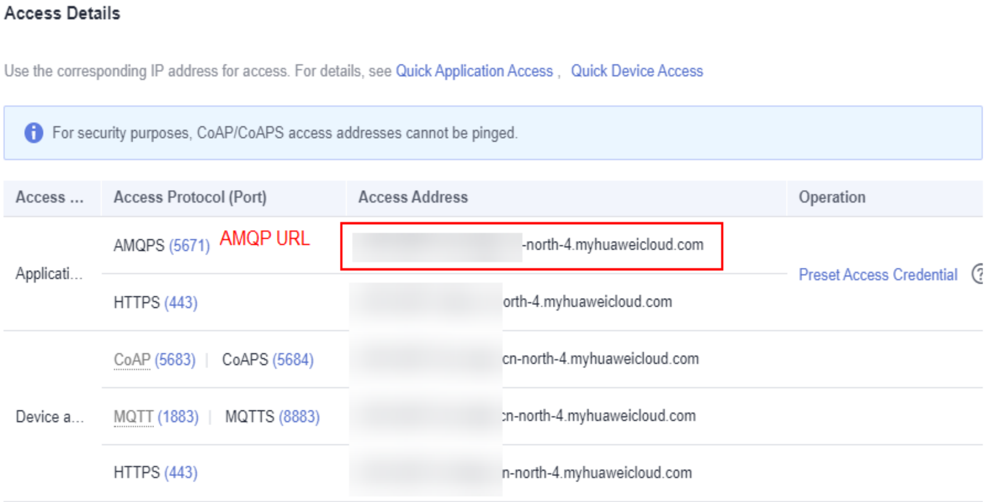
Step 6 Modify the parameters in the AMQP sample code obtained in **Step 1**.



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

public static void main(String[] args) throws Exception {
    [redacted]
    String accessKey = "yourAccessKey";
    long timeStamp = System.currentTimeMillis();
    [redacted]
    String userName = "accessKey=" + accessKey + "|timestamp=" + timeStamp;
    [redacted]
    String password = "yourAccessCode";
    [redacted]
    String baseUrl = "yourAMQPUrl";
    String connectionUrl = "amqps://" + baseUrl + ":5671?amqp.vhost=default&amqp.idleTimeout=8000&amqp.saslM
    Hashtable<String, String> hashtable = new Hashtable<>();
    hashtable.put("connectionfactory.HwConnectionURL", connectionUrl);
    [redacted]
    String queueName = "yourQueue";
    hashtable.put("queue.HwQueueName", queueName);
    hashtable
            .put(Context.INITIAL_CONTEXT_FACTORY, "org.apache.qpid.jms.jndi.JmsInitialContextFactory");
    Context context = new InitialContext(hashtable);
```

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



Access Details

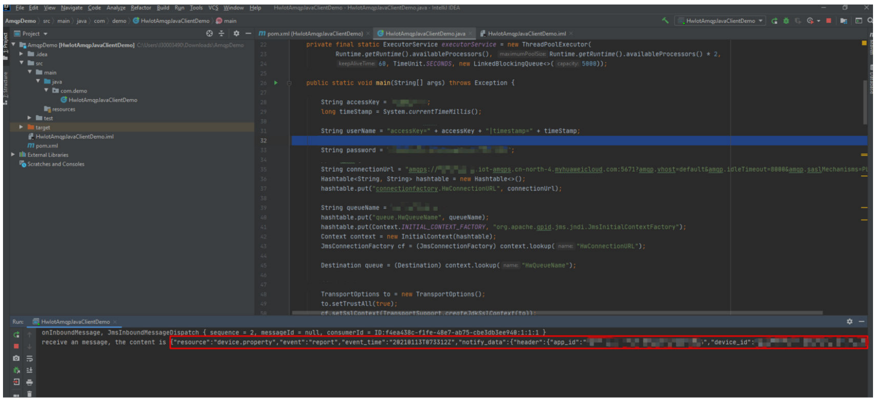
Use the corresponding IP address for access. For details, see Quick Application Access , Quick Device Access

For security purposes, CoAP/CoAPS access addresses cannot be pinged.

Access ...	Access Protocol (Port)	Access Address	Operation
Applicati...	AMQPS (5671) AMQP URL	[redacted]-north-4.myhuaweicloud.com	Preset Access Credential
	HTTPS (443)	[redacted]orth-4.myhuaweicloud.com	
Device a...	CoAP (5683) | CoAPS (5684)	[redacted]cn-north-4.myhuaweicloud.com	
	MQTT (1883) | MQTTS (8883)	[redacted]:n-north-4.myhuaweicloud.com	
	HTTPS (443)	[redacted]n-north-4.myhuaweicloud.com	

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

Step 7 AMQP data is received successfully.



```
    private final static ExecutorService executorService = new ThreadPoolExecutor(
            Runtime.getRuntime().availableProcessors(), maximumPoolSize: Runtime.getRuntime().availableProcessors() * 2,
            keepAliveTime: 60, TimeUnit.SECONDS, new LinkedBlockingQueue<>( capacity: 5000));

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

        String accessKey = [redacted];
        long timeStamp = System.currentTimeMillis();

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

        String password = [redacted];

        String connectionUrl = "amqps://[redacted].iot-amqps.cn-north-4.myhuaweicloud.com:5671?amqp.vhost=default&amqp.idleTimeout=8000&amqp.saslMechanisms=Pl
        Hashtable<String, String> hashtable = new Hashtable<>();
        hashtable.put("connectionfactory.HwConnectionURL", connectionUrl);

        String queueName = [redacted]
        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( name: "HwConnectionURL");

        Destination queue = (Destination) context.lookup( name: "HwQueueName");

        TransportOptions to = new TransportOptions();
        to.setTrustAll(true);
        cf.setSslContext(TransportSupport.createJdkSslContext(to));
```

```
onInboundMessage, JmsInboundMessageDispatch { sequence = 2, messageId = null, consumerId = ID:f4ea438c-f1fe-48e7-ab75-cbe3db3ee940:1:1:1 }
receive an message, the content is {"resource":"device.property","event":"report","event_time":"20210113T073312Z","notify_data":{"header":{"app_id":"[redacted]","device_id":[redacted]
```

----End

Additional Information

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

2.3.2 Quick Experience Based on C Sample Code

Overview

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

Prerequisites

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

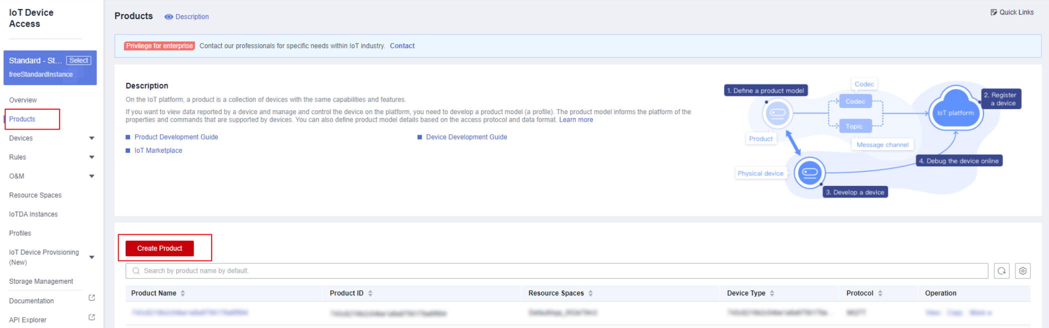
Uploading a Product Model

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

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

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

Figure 2-7 Creating a product



Contact our professionals for specific needs within IoT industry.

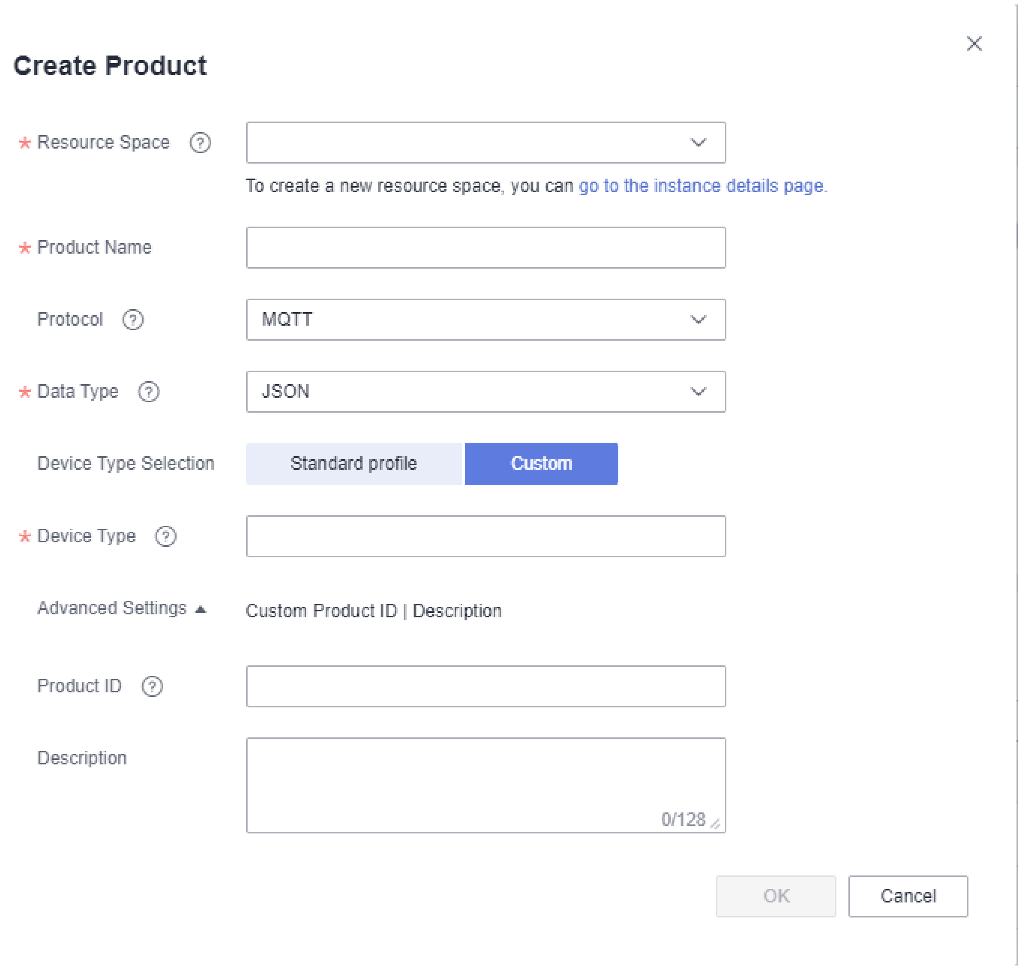
Description

On the IoT platform, a product is a collection of devices with the same capabilities and features.

If you want to view data reported by a device and manage and control the device on the platform, you need to develop a product model (a profile). The product model informs the platform of the properties and commands that are supported by devices. You can also define product model details based on the access protocol and data format. Learn more

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

Figure 2-8 Creating an MQTT product

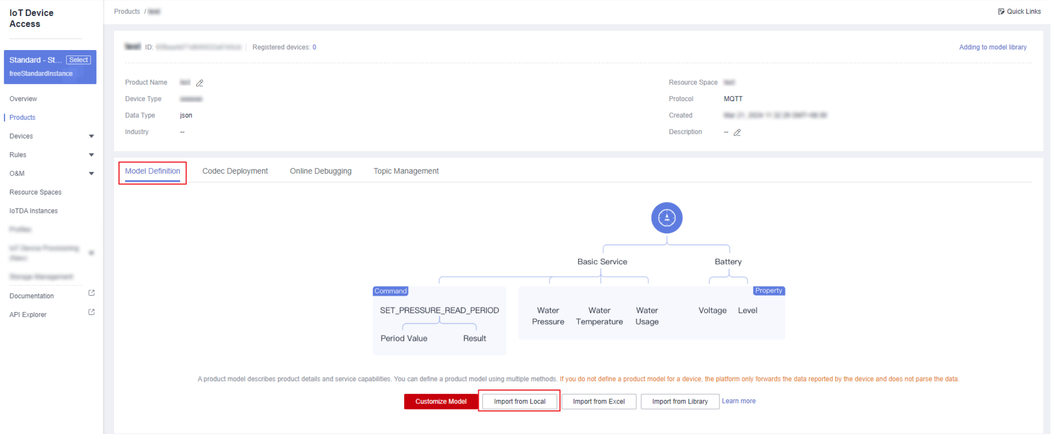


* Resource Space ⓘ
To create a new resource space, you can go to the instance details page.
* Product Name
Protocol ⓘ MQTT
* Data Type ⓘ JSON
Device Type Selection: Custom
* Device Type ⓘ
Advanced Settings ▲ Custom Product ID | Description
Product ID ⓘ
Description 0/128

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

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

Figure 2-9 Uploading an MQTT product model



[redacted] ID: [redacted] | Registered devices: 0

Product Name [redacted]

Device Type [redacted]

Data Type json

Industry --

Resource Space [redacted]

Protocol MQTT

Created [redacted]

Description --

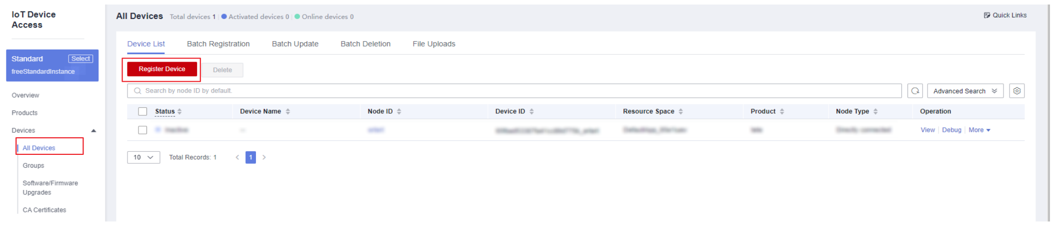
A product model describes product details and service capabilities. You can define a product model using multiple methods. If you do not define a product model for a device, the platform only forwards the data reported by the device and does not parse the data.

----End

Creating a Device

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

Figure 2-10 Registering a device

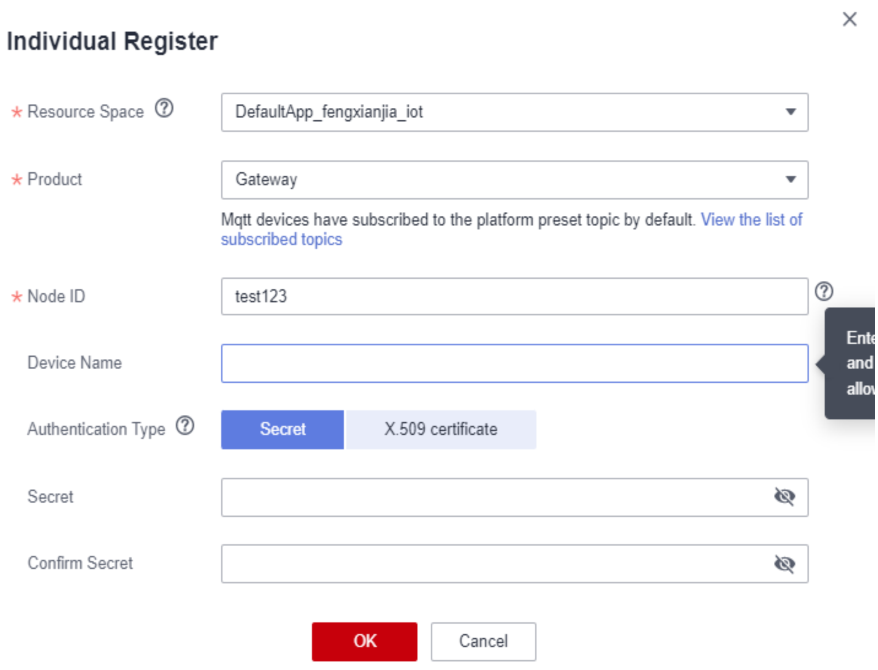


All Devices Total devices 1 | Activated devices 0 | Online devices 0

☐	Status	Device Name	Node ID	Device ID	Resource Space	Product	Node Type	Operation
☐	[redacted]	--	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	View | Debug | More

10 Total Records: 1 < 1 >

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



Individual Register

* Resource Space: DefaultApp_fengxianjia_iot

* Product: Gateway

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

* Node ID: test123

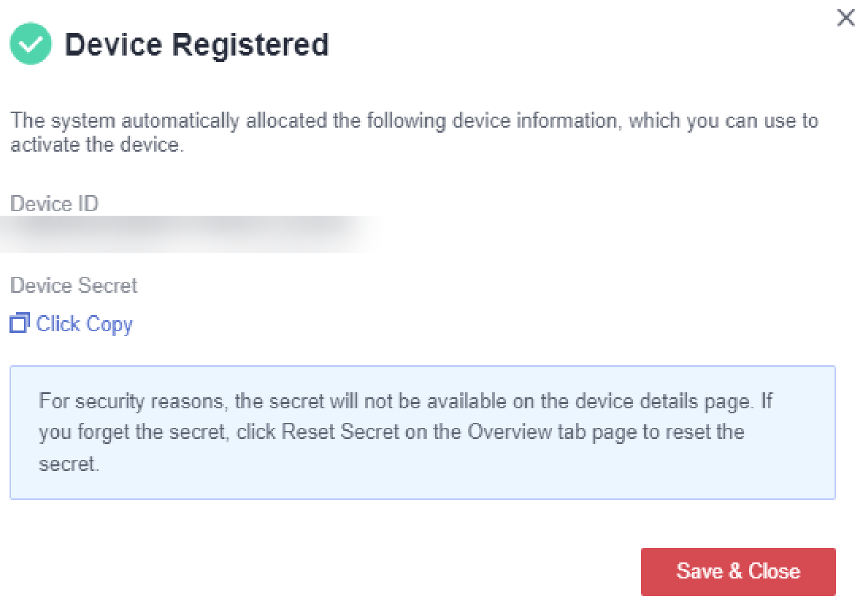
Device Name:

Authentication Type: Secret | X.509 certificate

Secret:

Confirm Secret:

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



Device Registered

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

Device ID

[redacted]

Device Secret

Click Copy

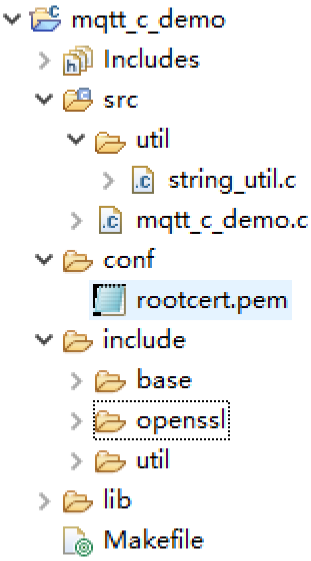
For security reasons, the secret will not be available on the device details page. If you forget the secret, click Reset Secret on the Overview tab page to reset the secret.

----End

Importing Sample Code

Step 1 Download the sample code quickStart(C).

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



```
mqtt_c_demo
  Includes
  src
    util
      string_util.c
    mqtt_c_demo.c
  conf
    rootcert.pem
  include
    base
    openssl
    util
  lib
  Makefile
```

Description of the directories:

- **src**: source code directory

  mqtt_c_demo: core source code of the demo

  util/string_util.c: utility resource file

- **conf**: certificate directory

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

- **include**: header files

  base: dependent Paho header files

  openssl: dependent OpenSSL header files

  util: header files of the dependent tool resources

- **lib**: dependent library file

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

  libpaho-mqtt3as.so*: Paho library file

- **Makefile**: Makefile

----End

Compiling Library Files

- Compiling the OpenSSL library

  a. Download **OpenSSL**, upload it to any directory on the Linux compiler, and run the following command to decompress it:

  ```
  tar -zxvf openssl-1.1.1d.tar.gz
  ```

  b. Generate a **makefile**.

  Run the following command to access the OpenSSL source code directory:

  ```
  cd openssl-1.1.1d
  ```

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

  ```
  mkdir /home/test
  ```

Create a directory for OpenSSL compilation.

```
mkdir /home/test/openssl
```

Create a configuration file directory.

```
mkdir /home/test/openssl/ssl
```

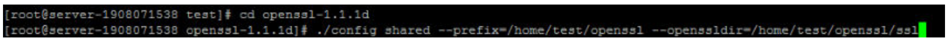
Run the following configuration command:

```
./config shared --prefix=/home/test/openssl --openssldir=/home/test/openssl/ssl
```

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

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

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



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

c. Generate library files.

Run the following command in the OpenSSL source code directory:

```
make depend
```

Run the following command for compilation:

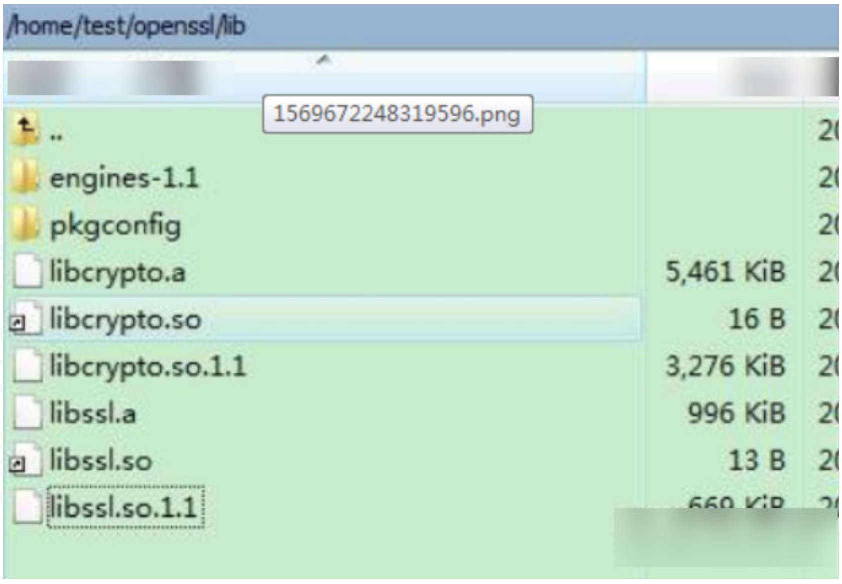
```
make
```

Install OpenSSL.

```
make install
```

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

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



```
/home/test/openssl/lib
1569672248319596.png
..
engines-1.1
pkgconfig
libcrypto.a          5,461 KiB
libcrypto.so              16 B
libcrypto.so.1.1     3,276 KiB
libssl.a               996 KiB
libssl.so                 13 B
libssl.so.1.1          660 KiB
```

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

- Compiling the Eclipse Paho library file

a. Download the **paho.mqtt.c source code**.

b. Decompress the package and upload it to the Linux compiler.

c. Modify the **makefile**.

   i. Run the following command to edit the **makefile**:

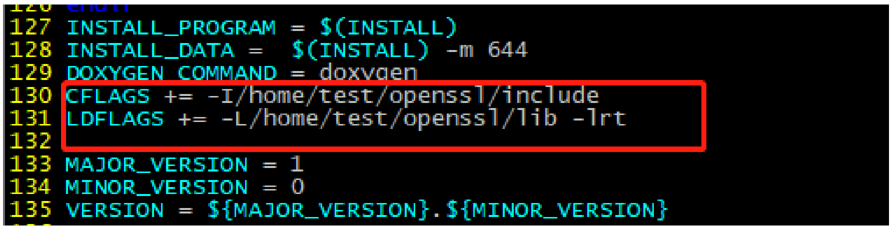
   ```
   vim Makefile
   ```

   ii. Run the following command to display the number of lines:

   ```
   :set nu
   ```

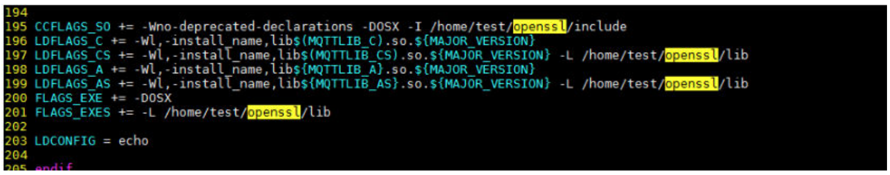
   iii. Add the following two lines (customized OpenSSL header files and library files) after line 129:

   ```
   CFLAGS += -I/home/test/openssl/include
   LDFLAGS += -L/home/test/openssl/lib -lrt
   ```



```
127 INSTALL_PROGRAM = $(INSTALL)
128 INSTALL_DATA =  $(INSTALL) -m 644
129 DOXYGEN_COMMAND = doxygen
130 CFLAGS += -I/home/test/openssl/include
131 LDFLAGS += -L/home/test/openssl/lib -lrt
132
133 MAJOR_VERSION = 1
134 MINOR_VERSION = 0
135 VERSION = ${MAJOR_VERSION}.${MINOR_VERSION}
```

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



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

d. Start the compilation.

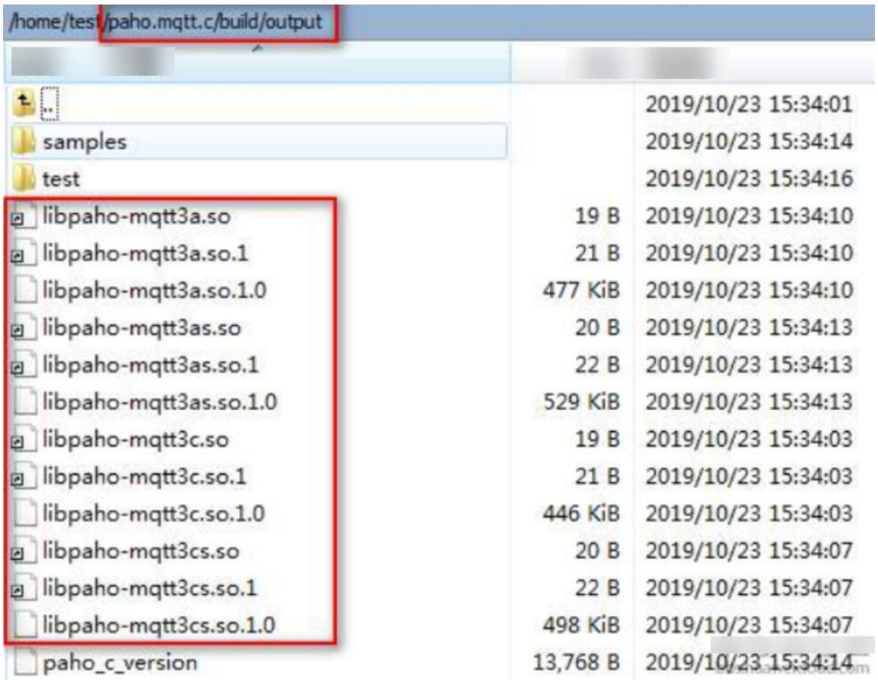
   i. Run the following command:

   ```
   make clean
   ```

   ii. Run the following command:

   ```
   make
   ```

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



/home/test/paho.mqtt.c/build/output

..		2019/10/23 15:34:01
samples		2019/10/23 15:34:14
test		2019/10/23 15:34:16
libpaho-mqtt3a.so	19 B	2019/10/23 15:34:10
libpaho-mqtt3a.so.1	21 B	2019/10/23 15:34:10
libpaho-mqtt3a.so.1.0	477 KiB	2019/10/23 15:34:10
libpaho-mqtt3as.so	20 B	2019/10/23 15:34:13
libpaho-mqtt3as.so.1	22 B	2019/10/23 15:34:13
libpaho-mqtt3as.so.1.0	529 KiB	2019/10/23 15:34:13
libpaho-mqtt3c.so	19 B	2019/10/23 15:34:03
libpaho-mqtt3c.so.1	21 B	2019/10/23 15:34:03
libpaho-mqtt3c.so.1.0	446 KiB	2019/10/23 15:34:03
libpaho-mqtt3cs.so	20 B	2019/10/23 15:34:07
libpaho-mqtt3cs.so.1	22 B	2019/10/23 15:34:07
libpaho-mqtt3cs.so.1.0	498 KiB	2019/10/23 15:34:07
paho_c_version	13,768 B	2019/10/23 15:34:14

f. Copy the Paho library file.

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

Establishing a Connection

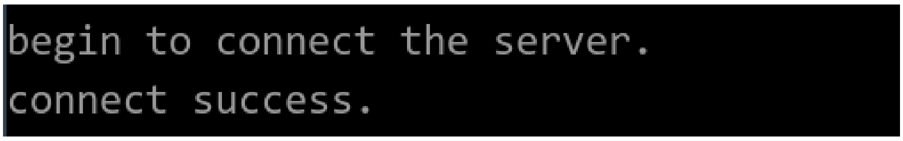
To connect a device or gateway to the platform, upload the device information to bind the device or gateway to the platform.

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

Step 2 Start the connection.

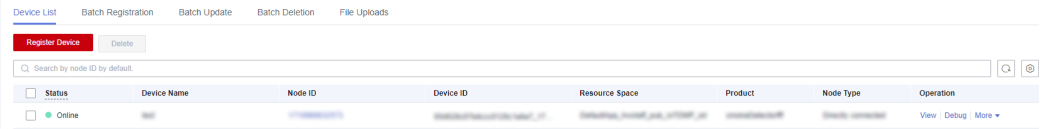
1. Run the **make** command for compilation. Delete **-m64** from the **makefile** in a 32-bit OS.
2. Run **export LD_LIBRARY_PATH=./lib/** to load the library file.
3. Run **./MQTT_Demo.o**.

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



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

Figure 2-11 Device online status



	Status	Device Name	Node ID	Device ID	Resource Space	Product	Node Type	Operation
□	● Online	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	[redacted]	View | Debug | More ▾

----End

Reporting Properties

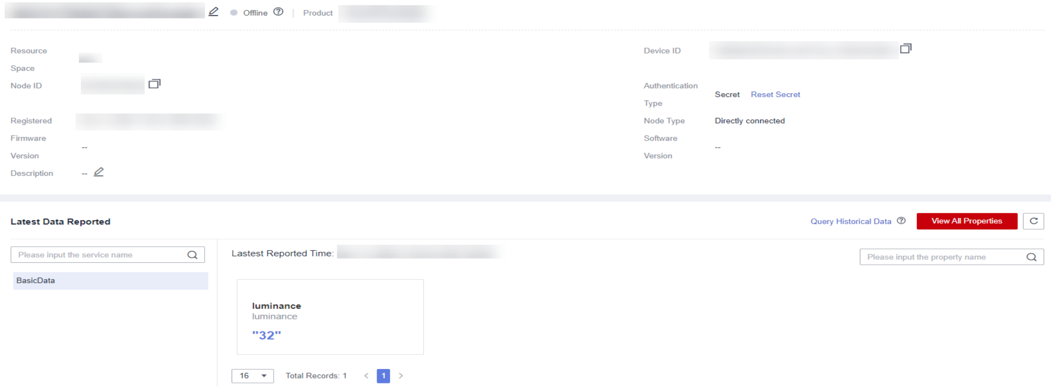
A device reports its properties to IoTDA. The sample code implements scheduled reporting. You can view the data reported by the device in IoTDA. For details, see **Device Reporting Properties**.

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

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

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

The reported properties are displayed on the device details page.



[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

BasicData

Lastest Reported Time: [redacted]

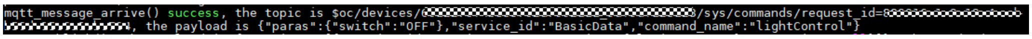
luminance
luminance
"32"

16 ▾ Total Records: 1 < 1 >

Receiving a Command

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

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



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

Obtaining Data Reported by a Device from the Cloud

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

Additional Information

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

3 Quick Application Access

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

Local Debugging

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

The procedure is as follows:

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

Step 2 Create an MQTT product.

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

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

- **Step 1 Enable IoTDA.**

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

- **Step 2 Create a product.**

  Create a product on IoTDA before calling APIs.

  a. Access the IoTDA service page and click **Access Console**.

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

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

Basic Information

Resource Space	The platform automatically allocates the created product to the default resource space. If you want to allocate the product to another resource space, select the resource space from the drop-down list. If a **resource space** does not exist, create one.
Product Name	Customize the product name. The value can contain up to 64 characters. Only letters, digits, and special characters (_?'#().,&%@!-) are allowed.
Protocol	MQTT is recommended.
Data Type	Select **JSON**.
Manufacturer	Customize the manufacturer name. The value can contain up to 32 characters. Only letters, digits, and special characters (_?'#().,&%@!-) are allowed.
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.
Device Type	If a product model preset on the platform is used, the device type is automatically matched and does not need to be manually specified.
Advanced Settings	
Product ID	Set a unique identifier for the product. If this parameter is specified, IoTDA uses the specified product ID. If this parameter is not specified, IoTDA allocates a product ID.
Description	Provide a description for the product. Set this parameter based on the site requirements.

- **Step 3 Configure the environment.**

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

- **Step 4 Call the service.**

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

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

Advanced Experience

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

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