Alibaba Cloud
IoT Platform

Quick Start

Issue: 20190921

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.
2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company, or individual in any form or by any means without the prior written consent of Alibaba Cloud.
3. The content of this document may be changed due to product version upgrades , adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.
4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults " and "as available" basis. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity , applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, special, or punitive damages, including lost profits arising from the use

or trust in this document, even if Alibaba Cloud has been notified of the possibility of such a loss.

5. By law, all the content of the Alibaba Cloud website, including but not limited to works, products, images, archives, information, materials, website architecture, website graphic layout, and webpage design, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of the Alibaba Cloud website, product programs, or content shall be used, modified , reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates . The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates).
6. Please contact Alibaba Cloud directly if you discover any errors in this document.

Generic conventions

Table -1: Style conventions

Style	Description	Example
⛔	This warning information indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	⛔ Danger: Resetting will result in the loss of user configuration data.
⚠	This warning information indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	⚠ Warning: Restarting will cause business interruption. About 10 minutes are required to restore business.
📋	This indicates warning information, supplementary instructions, and other content that the user must understand.	❗ Notice: Take the necessary precautions to save exported data containing sensitive information.
	This indicates supplemental instructions, best practices, tips, and other content that is good to know for the user.	📋 Note: You can use Ctrl + A to select all files.
>	Multi-level menu cascade.	Settings > Network > Set network type
Bold	It is used for buttons, menus, page names, and other UI elements.	Click OK.
`Courier font`	It is used for commands.	Run the `cd / d C :/ windows` command to enter the Windows system folder.
Italics	It is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	It indicates that it is a optional value, and only one item can be selected.	`ipconfig` *[-all|-t]*

Style	Description	Example
{} or {a|b}	**It indicates that it is a required value, and only one item can be selected.**	`swich` *{stand | slave}*

Contents

1 Use IoT Platform

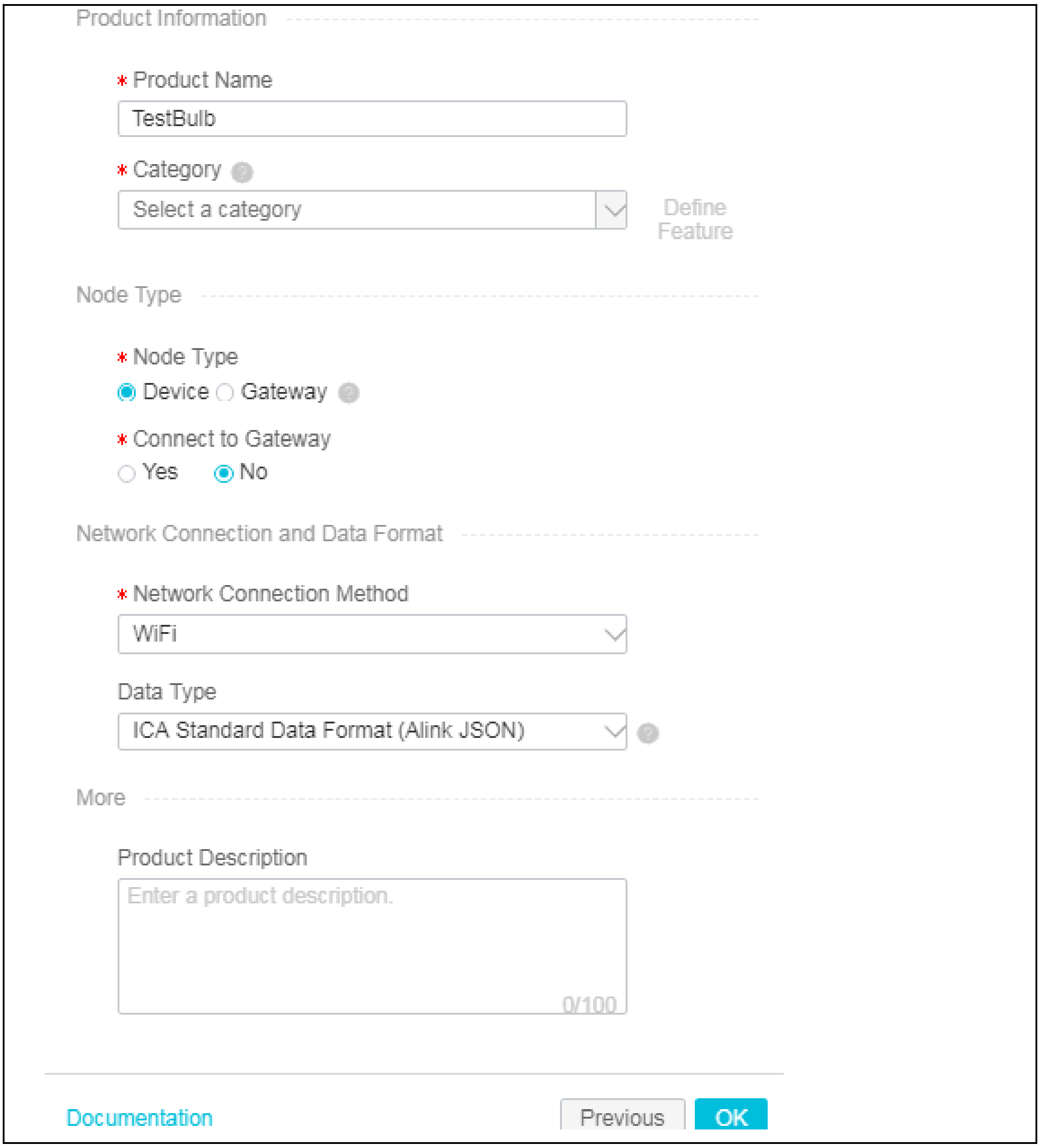
1.1 Create products and devices

The first step in using IoT Platform is to create products and devices. A product is a collection of devices that typically have the same features. You can manage devices in batch by managing the corresponding product.

Procedure

1. Log on to the IoT Platform console.

2. Create a product.
    a) In the left-side navigation pane, click Devices > Product. On the Products page, click Create Product.
    b) Enter all the required information and then click OK.



Product Information

* Product Name

TestBulb

* Category

Select a category

Define Feature

Node Type

* Node Type

● Device ○ Gateway

* Connect to Gateway

○ Yes ● No

Network Connection and Data Format

* Network Connection Method

WiFi

Data Type

ICA Standard Data Format (Alink JSON)

More

Product Description

Enter a product description.

0/100

The parameters are described as follows:

Parameter	Description
Product Name	In this example, the product is named as TestBulb. The product name must be unique within the account. A Product name is 4 to 30 characters in length, and can contain Chinese characters, English letters, digits and underscores. A Chinese character counts as two characters .
Category	In this example, the product category is Custom category indicating that features of the product is self-defined.
Node Type	In this example, the node type is Device. · Device: Indicates that devices of this product cannot be mounted with sub-devices. This kind of devices can connect to IoT Platform directly or as sub-devices of gateway devices. · Gateway: Indicates that devices of this product connect to IoT Platform directly and can be mounted with sub-devices. A gateway can manage sub-devices, maintain topological relationships with sub-devices, and synchronize topological relationships to IoT Platform.
Connect to Gateway Note: This parameter appears if the node type is Device.	Indicates whether or not devices of this product can be connected to gateways as sub-devices. · Yes: Devices of this product can be connected to a gateway. · No: Devices of this product cannot be connected to a gateway.
Network Connection Method Note: This parameter appears if you select No for Connect to Gateway.	Select a network connection method for the devices. In this example, WiFi is selected.

Parameter	Description
Data Type	Select a format in which devices exchange data with IoT Platform. In this example, ICA Standard Data Format (Alink JSON) is selected. ICA Standard Data Format (Alink JSON): The standard data format defined by IoT Platform for device and IoT Platform communication.
Product Description	Describe the product information. You can enter up to 100 characters.

Once the product is created successfully, it appears in the product list.

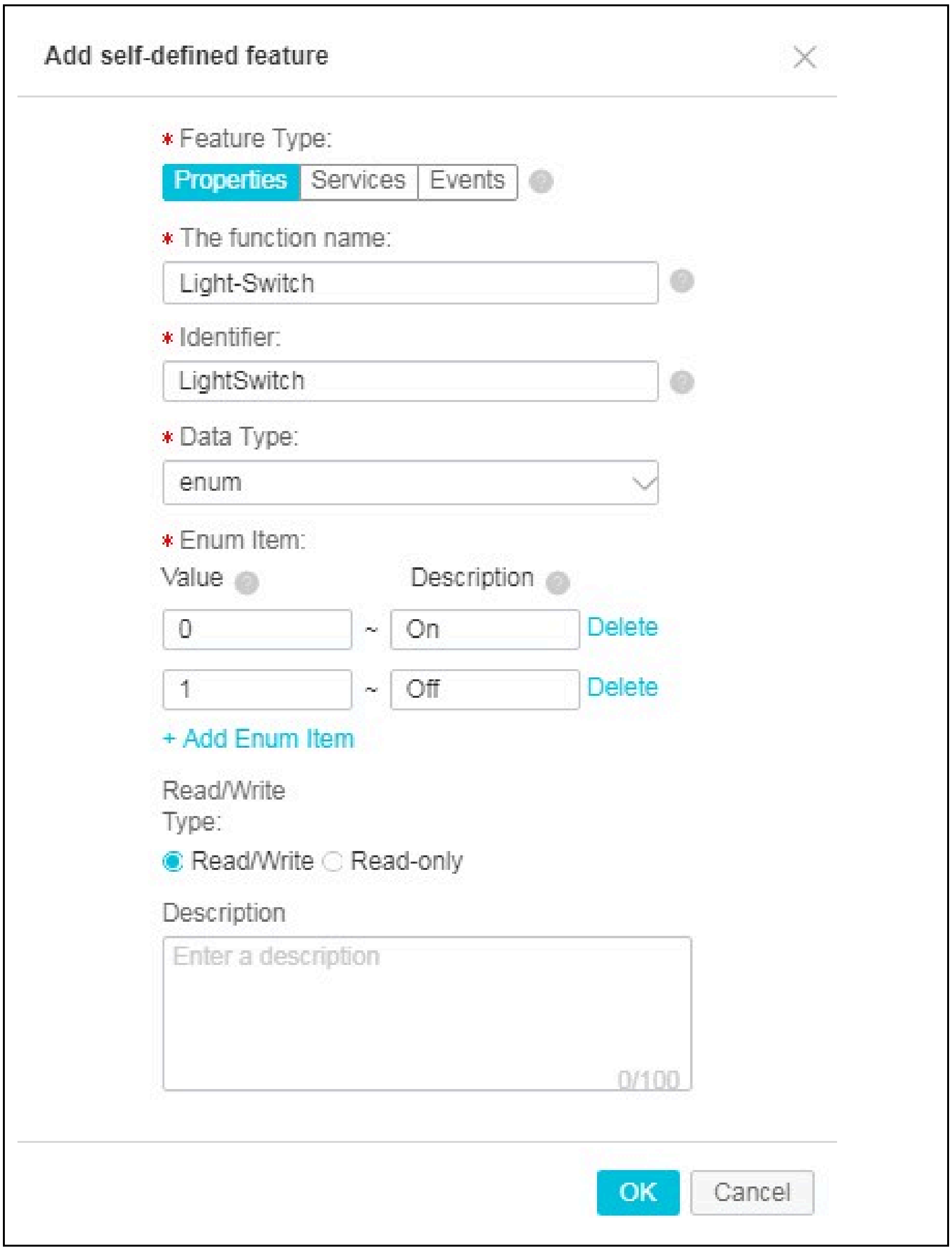
3. Define features for the product.

   a) In the product list, find the product and click View.

   b) On the product details page, click Define Feature.

   c) Click Add Feature corresponding to Self-Defined Feature.

   d) Define a property. In this example, a light switch property is defined. 0 indicates turning the light on and 1 indicates turning the light off.



Add self-defined feature

* Feature Type: Properties | Services | Events

* The function name: Light-Switch

* Identifier: LightSwitch

* Data Type: enum

* Enum Item:

Value		Description	
0	~	On	Delete
1	~	Off	Delete

+ Add Enum Item

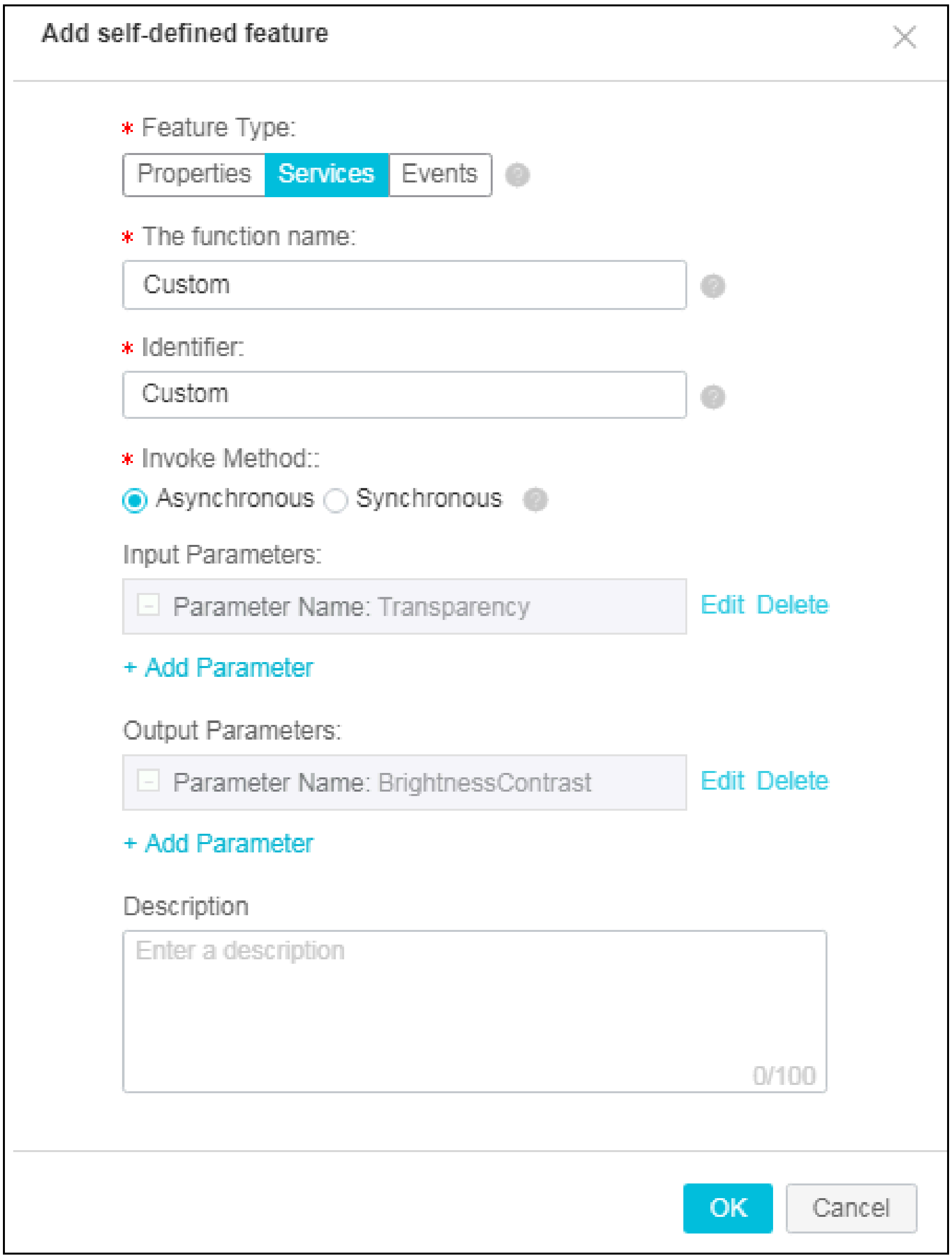
Read/Write Type: ● Read/Write ○ Read-only

Description

Enter a description

0/100

   e) Define a service. For example, you can add an input parameter for adjusting the brightness of the bulb, and add an output parameter for the bulb to report the brightness contrast between the bulb and the room environment.



Add self-defined feature

* Feature Type: Services

* The function name: Custom

* Identifier: Custom

* Invoke Method:: ● Asynchronous ○ Synchronous

Input Parameters:

Parameter Name: Transparency

Output Parameters:

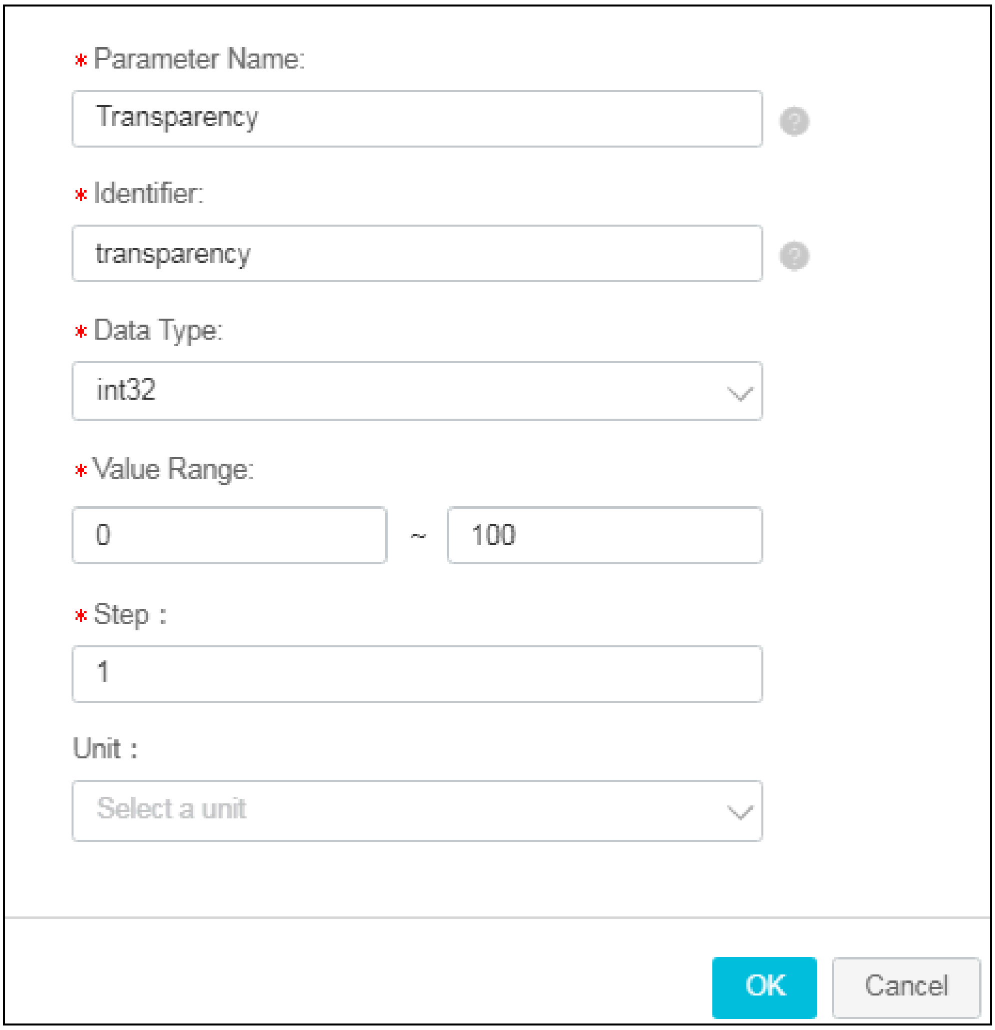
Parameter Name: BrightnessContrast

Description

Enter a description

0/100

The following figure shows an example of input parameter.



* Parameter Name: Transparency

* Identifier: transparency

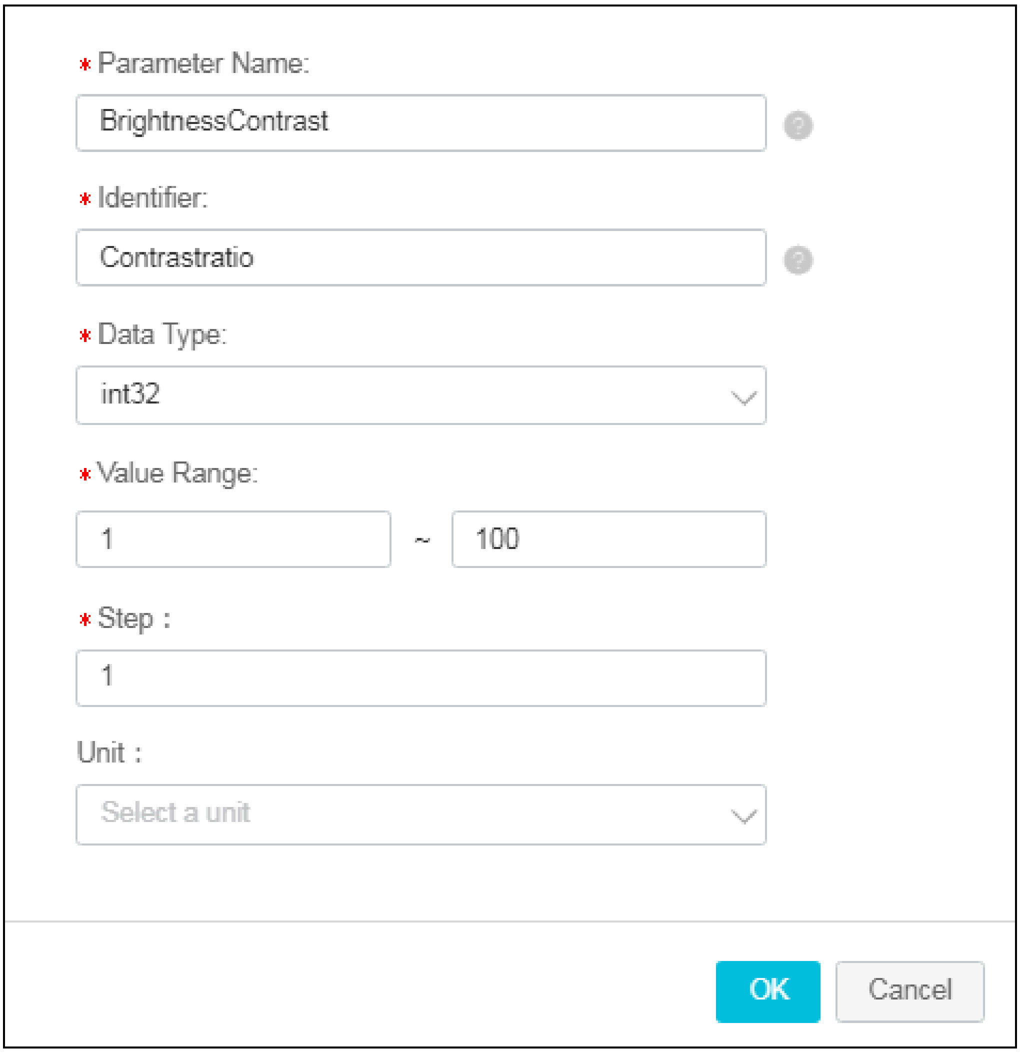
* Data Type: int32

* Value Range: 0 ~ 100

* Step : 1

Unit : Select a unit

The following figure shows an example of output parameter.



* Parameter Name:

BrightnessContrast

* Identifier:

Contrastratio

* Data Type:

int32

* Value Range:

1 ~ 100

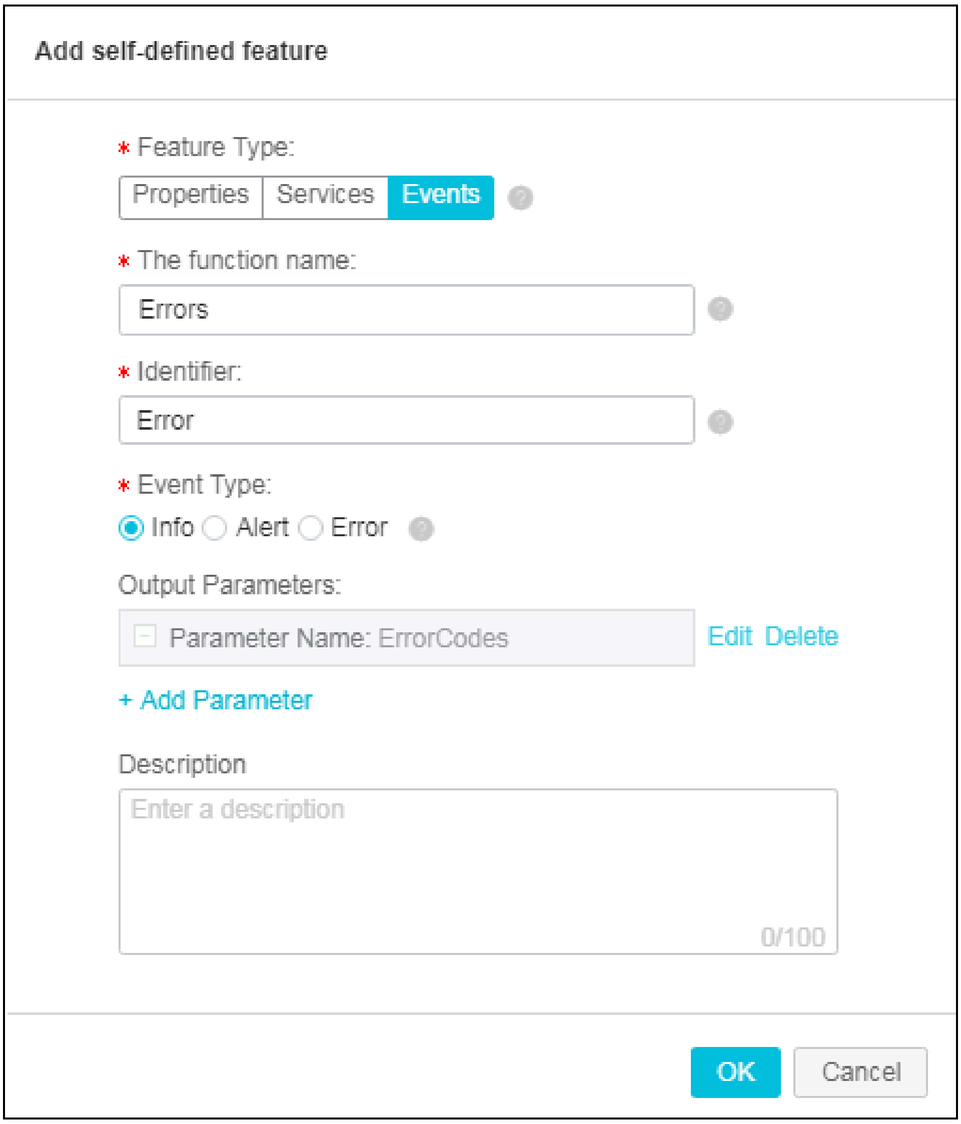
* Step :

1

Unit :

Select a unit

f) Define an event. You can define an event for devices to report errors.



Add self-defined feature

* Feature Type:

Properties | Services | **Events**

* The function name:

Errors

* Identifier:

Error

* Event Type:

● Info ○ Alert ○ Error

Output Parameters:

Parameter Name: ErrorCodes Edit Delete

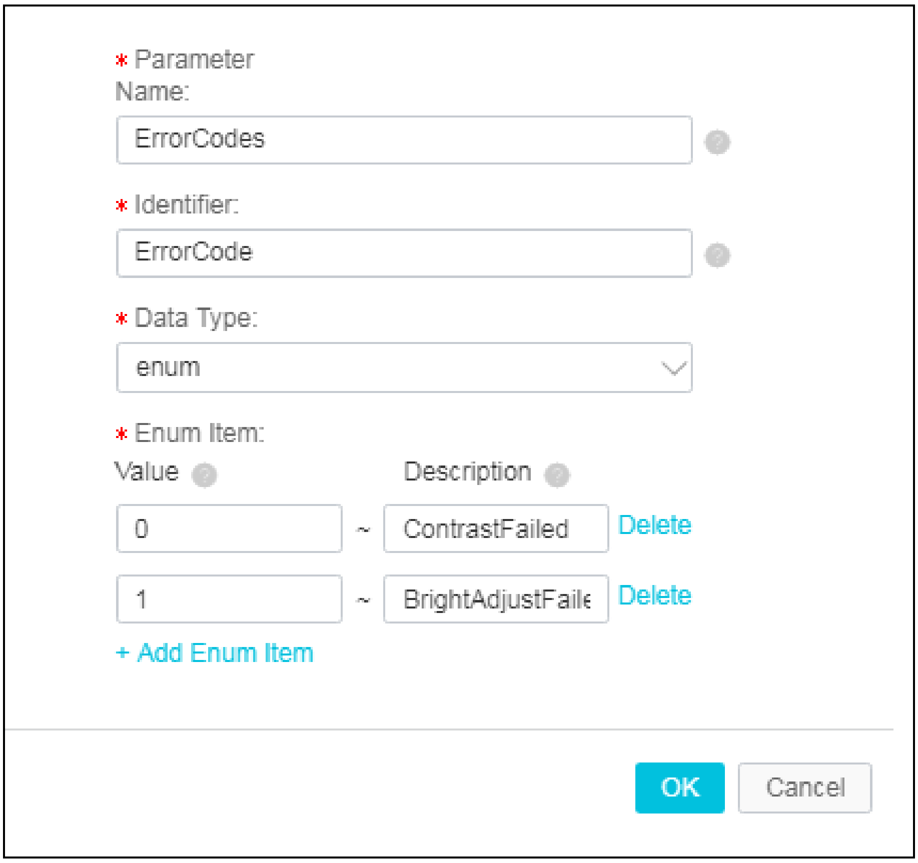
+ Add Parameter

Description

Enter a description 0/100

OK Cancel

The following figure shows an example of output parameter.



* Parameter Name: ErrorCodes

* Identifier: ErrorCode

* Data Type: enum

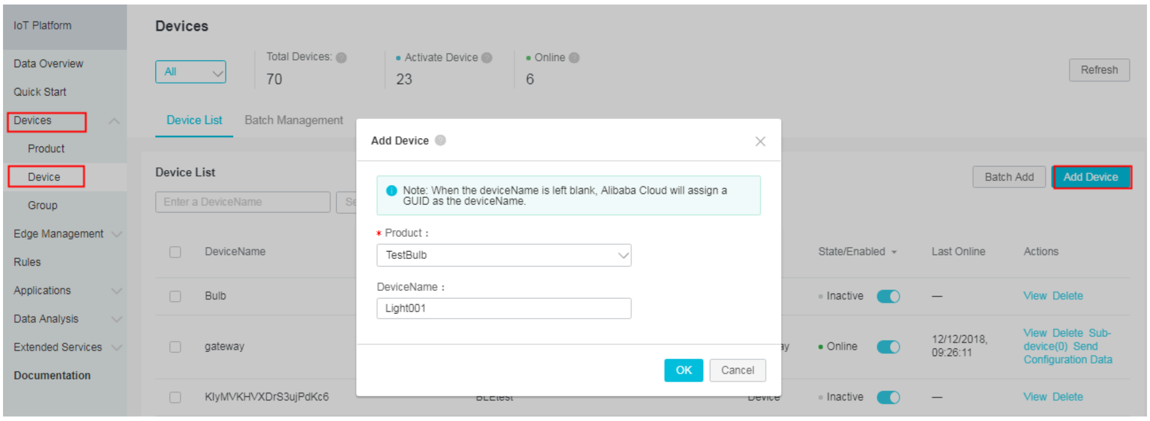
* Enum Item:

Value		Description	
0	~	ContrastFailed	Delete
1	~	BrightAdjustFaile	Delete

4. Create a device.

   a) In the left-side navigation pane, click Devices > Device.

   b) On the device management page, click Add Device. Select a product to which the device to be created belongs, and then enter a name for the device (DeviceName). Click OK.



Total Devices: 70

Activate Device 23

Online 6

Add Device

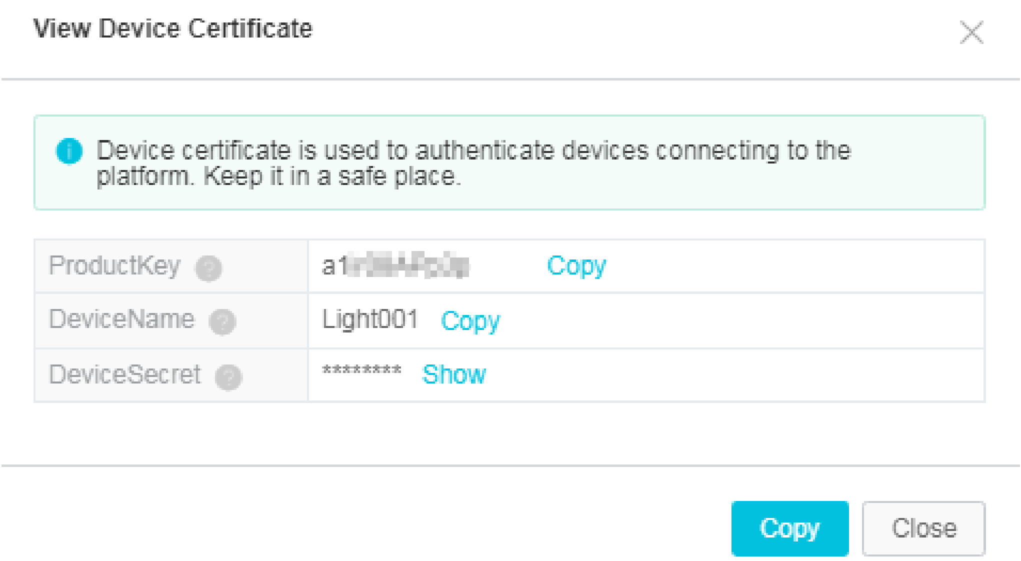
Note: When the deviceName is left blank, Alibaba Cloud will assign a GUID as the deviceName.

* Product : TestBulb

DeviceName : Light001

DeviceName	State/Enabled	Last Online	Actions
Bulb	Inactive	—	View Delete
gateway	Online	12/12/2018, 09:26:11	View Delete Sub-device(0) Send Configuration Data
KlyMVKHVXDrS3ujPdKc6	Inactive	—	View Delete

   c) Save the device certificate information. The certificate information includes ProductKey, DeviceName, and DeviceSecret. Keep this information confidential, because it is the certificate that will be used for device authentication when the device is connecting to IoT Platform.



View Device Certificate

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

ProductKey	a1[redacted] Copy
DeviceName	Light001 Copy
DeviceSecret	******** Show

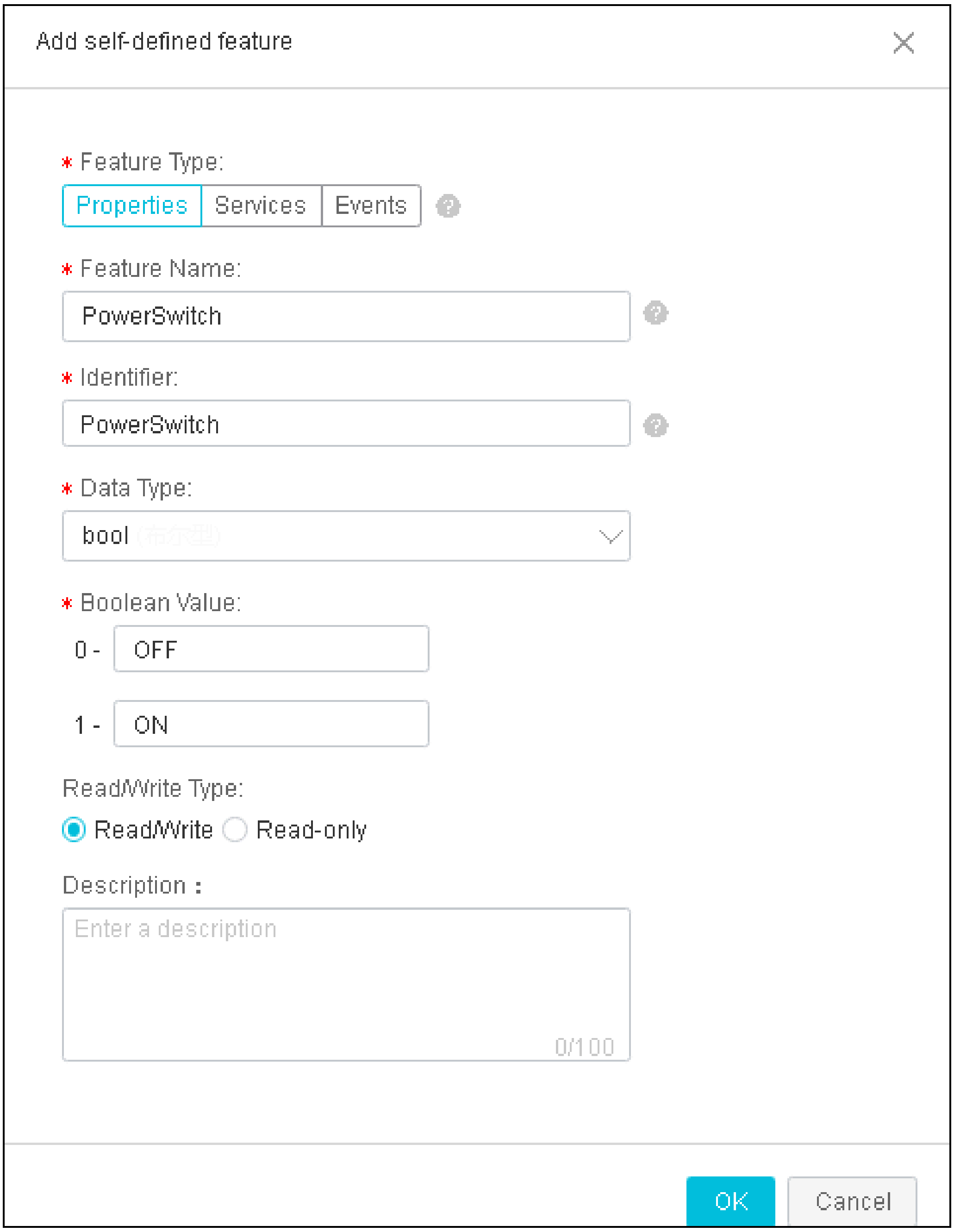
1.2 Define product features

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

Procedure

1. In the product list, select the product and click View.
2. On the Product Details page, click Define Feature.
3. In the Self-Defined Feature section, click Add Feature.

4. As shown below, add a property to define a switch. Click OK.



Add self-defined feature

* Feature Type:

Properties | Services | Events

* Feature Name:

PowerSwitch

* Identifier:

PowerSwitch

* Data Type:

bool (布尔型)

* Boolean Value:

0 - OFF

1 - ON

Read/Write Type:

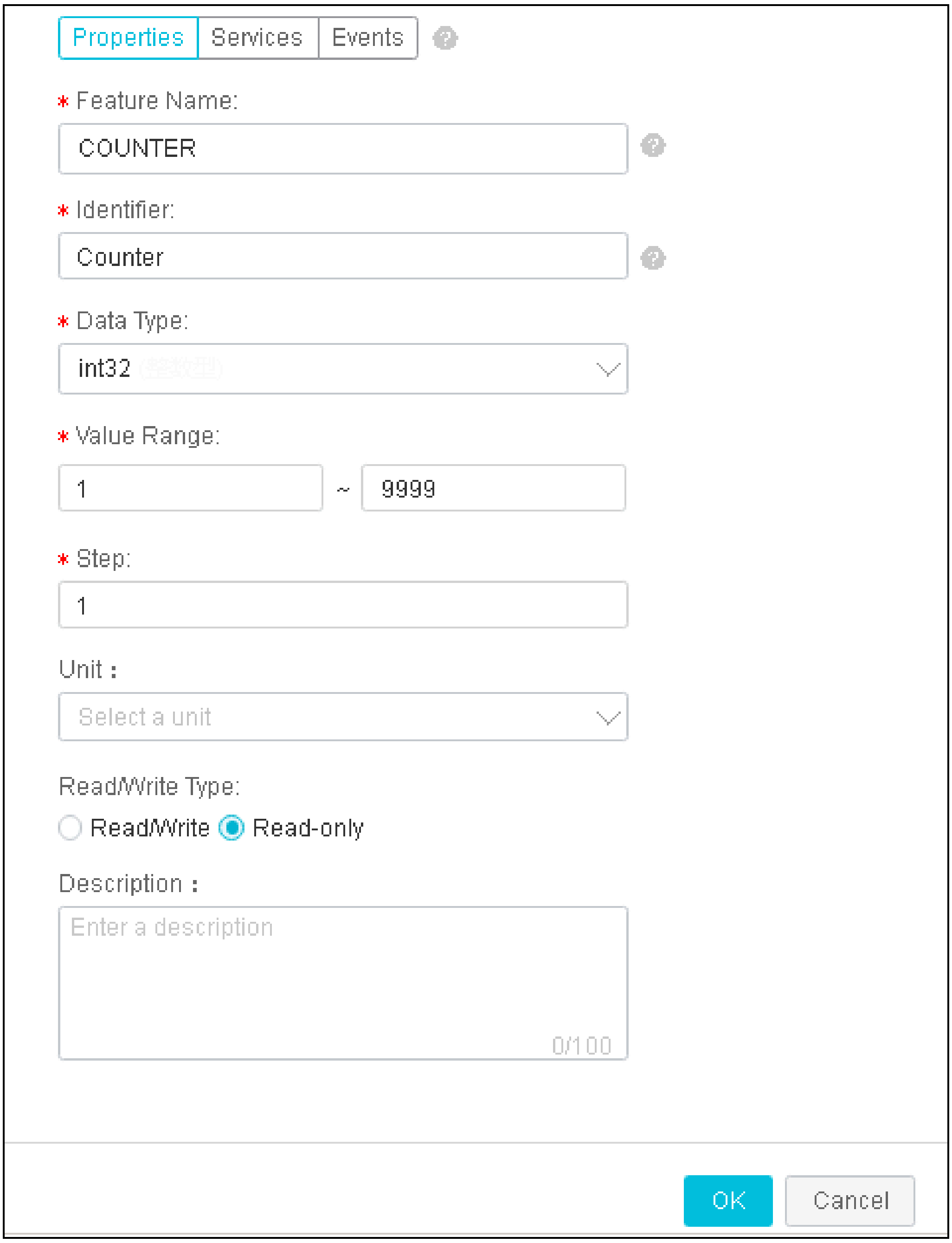
● Read/Write ○ Read-only

Description :

Enter a description

0/100

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



* Feature Name: COUNTER

* Identifier: Counter

* Data Type: int32 (整数型)

* Value Range: 1 ~ 9999

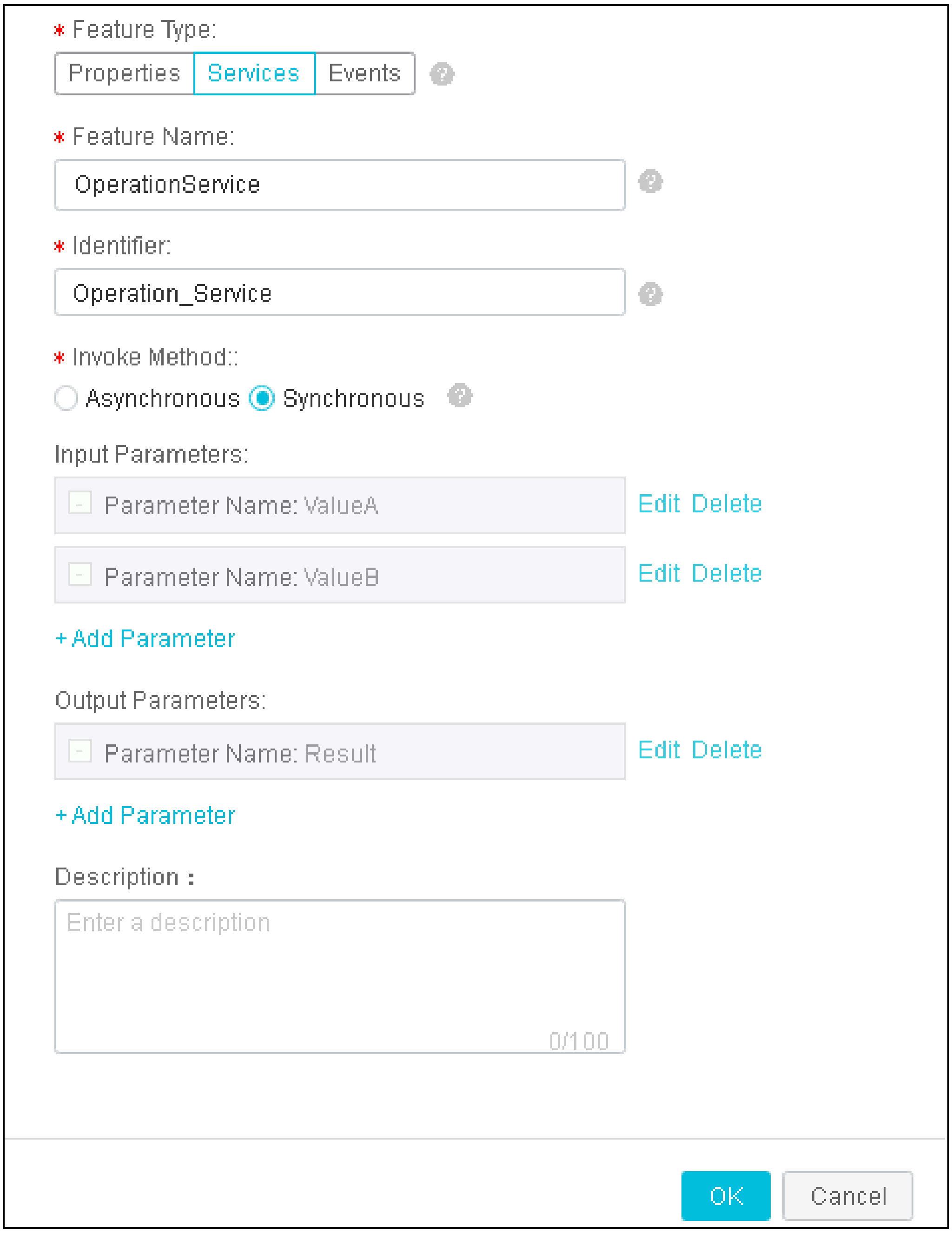
* Step: 1

Unit : Select a unit

Read/Write Type: ○ Read/Write ● Read-only

Description : Enter a description 0/100

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



* Feature Type:

Properties | Services | Events

* Feature Name:

OperationService

* Identifier:

Operation_Service

* Invoke Method::

○ Asynchronous ● Synchronous

Input Parameters:

Parameter Name: ValueA Edit Delete

Parameter Name: ValueB Edit Delete

+Add Parameter

Output Parameters:

Parameter Name: Result Edit Delete

+Add Parameter

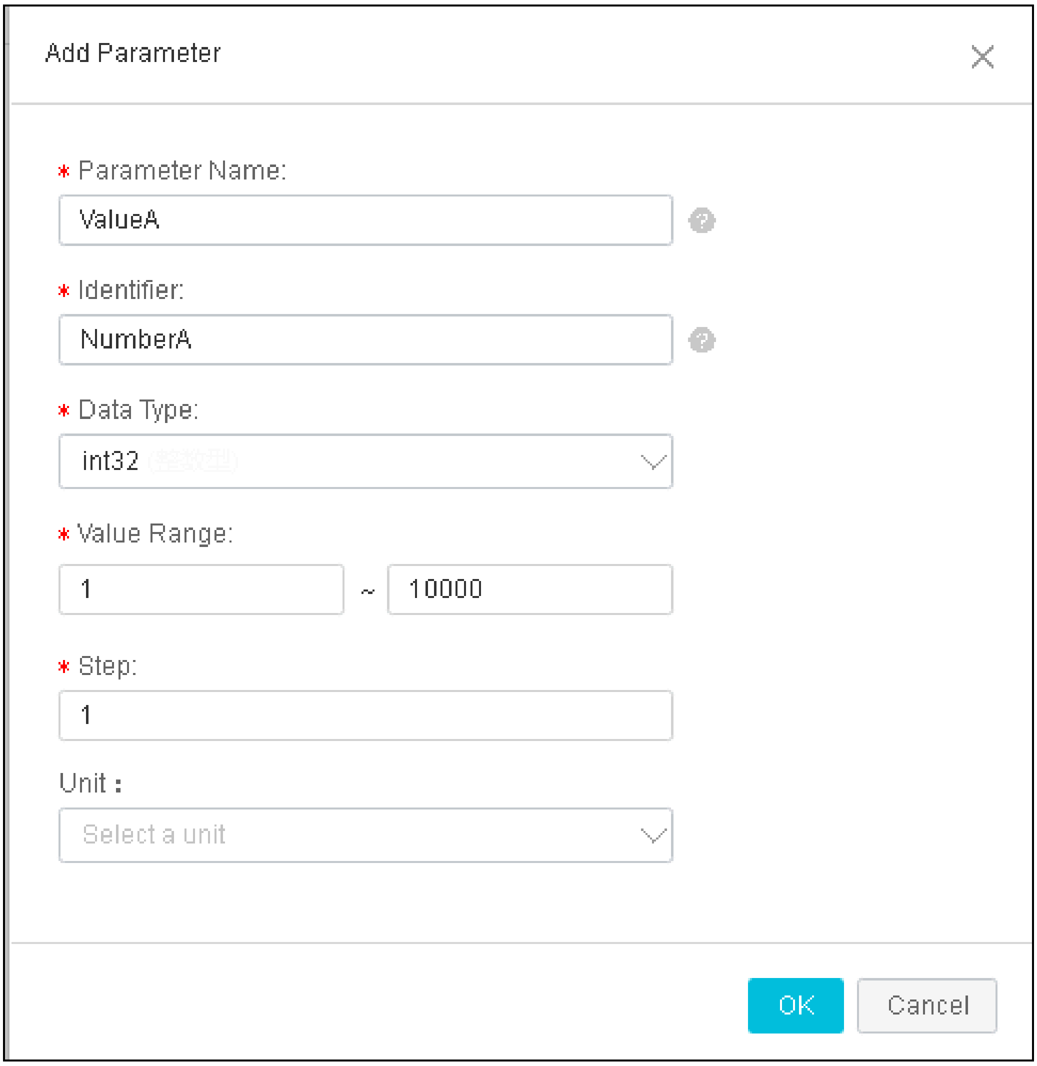
Description :

Enter a description

0/100

OK Cancel

- Value A is defined as follows:



Add Parameter

* Parameter Name: ValueA

* Identifier: NumberA

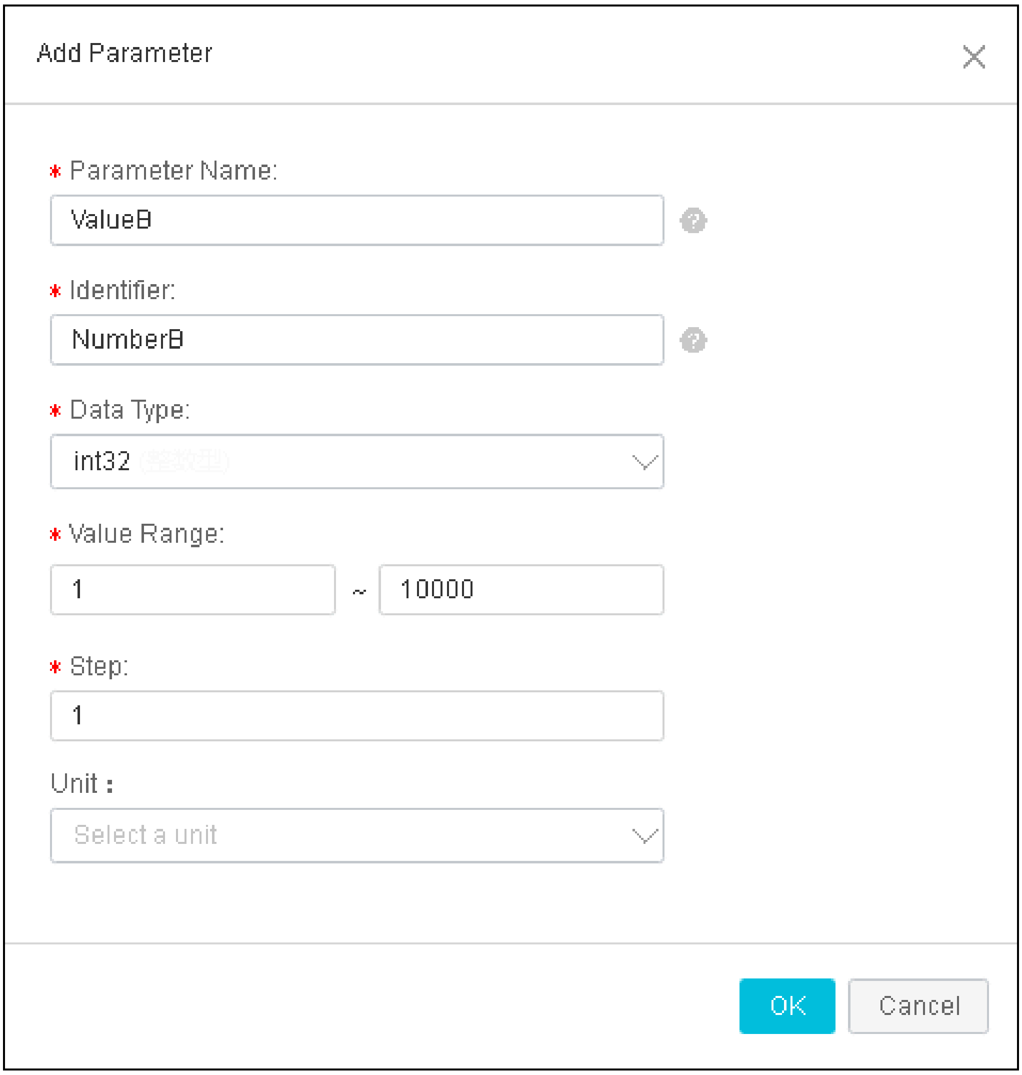
* Data Type: int32 (整数型)

* Value Range: 1 ~ 10000

* Step: 1

Unit : Select a unit

- **Value B is defined as follows:**



Add Parameter

* Parameter Name: ValueB

* Identifier: NumberB

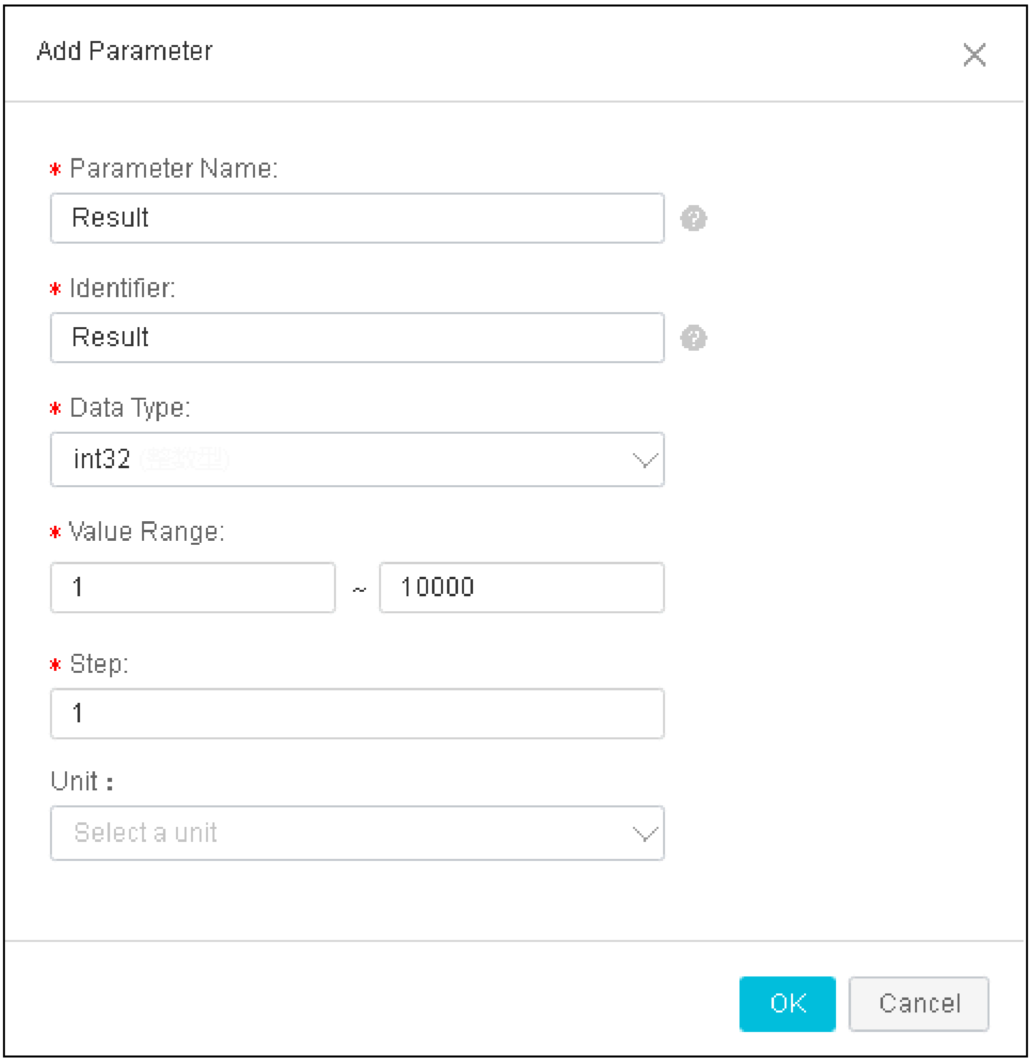
* Data Type: int32 (整数型)

* Value Range: 1 ~ 10000

* Step: 1

Unit : Select a unit

- **The output parameter indicates the calculation result.**



Add Parameter

* Parameter Name: Result

* Identifier: Result

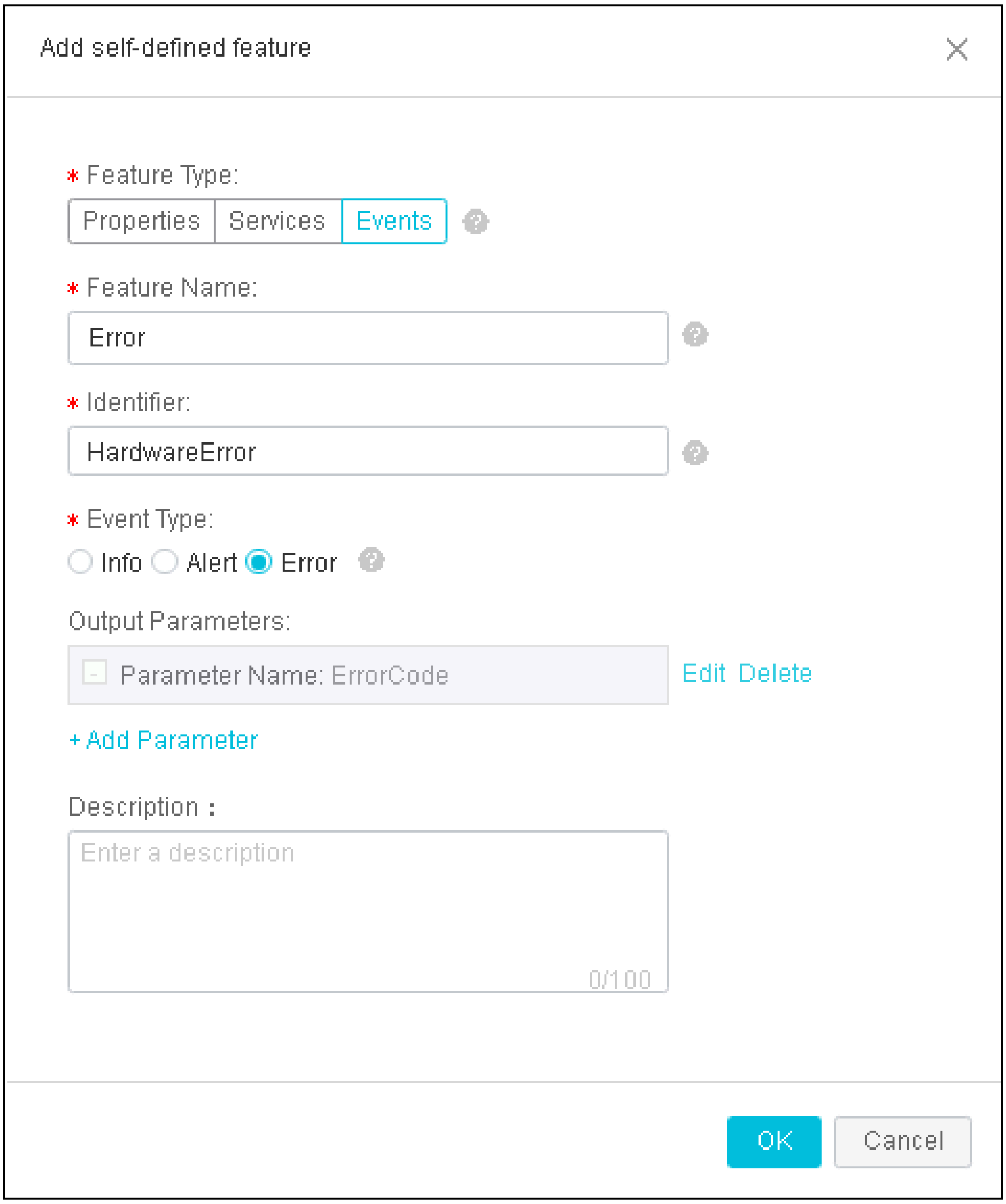
* Data Type: int32 (整数型)

* Value Range: 1 ~ 10000

* Step: 1

Unit : Select a unit

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



Add self-defined feature

* Feature Type: Properties | Services | [✓ Events]

* Feature Name: Error

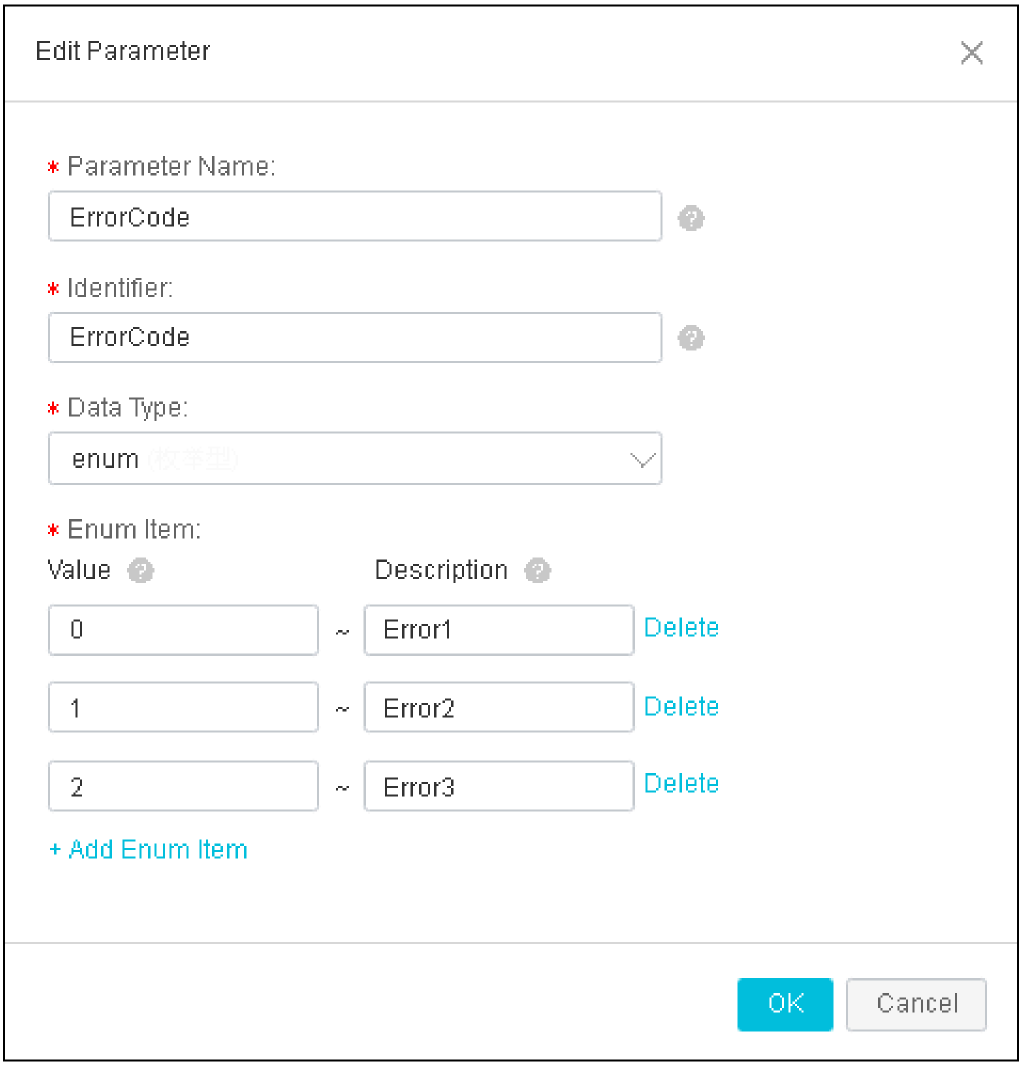
* Identifier: HardwareError

* Event Type: ○ Info ○ Alert ● Error

Output Parameters:

Parameter Name: ErrorCode Edit Delete

- The output parameter indicates the error code.



Edit Parameter

* Parameter Name: ErrorCode

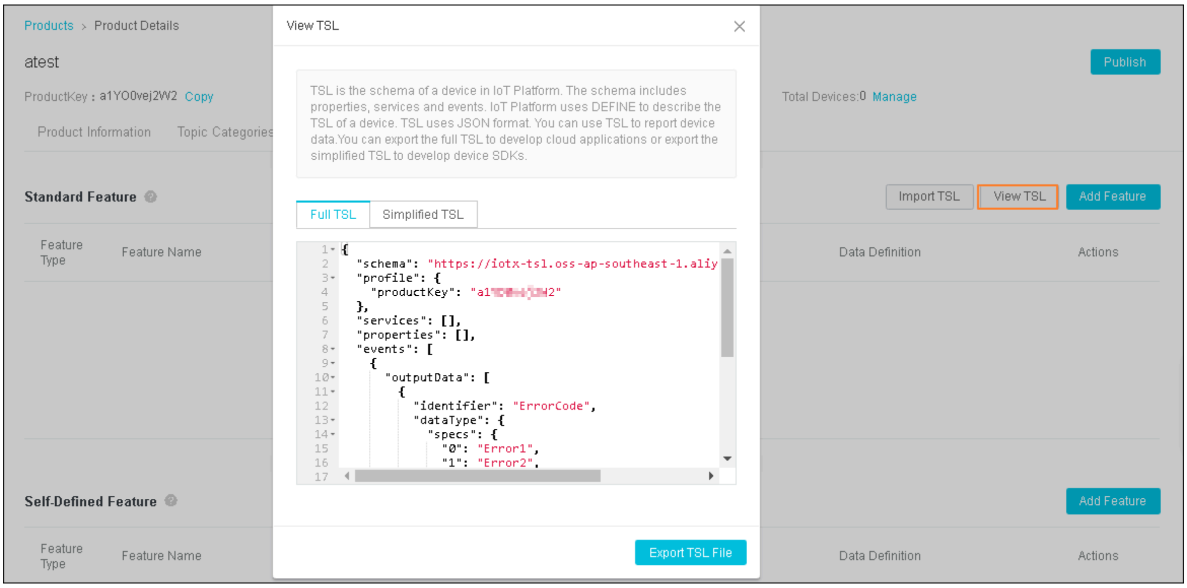
* Identifier: ErrorCode

* Data Type: enum (枚举型)

* Enum Item:

Value		Description	
0	~	Error1	Delete
1	~	Error2	Delete
2	~	Error3	Delete

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



atest

ProductKey : a1YO0vej2W2 Copy

Total Devices:0 Manage

Standard Feature

Feature Type	Feature Name	Data Definition	Actions

Self-Defined Feature

Feature Type	Feature Name	Data Definition	Actions

View TSL

TSL is the schema of a device in IoT Platform. The schema includes properties, services and events. IoT Platform uses DEFINE to describe the TSL of a device. TSL uses JSON format. You can use TSL to report device data.You can export the full TSL to develop cloud applications or export the simplified TSL to develop device SDKs.

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliy
  "profile": {
    "productKey": "a1[redacted]2"
  },
  "services": [],
  "properties": [],
  "events": [
    {
      "outputData": [
        {
          "identifier": "ErrorCode",
          "dataType": {
            "specs": {
              "0": "Error1",
              "1": "Error2",
```

What's next

#unique_6

1.3 Establish a connection between a device and IoT Platform

Alibaba Cloud IoT Platform provides device SDKs that allow devices to connect to IoT Platform. This article uses a sample program provided by IoT Platform to introduce how to connect the device to IoT Platform using the provided SDK.

Prerequisites

- The SDK used in this example is a C SDK for Linux system. We recommend that you develop this SDK on Ubuntu16.04 (64-bit)
- Software used in the development of the SDK: `make - 4 . 1`, `git - 2 . 7 . 4`, `gcc - 5 . 4 . 0`, `gcov - 5 . 4 . 0`, `lcov - 1 . 12`, `bash - 4 . 3 . 48`, `tar - 1 . 28`, and `mingw - 5 . 3 . 1` Using the following command to install the software:

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

Procedure

1. Log on to your Linux VM instance.
2. Download the C SDK 2.3.0.

```
wget   https :// github . com / aliyun / iotkit - embedded / archive
/ v2 . 3 . 0 . zip ? spm = a2c4g .  11186623 . 2 . 13 . 1f41492b5W
HpzV & file = v2 . 3 . 0 . zip
```

3. Use the unzip command to extract files from the package.
4. Open the demo program

```
vi   iotkit - embedded - 2 . 3 . 0 / examples / linkkit / linkkit_ex
ample_solo . c
```

5. Change the values of ProductKey, DeviceName, and DeviceSecret in the demo to be your device certificate information, and then save the file.

   See the following example:

```
//  for   demo   only
# define   PRODUCT_KE  Y       " a1I1nn8vPf  4 "
# define   DEVICE_NAM  E       " Light00 "
# define   DEVICE_SEC  RET     " n27gKXTxrU  x ********* QZEmoUX8Tc
 eM "
```

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

```
$  make   distclean
$  make
```

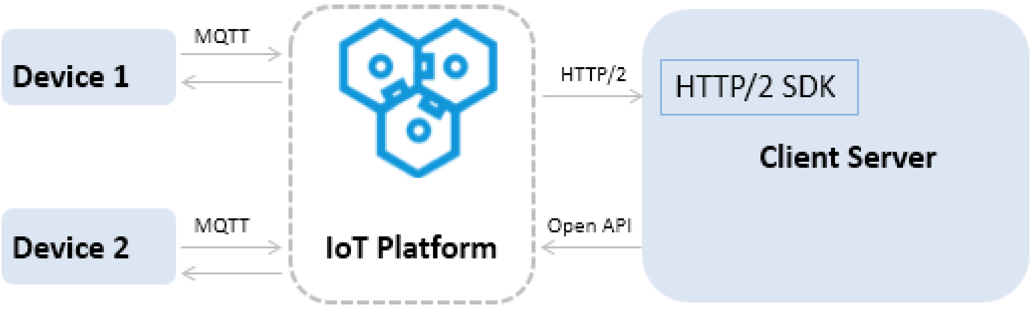
7. Run the sample program to connect the device to IoT Platform. In the IoT Platform console, you see that the device status is online, indicating that the device has been connected to IoT Platform successfully.

   Once the device has been connected to IoT Platform, it automatically report messages to IoT Platform. You see the device logs for message contents.

1.4 Subscribe to device messages from IoT Platform

After a device is connected to IoT Platform, the device directly reports data to IoT Platform. Then, the data is forwarded to your server over an HTTP/2 connection. This topic describes how to configure the service subscription function. You can connect your server to an HTTP/2 SDK to receive device data.

Context



Procedure

1. Configure the service subscription function for your product in the IoT Platform console.

   a) On the Products page, click View next to the target product.

   b) On the Product Details page, click Service Subscription > Set.

   c) Select the types of messages to which you want to subscribe, and click Save.

Message type	Description
Device Upstream Notification	Indicates the custom data and TSL model data that are reported by the device. The data includes property data , event data, property setting responses, and service call responses.

Message type	Description
Device Status Change Notifications	Indicates the notifications that are sent by the system when the status of a device changes. For example, the connection and disconnection notifications.
Device Changes Throughout Lifecycle	Indicates the notifications about device creation, deletion, disabling, and enabling.
Sub-Device Data Report Detected by Gateway	A gateway reports the information about the discovered sub-devices to IoT Platform. Make sure that the gateway has an application that can discover and report sub-device information.
Device Topological Relation Changes	Indicates notifications about the creation and removal of the topological relationships between a gateway and its sub-devices.

After you configure service subscription in the console, it takes approximately 1 minute for the settings to take effect.

2. Add dependencies.

If you use Apache Maven to manage Java projects, you must add the following dependencies to the pom.xml file.

Note:

Currently, only Java 8 and .NET SDKs are supported. For more information about SDK configuration, see #unique_9 or #unique_10.

```
< dependency >
    < groupId > com . aliyun . openservic  es </ groupId >
    < artifactId > iot - client - message </ artifactId >
    < version > 1 . 1 . 3 </ version >
</ dependency >

< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - core </ artifactId >
    < version > 3 . 7 . 1 </ version >
</ dependency >
```

3. Use the AccessKey information of your Alibaba Cloud account for identity authentication and connect the HTTP/2 SDK to IoT Platform.

```
// AccessKey   ID   of   your   Alibaba   Cloud   account
         String   accessKey  = " xxxxxxxxxx  xxxxx ";
        // AccessKey   Secret   of   your   Alibaba   Cloud
 account
         String   accessSecr  et  = " xxxxxxxxxx  xxxxx ";
        //  Region   ID   of   your   IoT   Platform   service
         String   regionId  = " cn - shanghai ";
```

```
       // User   ID   of   your   Alibaba   Cloud   account
        String   uid  = " xxxxxxxxxx  xx ";
       // Endpoint :  https ://${ uid }. iot - as - http2 .${
region }. aliyuncs . com
        String   endPoint  = " https ://" +  uid  + ". iot - as -
http2 ." +  regionId  + ". aliyuncs . com ";

       // Connection   configurat  ion
        Profile   profile  =  Profile . getAccessK  eyProfile (
endPoint ,  regionId ,  accessKey ,  accessSecr  et );

       // Construct   the   client
        MessageCli  ent   client  =  MessageCli  entFactory .
messageCli  ent ( profile );

       // Receive   data
        client . connect ( messageTok  en  -> {
            Message   m  =  messageTok  en . getMessage ();
            System . out . println (" receive   message   from  "
+  m );
            return   MessageCal  lback . Action . CommitSucc  ess ;
       });
```

Parameter description

Parameter	Description
accessKey	The AccessKey ID of your Alibaba Cloud account. To obtain the AccessKey ID, log on to the Alibaba Cloud console, hover over your account avatar, and click AccessKey. You are redirected to the Security Management page of the User Management console.
accessSecret	The AccessKey Secret of your Alibaba Cloud account. Obtain the AccessKey Secret in the same way you obtain the AccessKey ID.
uid	The account ID. To obtain the account ID, log on to the Alibaba Cloud console by using your Alibaba Cloud account, and click the account avatar. You are redirected to the Security Settings page of the Account Management console.
regionId	The region ID of your IoT Platform service. In the IoT Platform console, you can view the region in the left corner of the top navigation bar. For more information about regions, see Regions and zonesRegions and zones.

4. Verify that the HTTP/2 SDK can receive messages from the device.

   If messages can be received, you can obtain the following data from the message callback of the SDK.

Parameter	Description
`messageId`	The message ID generated by IoT Platform .
`topic`	The source topic of the message.
`payload`	The payload of the message. For more information, see #unique_11.
`generateTime`	The timestamp when the message was generated, in milliseconds.
`qos`	· 0: The message will be delivered only once. · 1: The message will be delivered at least once.

1.5 Send commands to devices

You can use applications in the cloud to call the `SetDeviceProperty` interface to send property setting commands to devices. This article introduces how to configure the device SDK to receive commands from IoT Platform.

Procedure

1. Import the SDK dependency into the maven project.

   The following examples show how to import the IoT Platform Java SDK dependency into the maven project.

```
<! --  https :// mvnreposit  ory . com / artifact / com . aliyun /
 aliyun - java - sdk - iot  -->
< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - iot </ artifactId >
    < version > 6 . 4 . 0 </ version >
</ dependency >
```

   Import the core module of the SDK.

```
< dependency >
    < groupId > com . aliyun </ groupId >
    < artifactId > aliyun - java - sdk - core </ artifactId >
    < version > 3 . 5 . 1 </ version >
```

```
</ dependency >
```

2. Initialize the SDK.

   The region ID in the endpoint must be the same as the region ID of the device. In the following example, the region ID is cn-shanghai.

```
 String   accessKey  = "< your   accessKey >";
 String   accessSecr  et  = "< your   accessSecr  et >";
 DefaultPro  file . addEndpoin  t (" cn - shanghai ", " cn -
 shanghai ", " Iot ", " iot . cn - shanghai . aliyuncs . com ");
 IClientPro  file   profile  =  DefaultPro  file . getProfile (" cn
 - shanghai ",  accessKey ,  accessSecr  et );
 DefaultAcs  Client   client  =  new   DefaultAcs  Client ( profile
 );
```

3. Call the SetDeviceProperty operation to send a property setting request to a device. In the following example, the value of the property LightSwitch is set to 1.

   Example:

```
 SetDeviceP  ropertyReq  uest   request  =  new   SetDeviceP
 ropertyReq  uest ();
 request . setProduct  Key (" a1I1xxxxPf  4 ");
 request . setDeviceN  ame (" Light001 ");
 JSONObject   itemJson  =  new   JSONObject ();
 itemJson . put (" LightSwitc  h ",  1 );
 request . setItems ( itemJson . toString ());

 try  {
     SetDeviceP  ropertyRes  ponse   response  =  client .
 getAcsResp  onse ( request );
     System . out . println ( response . getRequest  Id () + ",
 success : " +  response . getSuccess ());
} catch  ( ClientExce  ption   e ) {
     e . printStack  Trace ();
}
```

Note:

For more information about how to call the SetDeviceProperty operation, see SetDeviceProperty.

4. If the device has received the request, the log output is as follows:

```
[ inf ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1617 ):  Downstream
   Topic : '/ sys / a1I1nn8vPf  4 / Light001 / thing / service /
 property / set '
[ inf ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1618 ):  Downstream
 Payload :

< {
<     " method ": " thing . service . property . set ",
<     " id ": " 200864995 ",
<     " params ": {
<         " LightSwitc  h ":  1
<     },
<     " version ": " 1 . 0 . 0 "
< }

[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1623 ):          Packet
 Ident  :  00000000
[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1624 ):          Topic
 Length  :  52
[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1628 ):            Topic
   Name  : / sys / a1I1nn8vPf  4 / Light001 / thing / service /
 property / set
[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1631 ):      Payload
 Len / Room  :  101  /  109
[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1632 ):        Receive
 Buflen  :  166
[ dbg ]  iotx_mc_ha  ndle_recv_  PUBLISH ( 1643 ):  delivering   msg
  ...
[ dbg ]  iotx_mc_de  liver_mess  age ( 1344 ):  topic   be
 matched
[ inf ]  dm_msg_pro  c_thing_se  rvice_prop  erty_set ( 134 ):
 thing / service / property / set
[ dbg ]  dm_msg_req  uest_parse ( 130 ):  Current   Request
 Message   ID :  200864995
[ dbg ]  dm_msg_req  uest_parse ( 131 ):  Current   Request
 Message   Version :  1 . 0 . 0
[ dbg ]  dm_msg_req  uest_parse ( 132 ):  Current   Request
 Message   Method :  thing . service . property . set
[ dbg ]  dm_msg_req  uest_parse ( 133 ):  Current   Request
 Message   Params : {" LightSwitc  h ": 1 }
[ dbg ]  dm_ipc_msg  _insert ( 87 ):  dm   msg   list   size :  0 ,
 max   size :  50
[ inf ]  dm_msg_res  ponse ( 262 ):  Send   URI : / sys /
 a1I1nn8vPf  4 / Light001 / thing / service / property / set_reply
 ,  Payload : {" id ":" 200864995 "," code ": 200 ," data ":{}}
[ inf ]  MQTTPublis  h ( 515 ):  Upstream   Topic : '/ sys /
 a1I1nn8vPf  4 / Light001 / thing / service / property / set_reply '
[ inf ]  MQTTPublis  h ( 516 ):  Upstream   Payload :

> {
>     " id ": " 200864995 ",
>     " code ":  200 ,
>     " data ": {
>     }
> }

[ inf ]  dm_client_  publish ( 121 ):  Publish   Result :  0
[ inf ]  _iotx_link  kit_event_  callback ( 223 ):  Receive
 Message   Type :  15
[ inf ]  _iotx_link  kit_event_  callback ( 225 ):  Receive
 Message : {" devid ": 0 ," payload ":{" LightSwitc  h ": 1 }}
[ dbg ]  _iotx_link  kit_event_  callback ( 403 ):  Current   Devid
 :  0
[ dbg ]  _iotx_link  kit_event_  callback ( 404 ):  Current
 Payload : {" LightSwitc  h ": 1 }
 user_prope  rty_set_ev  ent_handle  r .  160 :  Property   Set
 Received ,  Devid :  0 ,  Request : {" LightSwitc  h ": 1 }
```

2 Connect to IoT Platform using MQTT.fx

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

Prerequisites

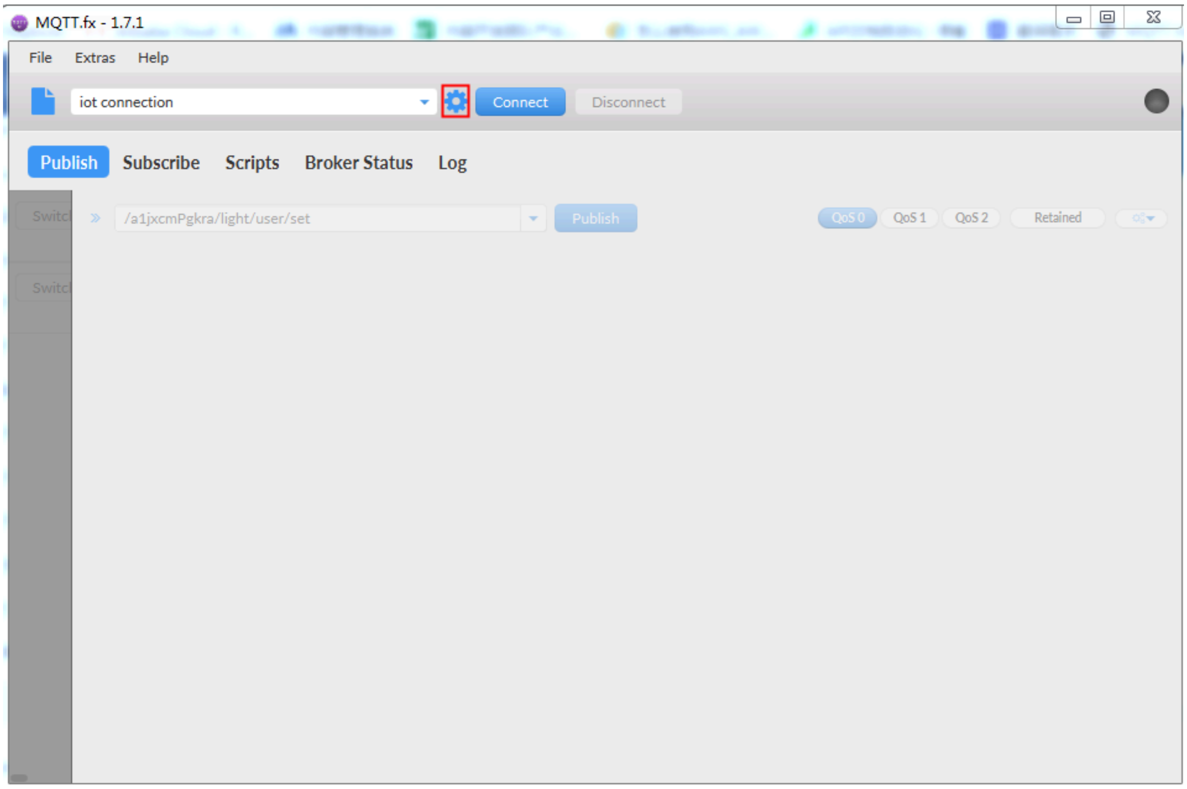
You have created products and devices in the IoT Platform console, and have got the ProductKey, DeviceName, and DeviceSecret of the devices. When you set the connection parameters for MQTT.fx, you will use the values of the ProductKey, DeviceName, and DeviceSecret. See #unique_15, #unique_16, and #unique_17for help when creating products and devices.

Procedure

1. Download and install the MQTT.fx software.

   Download the MQTT.fx software for Windows from MQTT.fx website.

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



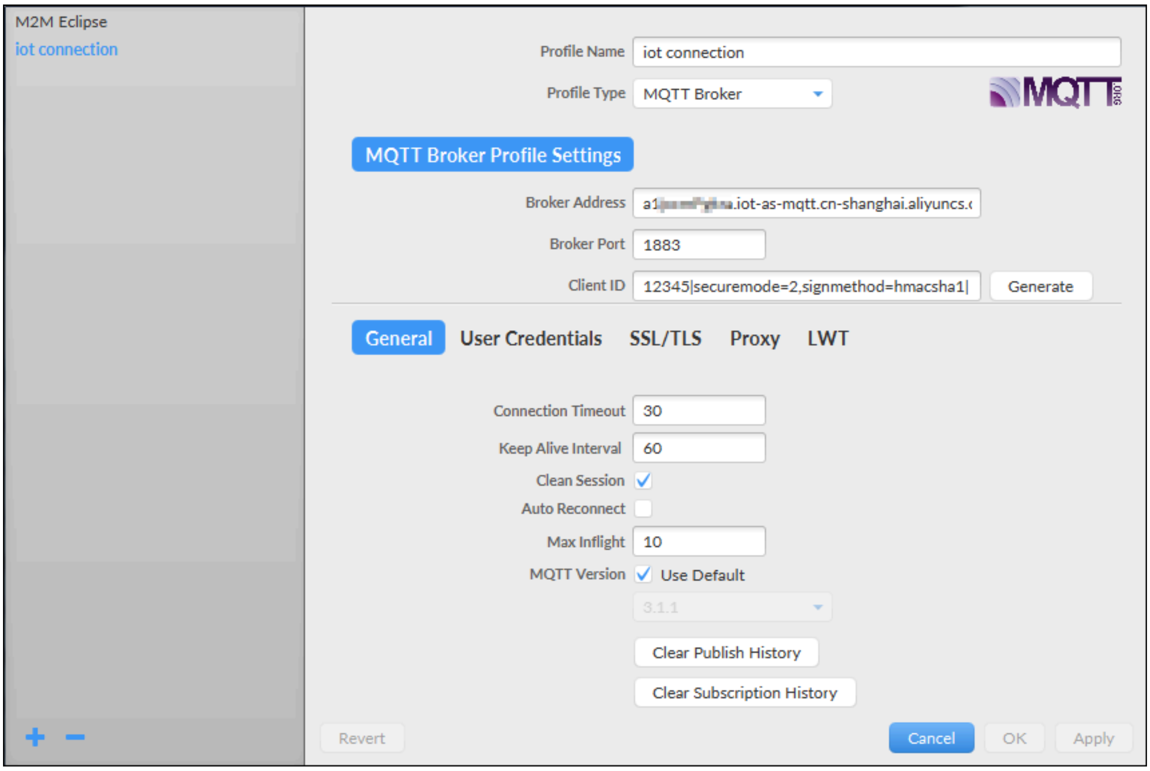
3. Set the connection parameters.

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

   The procedure is as follows:

   a. Enter basic information. See the following table for parameter descriptions.

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



M2M Eclipse
iot connection

Profile Name: iot connection
Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: a1[redacted].iot-as-mqtt.cn-shanghai.aliyuncs.c
Broker Port: 1883
Client ID: 12345|securemode=2,signmethod=hmacsha1|

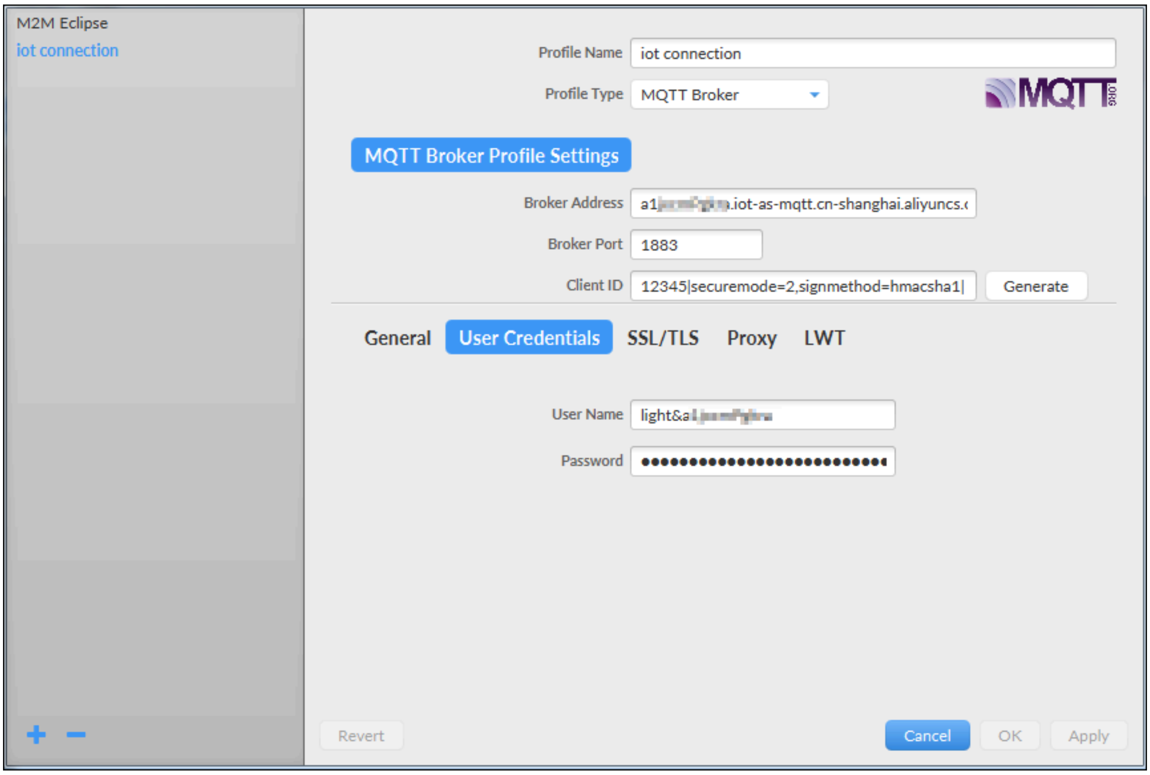
General | User Credentials | SSL/TLS | Proxy | LWT

Connection Timeout: 30
Keep Alive Interval: 60
Clean Session: ☑
Auto Reconnect: □
Max Inflight: 10
MQTT Version: ☑ Use Default
3.1.1

Parameter	Description
Profile Name	Enter a custom profile name.
Profile Type	Select MQTT Broker.
Broker Address	Enter the connection domain in the format of `${YourProductKey}.iot-as-mqtt.${region}.aliyuncs.com`. In this format, variable *`${region}`* indicates the region ID of your IoT Platform service region. For region IDs, see Regions and zones Regions and zones. Example: alPUPCoxxxx.iot-as-mqtt.cn-shanghai.aliyuncs.com.
Broker Port	Set to `1883`.

Parameter	Description
Client ID	Enter a value in the format of `${clientId}\|securemode=3,signmethod=hmacsha1\|`. Example: `12345 \| securemode = 3 , signmethod = hmacsha1 \|`. The parameters are described as follows: · *${clientId}* is a custom client ID. It can be any value within 64 characters. We recommend that you use the MAC address or SN code of the device as the value of clientId. · *securemode* is the security mode of the connection. If you use the TCP mode, set it as securemode=3; if you use the TLS mode, set it as securemode=2. · *signmethod* is the signature method that you want to use. IoT Platform supports `hmacmd5` and `hmacsha1` . Note: Do not click Generate after you enter the Client ID information.

b. Click User Credentials, and enter your User Name and Password.



M2M Eclipse
iot connection

Profile Name: iot connection
Profile Type: MQTT Broker

MQTT Broker Profile Settings

Broker Address: a1[redacted].iot-as-mqtt.cn-shanghai.aliyuncs.c
Broker Port: 1883
Client ID: 12345|securemode=2,signmethod=hmacsha1|

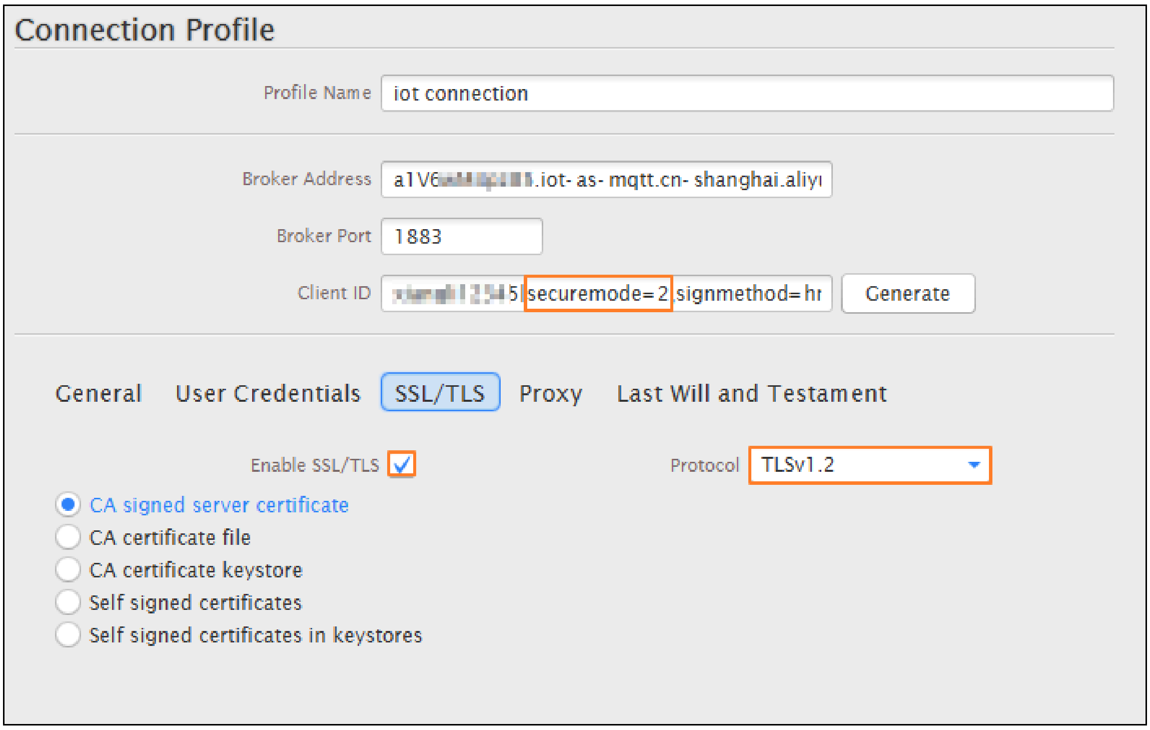
User Name: light&a[redacted]
Password: ●●●●●●●●●●●●●●●●●●●●●●●●●●

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

Parameter	Description
Password	You must enter an encrypted value of the input parameters. · IoT Platform provides a Password Generator for you to generate one easily. Parameters in the password generator: - productKey: The unique identifier of the product to which the device belongs. You can view this information on the device details page in the console. - deviceName: The name of the device. You can view this information on the device details page in the console. - deviceSecret: The device secret. You can view this information on the device details page in the console. - timestamp: (Optional) Timestamp of the current system time. - clientId: The custom client ID, which must be the same as the value of *${clientId}* in Client ID. - method: The signature algorithm, which must be the same as the value of *signmethod* in Client ID. · You can also encrypt a password by yourself. Generate a password manually: A. Sort and join the parameters. Sort and join the parameters `clientId` , `deviceName` , `productKey` , and `timestamp` in a lexicographical order. (If you have not set a timestamp, do not include timestamp in the string.) Joint string example: `clientId12345deviceNamedeviceproductKeyfOAt5H5TOWF` B. Encrypt. Use the `deviceSecret` of the device as the secret key to encrypt the joint string by the signature algorithm defined in Client ID. Suppose the `deviceSecret` of the device is abc123, the encryption format is `hmacsha1(abc123,clientId12345deviceNamedeviceproductKeyfOAt5H5TOWF)`.

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

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



Connection Profile

Profile Name: iot connection

Broker Address: a1V6[redacted].iot-as-mqtt.cn-shanghai.aliyı

Broker Port: 1883

Client ID: [redacted]5|securemode=2,signmethod=hı

Enable SSL/TLS ☑

Protocol: TLSv1.2

● CA signed server certificate

○ CA certificate file

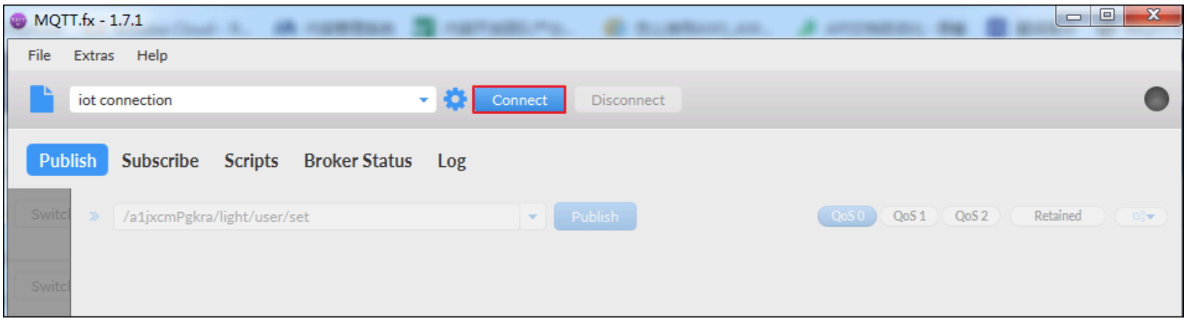
○ CA certificate keystore

○ Self signed certificates

○ Self signed certificates in keystores

d. Enter all the required information, and then click OK.

4. Click Connect to connect to IoT Platform.



iot connection

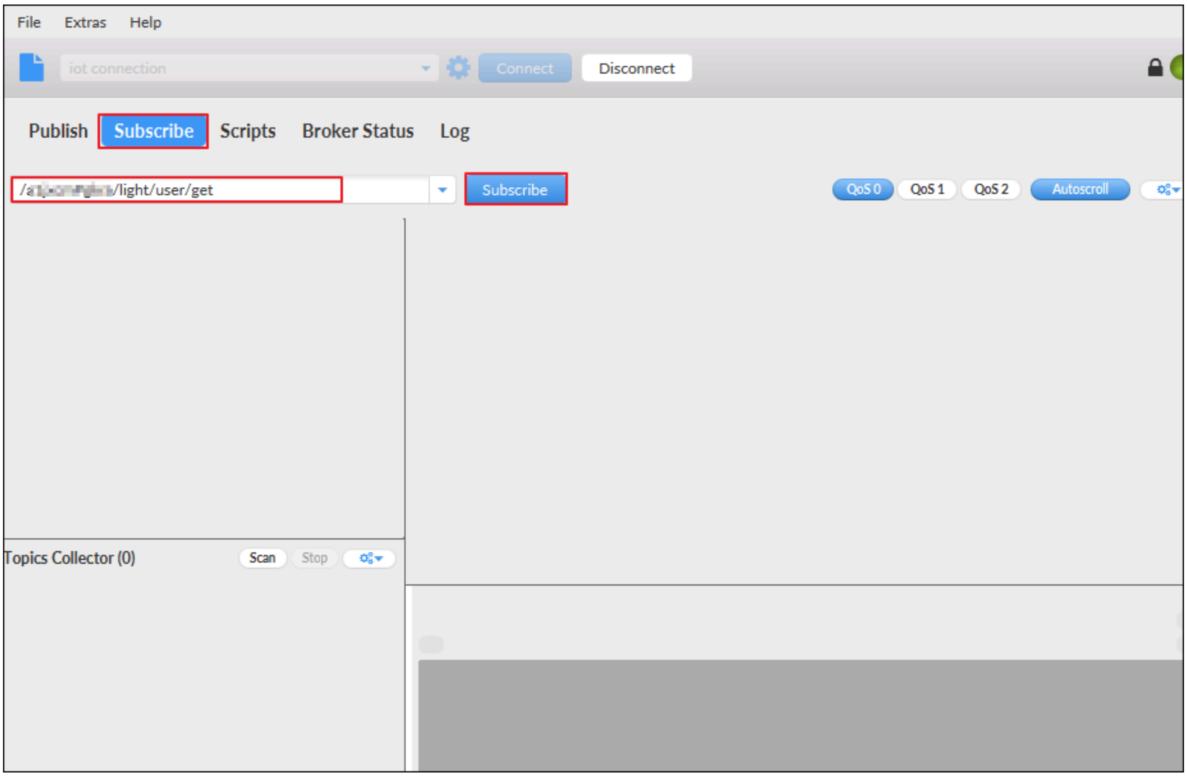
/a1jxcmPgkra/light/user/set

Message communication test

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

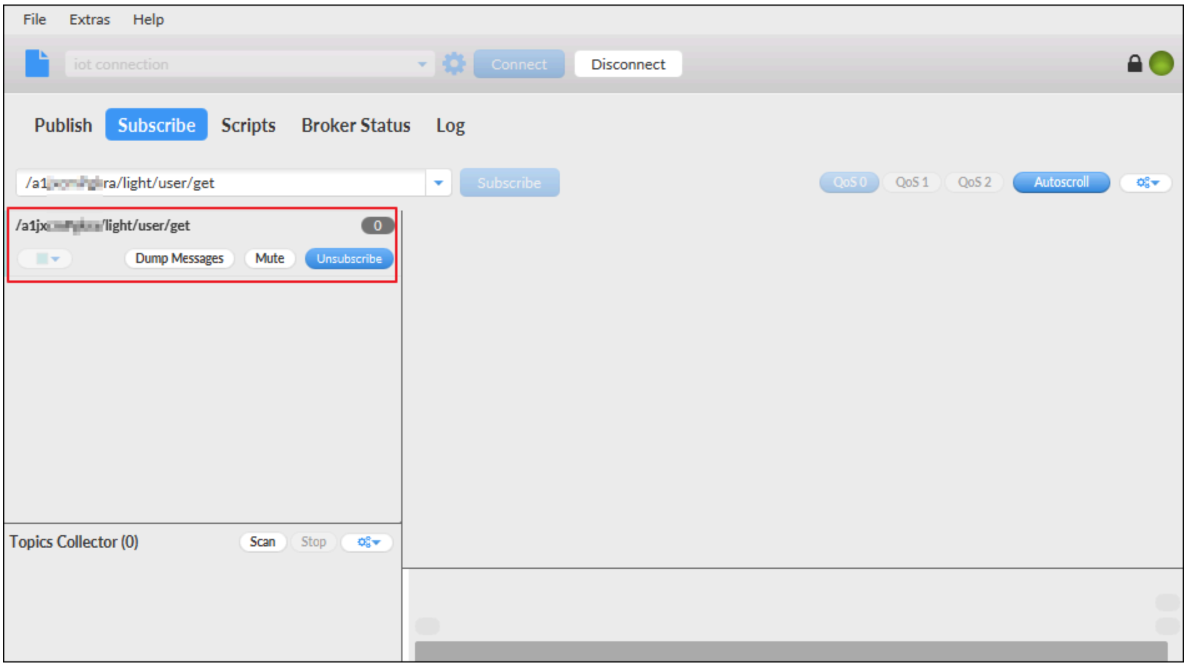
1. In MQTT.fx, click Subscribe.

2. Enter a topic of the device, and then click Subscribe.



/a1[redacted]/light/user/get

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

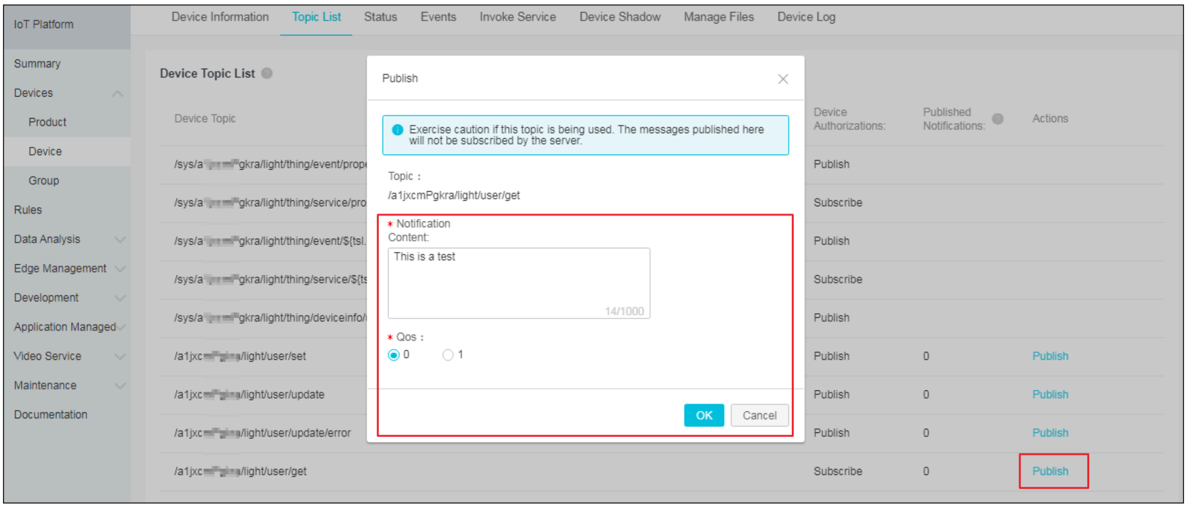


/a1[redacted]ra/light/user/get

/a1jx[redacted]/light/user/get

3. In the IoT Platform console, in the Topic List of the Device Details page, click the Publish button of the topic that you have subscribed to.

4. Enter message content, and click OK.



Publish

Exercise caution if this topic is being used. The messages published here will not be subscribed by the server.

Topic：
/a1jxcmPgkra/light/user/get

* Notification Content:

This is a test

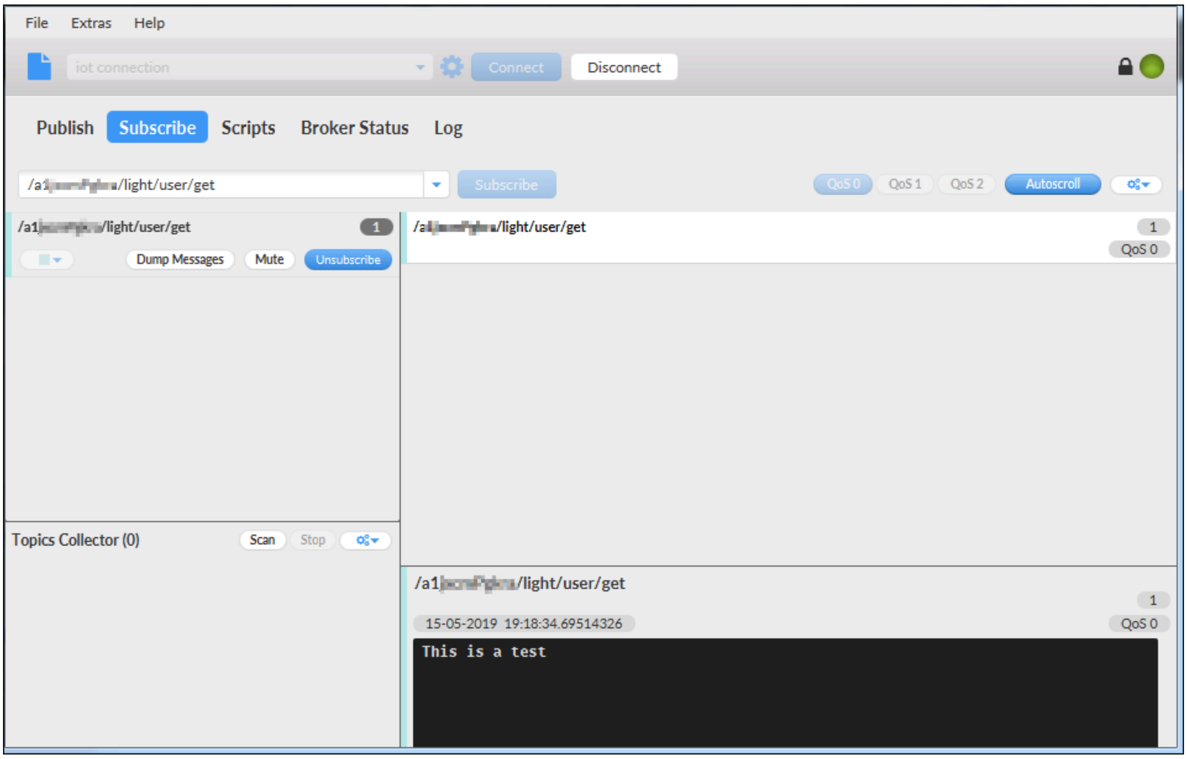
14/1000

* Qos：
● 0 ○ 1

OK Cancel

Device Topic	Device Authorizations:	Published Notifications:	Actions
/sys/a[redacted]gkra/light/thing/event/prope	Publish		
/sys/a[redacted]gkra/light/thing/service/pro	Subscribe		
/sys/a[redacted]gkra/light/thing/event/${tsl.	Publish		
/sys/a[redacted]gkra/light/thing/service/${ts	Subscribe		
/sys/a[redacted]gkra/light/thing/deviceinfo/	Publish		
/a1jxc[redacted]/light/user/set	Publish	0	Publish
/a1jxc[redacted]/light/user/update	Publish	0	Publish
/a1jxc[redacted]/light/user/update/error	Publish	0	Publish
/a1jxc[redacted]/light/user/get	Subscribe	0	Publish

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



/a[redacted]/light/user/get

/a1[redacted]/light/user/get 1

/a[redacted]/light/user/get 1 QoS 0

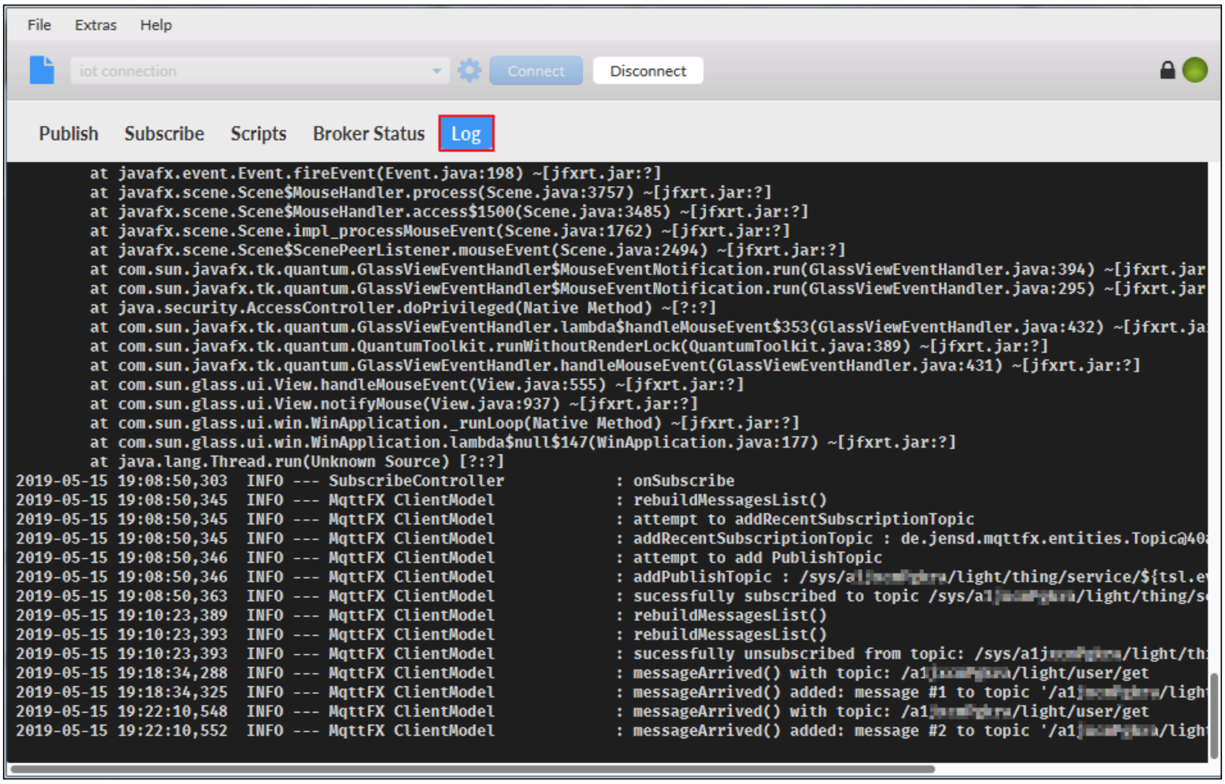
/a1[redacted]/light/user/get 1

15-05-2019 19:18:34.69514326 QoS 0

```
This is a test
```

View logs

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



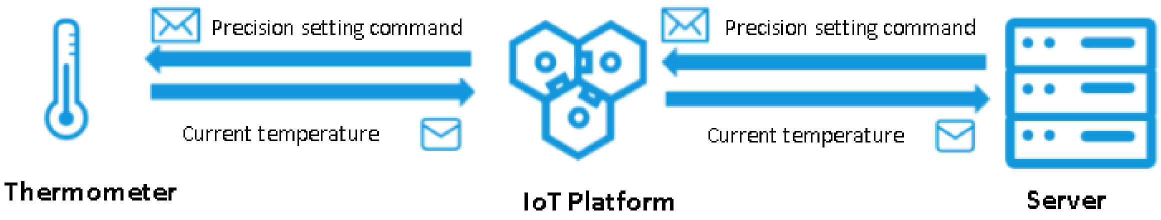
```
        at javafx.event.Event.fireEvent(Event.java:198) ~[jfxrt.jar:?]
        at javafx.scene.Scene$MouseHandler.process(Scene.java:3757) ~[jfxrt.jar:?]
        at javafx.scene.Scene$MouseHandler.access$1500(Scene.java:3485) ~[jfxrt.jar:?]
        at javafx.scene.Scene.impl_processMouseEvent(Scene.java:1762) ~[jfxrt.jar:?]
        at javafx.scene.Scene$ScenePeerListener.mouseEvent(Scene.java:2494) ~[jfxrt.jar:?]
        at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:394) ~[jfxrt.jar
        at com.sun.javafx.tk.quantum.GlassViewEventHandler$MouseEventNotification.run(GlassViewEventHandler.java:295) ~[jfxrt.jar
        at java.security.AccessController.doPrivileged(Native Method) ~[?:?]
        at com.sun.javafx.tk.quantum.GlassViewEventHandler.lambda$handleMouseEvent$353(GlassViewEventHandler.java:432) ~[jfxrt.ja
        at com.sun.javafx.tk.quantum.QuantumToolkit.runWithoutRenderLock(QuantumToolkit.java:389) ~[jfxrt.jar:?]
        at com.sun.javafx.tk.quantum.GlassViewEventHandler.handleMouseEvent(GlassViewEventHandler.java:431) ~[jfxrt.jar:?]
        at com.sun.glass.ui.View.handleMouseEvent(View.java:555) ~[jfxrt.jar:?]
        at com.sun.glass.ui.View.notifyMouse(View.java:937) ~[jfxrt.jar:?]
        at com.sun.glass.ui.win.WinApplication._runLoop(Native Method) ~[jfxrt.jar:?]
        at com.sun.glass.ui.win.WinApplication.lambda$null$147(WinApplication.java:177) ~[jfxrt.jar:?]
        at java.lang.Thread.run(Unknown Source) [?:?]
2019-05-15 19:08:50,303  INFO --- SubscribeController            : onSubscribe
2019-05-15 19:08:50,345  INFO --- MqttFX ClientModel             : rebuildMessagesList()
2019-05-15 19:08:50,345  INFO --- MqttFX ClientModel             : attempt to addRecentSubscriptionTopic
2019-05-15 19:08:50,345  INFO --- MqttFX ClientModel             : addRecentSubscriptionTopic : de.jensd.mqttfx.entities.Topic@40
2019-05-15 19:08:50,346  INFO --- MqttFX ClientModel             : attempt to add PublishTopic
2019-05-15 19:08:50,346  INFO --- MqttFX ClientModel             : addPublishTopic : /sys/a[redacted]/light/thing/service/${tsl.e
2019-05-15 19:08:50,363  INFO --- MqttFX ClientModel             : sucessfully subscribed to topic /sys/a1[redacted]/light/thing/s
2019-05-15 19:10:23,389  INFO --- MqttFX ClientModel             : rebuildMessagesList()
2019-05-15 19:10:23,393  INFO --- MqttFX ClientModel             : rebuildMessagesList()
2019-05-15 19:10:23,393  INFO --- MqttFX ClientModel             : sucessfully unsubscribed from topic: /sys/a1j[redacted]/light/th
2019-05-15 19:18:34,288  INFO --- MqttFX ClientModel             : messageArrived() with topic: /a1[redacted]/light/user/get
2019-05-15 19:18:34,325  INFO --- MqttFX ClientModel             : messageArrived() added: message #1 to topic '/a1[redacted]/light
2019-05-15 19:22:10,548  INFO --- MqttFX ClientModel             : messageArrived() with topic: /a1[redacted]/light/user/get
2019-05-15 19:22:10,552  INFO --- MqttFX ClientModel             : messageArrived() added: message #2 to topic '/a1[redacted]/light
```

3 Use custom topics for communication

You can define custom topic categories in IoT Platform. Then, a device can send messages to a custom topic, and your server can receive the messages through the HTTP/2 SDK. Your server can also call the API operation #unique_19 to send commands to the device.

Scenario

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



Prepare the development environment

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

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

```
< dependenci  es >
 < dependency >
     < groupId > com . aliyun . alink . linksdk </ groupId >
     < artifactId > iot - linkkit - java </ artifactId >
     < version > 1 . 2 . 0 . 1 </ version >
     < scope > compile </ scope >
 </ dependency >
 <  dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - core </ artifactId >
      < version > 3 . 7 . 1 </ version >
  </ dependency >
  < dependency >
      < groupId > com . aliyun </ groupId >
      < artifactId > aliyun - java - sdk - iot </ artifactId >
      < version > 6 . 9 . 0 </ version >
  </ dependency >
  < dependency >
```

```
     < groupId > com . aliyun . openservic  es </ groupId >
     < artifactId > iot - client - message </ artifactId >
     < version > 1 . 1 . 2 </ version >
</ dependency >
</ dependenci  es >
```

Create a product and a device

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

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

   For more information, see #unique_15.
4. After the product is created, find the product and click View.
5. On the Topic Categories tab of the Product Details page, add custom topic categories.

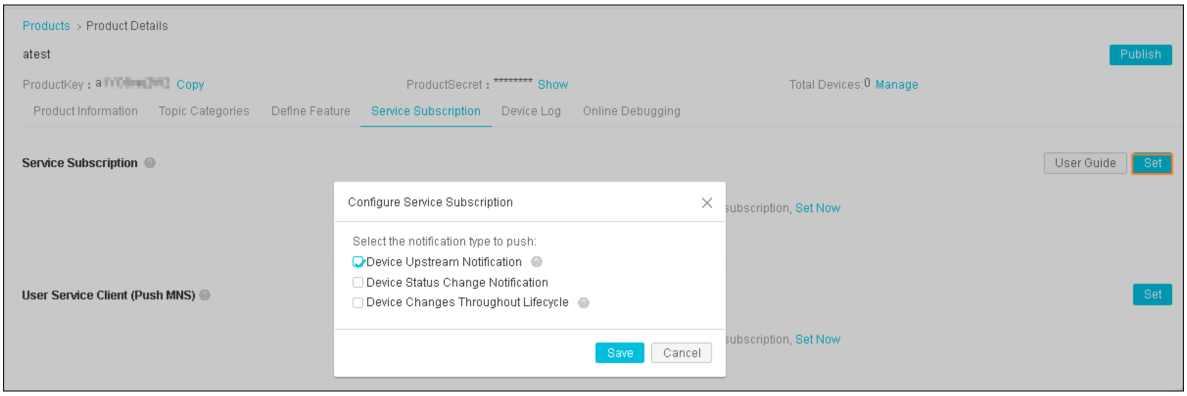
   For more information, see #unique_20.

   In this example, add the following two topic categories:

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

6. On the Service Subscription tab, set the type of messages to be pushed to the HTTP/2 SDK to Device Upstream Notification.

   For more information, see #unique_9.



Products > Product Details

atest

ProductKey : a[redacted] Copy

ProductSecret : ******** Show

Total Devices:0 Manage

Product Information | Topic Categories | Define Feature | Service Subscription | Device Log | Online Debugging

Service Subscription

User Guide | Set

Configure Service Subscription

Select the notification type to push:

☑ Device Upstream Notification

□ Device Status Change Notification

□ Device Changes Throughout Lifecycle

Save | Cancel

ubscription, Set Now

User Service Client (Push MNS)

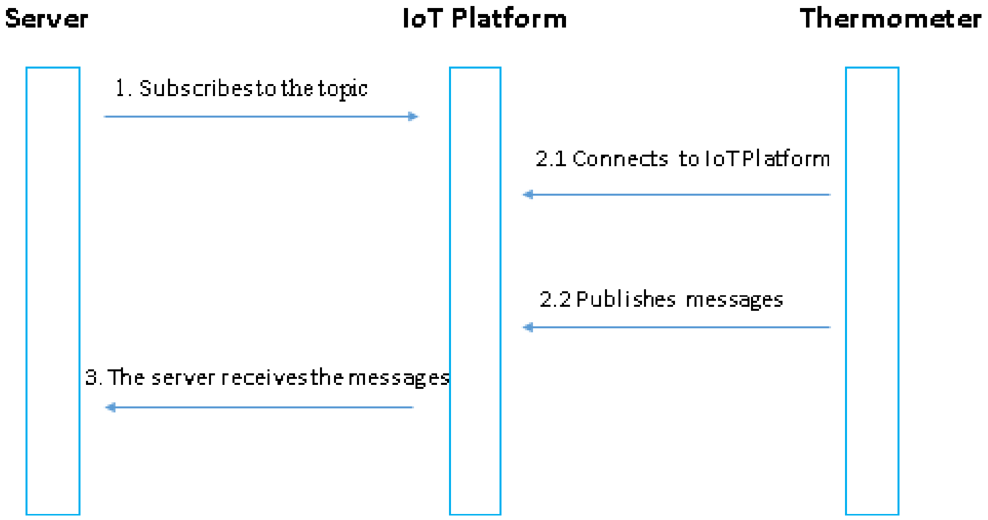
Set

ubscription, Set Now

7. In the left-side navigation pane, choose Devices and add a thermometer device under the thermometer product that has been created. For more information, see #unique_16.

The server receives messages from the device

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


Server
IoT Platform
Thermometer
1. Subscribes to the topic
2.1 Connects to IoT Platform
2.2 Publishes messages
3. The server receives the messages

- Configure the HTTP/2 SDK, which will be installed in the server, and set a callback for the specified topic.

  After service subscription is configured, the server listens to all the messages sent by all the devices under the product.

  - Connect the HTTP/2 SDK to IoT Platform.

```
// Configure identity verificati on informatio n .
 String accessKey = " The AccessKey ID of your
 Alibaba Cloud account ";
 String accessSecr et = " The AccessKey Secret of your
   Alibaba Cloud account ";
 String uid = " Your Alibaba Cloud account ID ";
 String regionId = " The ID of the region where the
   device is located ";
 String productKey = " The key of the product ";
 String deviceName = " The name of the device ";
 String endPoint = " https ://" + uid + ". iot - as - http2
 ." + regionId + ". aliyuncs . com ";
// Configure the connection .
 Profile profile = Profile . getAccessK eyProfile ( endPoint
 , regionId , accessKey , accessSecr et );

// Construct a client .
 MessageCli ent client = MessageCli entFactory . messageCli
 ent ( profile );

// Receive the message .
 client . connect ( new MessageCal lback () {
    @ Override
     public Action consume ( MessageTok en messageTok en )
 {
         Message m = messageTok en . getMessage ();
         System . out . println ("\ ntopic =" + m . getTopic ());
         System . out . println (" payload =" + new String ( m
 . getPayload ()));
         System . out . println (" generateTi me =" + m .
 getGenerat eTime ());
        // CommitSucc ess indicates that the current
   message has been consumed . In this case , IoT
 Platform will delete the message . If the message
   is not marked with CommitSucc ess , IoT Platform
 will retain the message until it expires .
         return MessageCal lback . Action . CommitSucc ess ;
     }
});
```

  - Configure the message receiving interface.

    In this example, the server subscribes to the topic /$ {productKey}/$ { deviceName}/user/devmsg. Therefore, you need to set a callback for this topic.

```
// Define the callback .
 MessageCal lback messageCal lback = new MessageCal lback
 () {
// Obtain and display the message . You can set
 the required action for the server here .
             @ Override
```

```
            public   Action   consume ( MessageTok  en
messageTok  en ) {
                Message   m  =  messageTok  en . getMessage ();
                log . info (" receive  : " +  new   String (
messageTok  en . getMessage (). getPayload ()));
                System . out . println ( messageTok  en .
getMessage ());
                return   MessageCal  lback . Action . CommitSucc
ess ;
           }
       };
       //  Register   the   callback .
        client . setMessage  Listener ("/" +  productKey  + "/"
+  deviceName  + "/ user / devmsg ",  messageCal  lback );
```

For more information, see #unique_9.

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

```
final   String   productKey  = " XXXXXX ";
final   String   deviceName  = " XXXXXX ";
final   String   deviceSecr  et  = " XXXXXXXXX ";
final   String   region  = " XXXXXX ";
```

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

```
 LinkKitIni  tParams   params  =  new   LinkKitIni  tParams ();
//  Configure   MQTT   connection   informatio  n .  Link   Kit
 uses   MQTT   as   the   underlying   protocol .
 IoTMqttCli  entConfig   config  =  new   IoTMqttCli  entConfig
 ();
 config . productKey  =  productKey ;
 config . deviceName  =  deviceName ;
 config . deviceSecr  et  =  deviceSecr  et ;
 config . channelHos  t  =  productKey  + ". iot - as - mqtt ." +
 region  + ". aliyuncs . com : 1883 ";
//  Configure   device   informatio  n .
 DeviceInfo   deviceInfo  =  new   DeviceInfo ();
 deviceInfo . productKey  =  productKey ;
 deviceInfo . deviceName  =  deviceName ;
 deviceInfo . deviceSecr  et  =  deviceSecr  et ;
//  Register   the   initial   device   status .
 Map < String ,  ValueWrapp  er >  propertyVa  lues  =  new
 HashMap < String ,  ValueWrapp  er >();

 params . mqttClient  Config  =  config ;
 params . deviceInfo  =  deviceInfo ;
 params . propertyVa  lues  =  propertyVa  lues ;
```

  - Initialize the connection.

```
//  Initialize   the   connection   and   set   the   callback
   that   is   called   when   the   initializa  tion   is
 successful .
 LinkKit . getInstanc  e (). init ( params ,  new   ILinkKitCo
 nnectListe  ner () {
     @ Override
      public   void   onError ( AError   aError ) {
         System . out . println (" Init   error :" +  aError );
     }

     //  Set   the   callback   that   is   called   when   the
 initializa  tion   is   successful .
     @ Override
      public   void   onInitDone ( InitResult   initResult ) {
          System . out . println (" Init   done :" +  initResult
 );
     }
 });
```

- **Send a message from the device.**

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

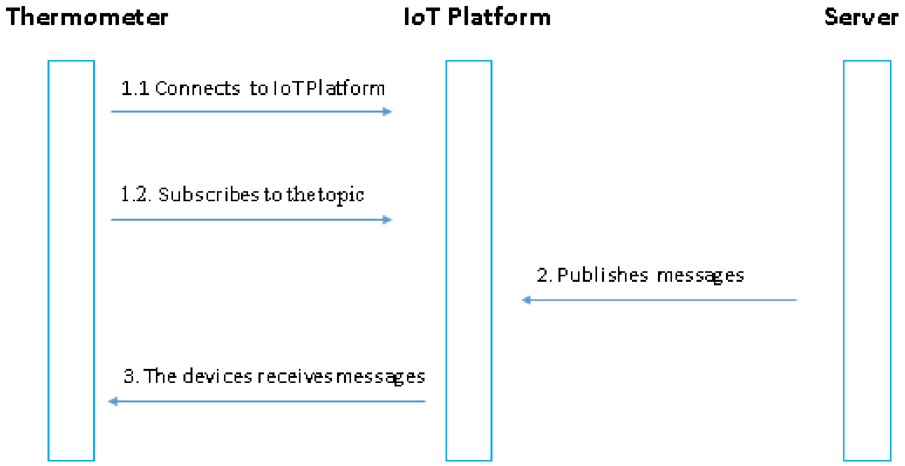
```
@ Override
  public   void   onInitDone ( InitResult   initResult ) {
     //  Set   the   topic   to   which   the   message   is
 published   and   the   message   content .
      MqttPublis  hRequest   request   =  new   MqttPublis
 hRequest ();
      request . topic   = "/" +  productKey  + "/" +  deviceName  +
 "/ user / devmsg ";
      request . qos   =  0 ;
      request . payloadObj   = "{\" temperatur  e \": 35 . 0 , \"
 time \":\" sometime \"}";
     //  Publish   the   message   and   set   the   callback
 that   is   called   when   the   message   is   published .
      LinkKit . getInstanc  e (). publish ( request ,  new
 IConnectSe  ndListener () {
         @ Override
          public   void   onResponse ( ARequest   aRequest ,
 AResponse   aResponse ) {
              System . out . println (" onResponse :" +  aResponse
 . getData ());
         }

         @ Override
          public   void   onFailure ( ARequest   aRequest ,
 AError   aError ) {
              System . out . println (" onFailure :" +  aError .
 getCode () +  aError . getMsg ());
         }
     });
 }
```

The server receives the following message:

```
 Message
{ payload ={" temperatur  e ": 35 . 0 , " time ":" sometime "},
 topic ='/ a1uzcH0 ****/ device1 / user / devmsg ',
 messageId =' 1131755639  450642944 ',
 qos = 0 ,
generateTi  me = 1558666546  105 }
```

The server sends messages to the device

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


Thermometer
IoT Platform
Server
1.1 Connects to IoTPlatform
1.2. Subscribes to the topic
2. Publishes messages
3. The devices receives messages

- Configure the device SDK to subscribe to a topic.

  For more information about how to configure device authentication information , set connection initialization parameters, and initialize the connection, see the device SDK configuration in the previous section.

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

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

```
//  Set   the   callback   that   is   called   when   the
 initializa  tion   is   successful .
@ Override
 public   void   onInitDone ( InitResult   initResult ) {
    //  Set   the   topic   to   which   the   device   subscribes .
     MqttSubscr  ibeRequest   request  =  new   MqttSubscr
 ibeRequest ();
     request . topic   = "/" +  productKey  + "/" +  deviceName  + "/
 user / cloudmsg ";
     request . isSubscrib  e  =  true ;
    //  Send   a   subscripti  on   request   and   set   the
 callbacks   that   are   called   when   the   subscripti  on
 succeeds   and   fails ,  respective  ly .
     LinkKit . getInstanc  e (). subscribe ( request ,  new
 IConnectSu  bscribeLis  tener () {
        @ Override
         public   void   onSuccess () {
             System . out . println ("");
        }
        @ Override
         public   void   onFailure ( AError   aError ) {

        }
    });

    //  Set   the   listener   for   listening   to   subscribed
 messages .
     IConnectNo  tifyListen  er   notifyList  ener  =  new
 IConnectNo  tifyListen  er () {
        //  Define   the   callback   that   is   called   when   a
   subscribed   message   is   received .
        @ Override
         public   void   onNotify ( String   connectId ,  String
 topic ,  AMessage   aMessage ) {
             System . out . println (
                " received   message   from   " +  topic  + ":" +
 new   String (( byte []) aMessage . getData ()));
        }

        @ Override
         public   boolean   shouldHand  le ( String   s ,  String
 s1 ) {
             return   false ;
        }

        @ Override
         public   void   onConnectS  tateChange ( String   s ,
 ConnectSta  te   connectSta  te ) {

        }
    };
     LinkKit . getInstanc  e (). registerOn  NotifyList  ener (
 notifyList  ener );
}
```

- **Configure the IoT SDK to call the #unique_19 operation to publish a message.**

  - **Configure identity verification information.**

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

  - **Set connection parameters.**

    ```
    //  Construct   a   client .
     DefaultPro  file   profile  =  DefaultPro  file . getProfile (
     regionId ,  accessKey ,  accessSecr  et );
     IAcsClient   client  =  new   DefaultAcs  Client ( profile );
    ```

  - **Set the parameters for publishing a message.**

    ```
     PubRequest   request  =  new   PubRequest ();
     request . setQos ( 0 );
    //  Set   the   topic   to   which   the   message   is
     published .
    ```

```
 request . setTopicFu  llName ("/" +  productKey  + "/" +
 deviceName  + "/ user / cloudmsg ");
 request . setProduct  Key ( productKey );
//  Set   the   message   content .  The   message   content
 must   be   encoded   in   Base64 .  Otherwise ,  the   message
 content   will   be   garbled   characters .
 request . setMessage  Content ( Base64 . encode ("{\" accuracy
 \": 0 . 001 ,\" time \": now }"));
```

- **Publish the message.**

```
 try  {
      PubRespons  e   response  =  client . getAcsResp  onse (
 request );
      System . out . println (" pub   success ?:" +  response .
 getSuccess ());
 }  catch  ( Exception   e ) {
      System . out . println ( e );
 }
```

The device receives the following message:

```
 msg  = [{" accuracy ": 0 . 001 ," time ": now }]
```

Appendix: demo

Click here to download and view the complete demo for this example.