Alibaba Cloud IoT Platform

Quick Start

Issue: 20180925

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

Contents

1 Quick start for IoT Platform Basic

This topic describes how to use the features provided by IoT Platform Basic. You can use an available sample program to implement bi-directional data transfer between a device and the cloud.

Scenario

A sample program is used to simulate a physical device and establish a connection from the device to IoT Platform. The platform forwards the device messages to Alibaba Cloud Message Service (MNS), and applications in the cloud listen on the MNS queues to obtain the device messages. Meanwhile, the cloud calls the Pub API to publish the messages to the platform. The device subscribes to the messages to obtain the instructions sent from the cloud.

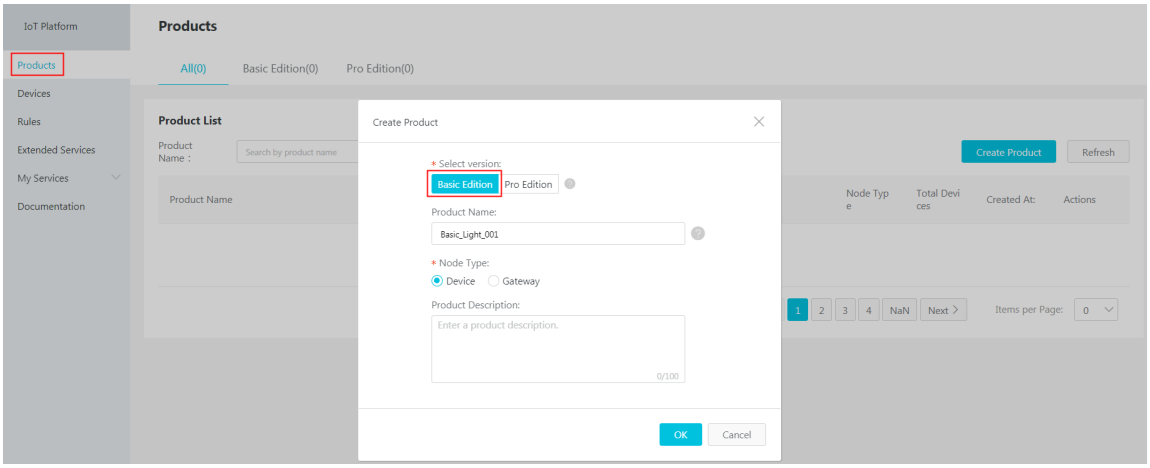
After you have completed the tasks described in this topic, message transfer is completed as shown in the following figure.

1.1 Add information in the console

When using IoT Platform, add products and devices in the console first. Then, obtain the device authentication and topic information. IoT Platform authenticates devices based on their ProductKey, DeviceName, and DeviceSecret information. It transfers data between devices and the cloud based on topics.

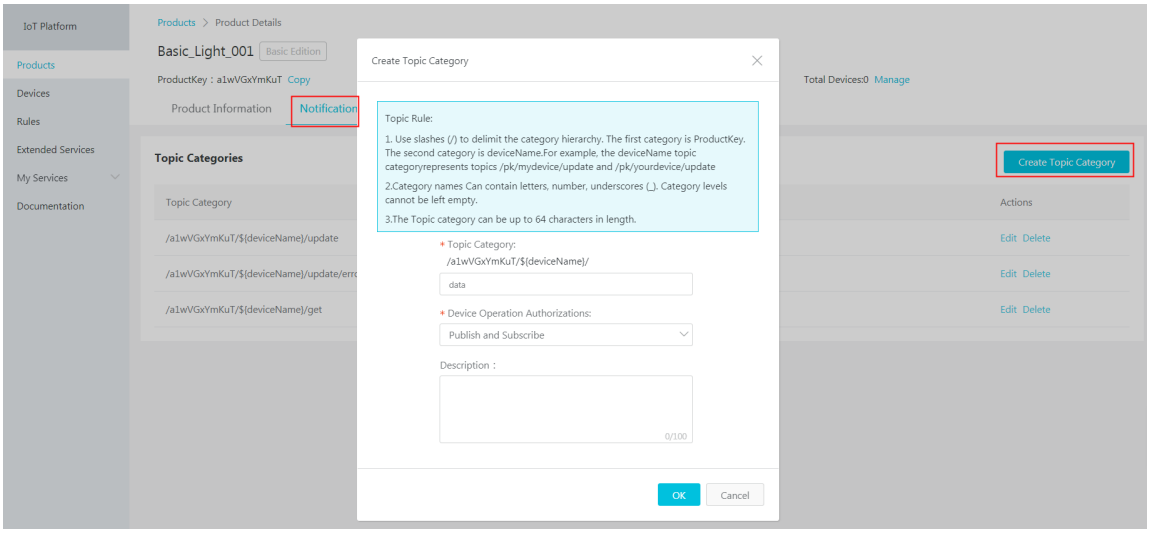
Procedure

1. Log on to the IoT Platform console using your Alibaba Cloud account. If this is your first time using this service, you need to make a request to activate it.
2. Create a product. A product is a group of devices sharing the same features. Creating a product can help you manage multiple devices at the same time.
   a) On the **Products** page, click **Create Product** and select **Basic Edition**.
   b) Enter the **`Product Name`** and select **`Node Type`**.



- Product Name: Identifies the product. Product names under the same account must be unique. In this example, we use the product model as the product name.
- Node Type: Select Device. With this setting, the devices under this product can directly connect to the IoT Hub.
  - Device: Sub-devices cannot be mounted. The devices can directly connect to the IoT Hub or be mounted to the IoT Hub as a sub-device of the gateway.
  - Gateway: Sub-devices can be mounted. The gateway has a sub-device management module that maintains the topology of sub-devices. The gateway can synchronize the topology to the cloud.

c) On the **Notifications** page, click **Define Topic Category** to create a new topic category, which transfers information uploaded by the devices.

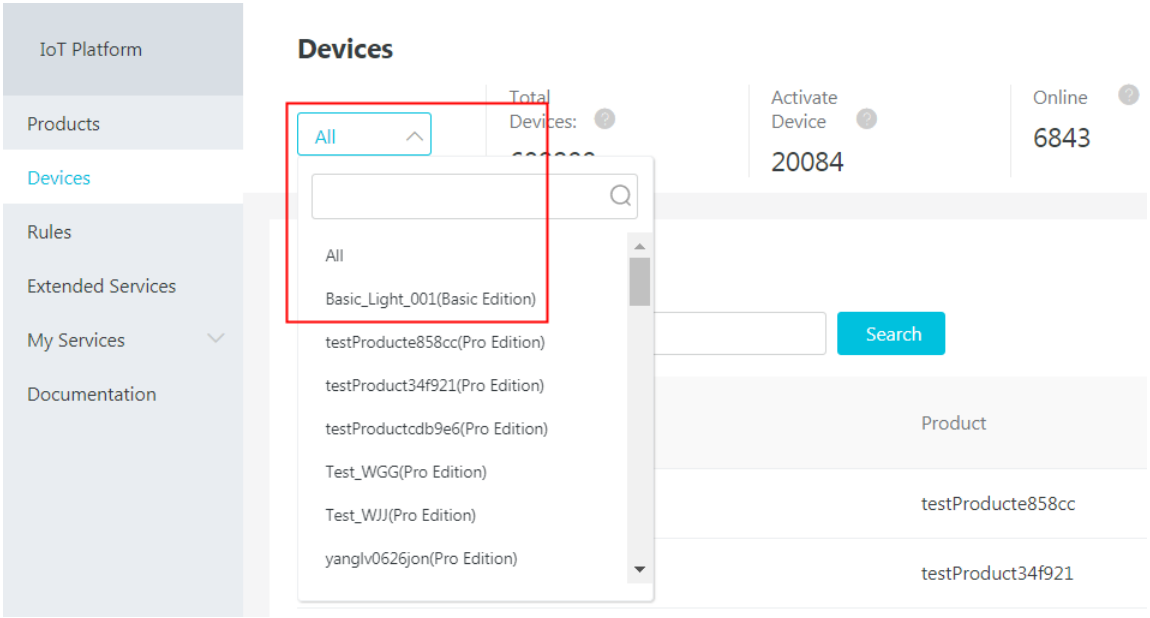


- Topic Category: Completes the topic suffix. In this example, enter **`data`**.

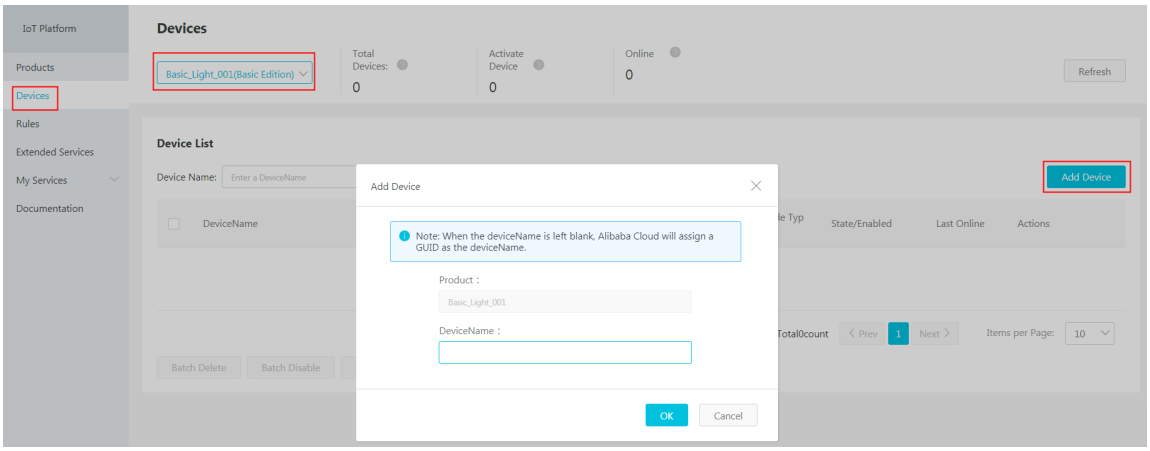
- Operation Permission: Select the permission that the device has for the corresponding topic. Select **Publish and Subscribe**. This means that the devices in this product can send messages to topics with a data suffix. They can also subscribe to messages from this topic.

3. Click **Devices** to enter the Devices page and add devices.

a) Select the product created in the previous step.



b) Under this product category, click **Add Device** to add a device.

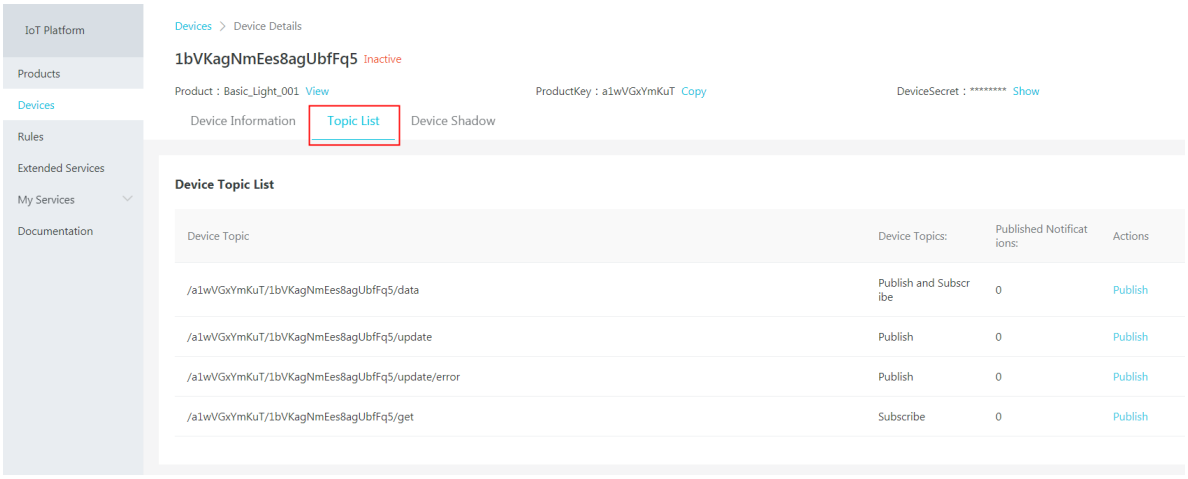


c) Once the devices are added, the following information is shown: ProductKey, DeviceName, and DeviceSecret. You can copy and save this information for device authentication later. The following is an example of the authentication information:

```
ProductKey: a1wmrZPO8o9
DeviceName: cbgiotkJ4O4WW59ivysa
```

```
DeviceSecret: H3cI7**********************ZeSU
```

4. Click **View** to view the devices you have added. Click **Topics** on the left side to view the permissions that the devices have to publish or subscribe to messages in different topics.



1.2 Establish a connection between a device and IoT Platform

Alibaba Cloud IoT Platform provides device SDKs that allow devices to connect to the platform. In this section, we use a sample program provided by the platform to simulate a device and develop the SDK for the device to implement communication between the device and IoT Platform.

Prerequisites

1. Select your development environment and an SDK for the environment. This example uses the C SDK for Linux.
2. The following software will be used in the SDK development and compilation environment: `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`. Run the following command to install the software:

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

Connect the device to IoT Platform

1. Use either of the following methods to download the device SDK:
   - Use **`gitclone`** to clone the source code to the Linux system by running the `git clone https://github.com/aliyun/iotkit-embedded` command.

     After the code is cloned, run the `git branch -r` command to view all branches, and run the **`git checkout`** command to access the latest branch.
   - Download the latest compressed package from the SDK download page, and decompress the package in the Linux environment. For more information, see *Download device SDKs*.

2. Add the obtained ProductKey, DeviceName, and DeviceSecret of the device to the `iotkit-embedded/sample/mqtt/mqtt-example.c` file.

```
#define PRODUCT_KEY "a1wmrZPO8o9"
#define DEVICE_NAME "cbgiotkJ4O4WW59ivysa"
#define DEVICE_SECRET "H3cI7**********************ZeSU"
```

3. Run the following commands to compile the SDK and generate a sample program:

```
make distclean

make
```

The generated sample program is in the `iotkit-embedded/output/release/bin` directory.

```
+-- bin
| +-- coap-example
| +-- ...
| +-- mqtt-example
+ -- Include
| +-- exports
| | +-- iot_export_coap.h
| | +-- ...
| +-Iot_export_shadow.h
| +-- imports
| +-Iot_import_coap.h
| | +-- ...
| | +-- iot_import_ota.h
| +-- iot_export.h
| +-- iot_import.h
+-- lib
| + -- Libiot_platform.a
| +-- libiot_sdk.a
| +-- libiot_tls.a
+-- src
+-- coap-example.c
+-- http-example.c
+-- Makefile
+-- mqtt-example.c
```

4. Run the following commands to execute the sample program. You can see that the device becomes online in the console.

```
cd output/release/bin

./mqtt-example
```

5. After the device becomes online, it starts to send data to the platform.

The following logs are output for the connection:

```
```
[inf] iotx_device_info_init(40): device_info created successfully!
```
```

```
[dbg] iotx_device_info_set(50): start to set device info!
[dbg] iotx_device_info_set(64): device_info set successfully!
[dbg] guider_print_dev_guider_info(248
): ................................................
[dbg] guider_print_dev_guider_info(249): ProductKey : a1wmrZPO8o9
[dbg] guider_print_dev_guider_info(250): DeviceName : cbgiotkJ4O
4WW59ivysa
[dbg] guider_print_dev_guider_info(251): DeviceID : H3cI7
**********************ZeSU
[dbg] guider_print_dev_guider_info(253
): ................................................
[dbg] guider_print_dev_guider_info(254): PartnerID Buf : ,partner_id
=example.demo.partner-id
[dbg] guider_print_dev_guider_info(255): ModuleID Buf : ,module_id=
example.demo.module-id
[dbg] guider_print_dev_guider_info(256): Guider URL :
[dbg] guider_print_dev_guider_info(258): Guider SecMode : 2 (TLS +
Direct)
[dbg] guider_print_dev_guider_info(260): Guider Timestamp :
2524608000000
[dbg] guider_print_dev_guider_info(261
): ................................................
[dbg] guider_print_dev_guider_info(267
): ................................................
[dbg] guider_print_conn_info(225):
 ----------------------------------------
[dbg] guider_print_conn_info(226): Host : 10.125.0.27
[dbg] guider_print_conn_info(227): Port : 1883
[dbg] guider_print_conn_info(230): ClientID : gsYfsxQJgeD.DailyEnvDN
|securemode=2,timestamp=2524608000000,signmethod=hmacsha1,gw=0,
partner_id=example.demo.partner-id,module_id=example.demo.module-id|
[dbg] guider_print_conn_info(232): TLS PubKey : 0x43c910 ('-----
BEGIN CERTI ...')
[dbg] guider_print_conn_info(235):
 ----------------------------------------
[inf] iotx_mc_init(1703): MQTT init success!
[inf] _ssl_client_init(176): Loading the CA root certificate ...
cert. version : 3
serial number : 04:00:00:00:00:01:15:4B:5A:C3:94
issuer name : C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign
Root CA
subject name : C=BE, O=GlobalSign nv-sa, OU=Root CA, CN=GlobalSign
Root CA
issued on : 1998-09-01 12:00:00
expires on : 2028-01-28 12:00:00
signed using : RSA with SHA1
RSA key size : 2048 bis
basic constraints : CA=true
key usage : Key Cert Sign, CRL Sign
[inf] _ssl_parse_crt(144): crt content:451
[inf] _ssl_client_init(184): ok (0 skipped)
[inf] _TLSConnectNetwork(346): Connecting /a1wmrZPO8o9.iot-as-mqtt.
cn-shanghai.aliyuncs.com/1883...
[inf] mbedtls_net_connect_timeout(291): setsockopt SO_SNDTIMEO
timeout: 10s
[inf] _TLSConnectNetwork(359): ok
[inf] _TLSConnectNetwork(364): . Setting up the SSL/TLS structure...
[inf] _TLSConnectNetwork(374): ok
[inf] _TLSConnectNetwork(409): Performing the SSL/TLS handshake...
[inf] _TLSCCnnectNetwork(417): ok
[inf] _TLSConnectNetwork(421): . Verifying peer X. 509 certificate..
[inf] _real_confirm(93): certificate verification result: 0x04
```

```
[inf] iotx_mc_connect(2035): mqtt connect success!
```

1.3 Servers subscribe to device messages

When devices are connected to IoT Platform, they report data to the platform. Data in the platform can be pushed to your server through a HTTP/2 channel. In this step, we set the service subscription through HTTP/2 and configure for HTTP/2 SDKs. You can then connect your server to an HTTP/2 SDK and the server can receive device data.

Procedure

1. In the *IoT Platform console*, you can set the service subscription for a product.
   a) On the **Products** page, find the product for which you want to set the service subscription and click **View**.
   b) On the product details page, click **Service Subscription**, and then click **Set Now**.
   c) Select the types of notifications which you want to push to your server (HTTP/2 SDK) and click **Save**.
      - Device Upstream Notification: Indicates messages of the topics in the device topic list, whose **Device Authorizations** are **Publish**.
      - Device Status Change Notification: Indicates the notifications that are sent by the system when the statuses of devices change. For example, the notifications upon devices going online or going offline.

   The subscription configuration in the console takes effect about one minute after the configuration is completed.
2. Connect to the HTTP/2 SDK.

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

```
<dependency>
    <groupId>com.aliyun.openservices</groupId>
    <artifactId>iot-client-message</artifactId>
    <version>1.1.0</version>
</dependency>

<dependency>
    <groupId>com.aliyun</groupId>
    <artifactId>aliyun-java-sdk-core</artifactId>
    <version>3.7.1</version>
```

```
</dependency>
```

3. Connect the SDK and IoT Platform using the AccessKey information of your Alibaba Cloud account for identity authentication.

```
String accessKey = "xxxxxxxxxxxxxxxx";
String accessSecret = "xxxxxxxxxxxxxx";
String uid = "xxxxxxxxxxxxxxxx";
String region = "cn-shanghai";
String endPoint = "https://${uid}.iot-as-http2.${region}.aliyuncs.
com"
Profile profile = new Profile(endPoint, region, accessKey,
accessSecret);
MessageClient client = MessageClientFactory.messageClient(profile);
client.connect(messageToken -> {
    Message m = messageToken.getMessage();
    System.out.println("receive message from " + m);
    return MessageCallback.Action.CommitSuccess;
});
```

The parameters are introduced as follows:

- **accessKey** and **accessSecret**: Log on to the Alibaba Cloud console, move the pointer to your account image, and click **AccessKey**. You are directed to the **User Management** page and you can create a new AccessKey or view the AccessKey ID and AccessKey Secret of an existing AccessKey on this page.
- **uid**: Log on to the Alibaba Cloud console, move the pointer to your account image, and click **Security Settings**. You are directed to the **Account Management** page and you can view your account ID on this page.
- **region**: The region of your IoT Platform service. For region IDs, see *Regions and zones*.

4. Test if the HTTP/2 SDK can successfully receive device messages by simulating a device to push messages using the following sample codes.

   a) Push a test message `hello world!` from a simulated device to the topic`/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/data`. Compile the message in `iotkit-embedded/sample/mqtt/mqtt-example.c` of the C SDK on a Linux system. The code example is as follows:

```
#define TOPIC_DATA "/"PRODUCT_KEY"/"DEVICE_NAME"/data"
#define PRODUCT_KEY "a1wmrZPO8o9"
#define DEVICE_NAME "cbgiotkJ4O4WW59ivysa"
/* Initialize topic information */
memset(&topic_msg, 0x0, sizeof(iotx_mqtt_topic_info_t));
strcpy(msg_pub, "message: hello world!") ;
topic_msg.qos = IOTX_MQTT_QOS1;
topic_msg.retain = 0;
topic_msg.dup = 0;
topic_msg.payload = (void *)msg_pub;
topic_msg.payload_len = strlen(msg_pub);
EXAMPLE_TRACE("\n publish message: \n topic: %s\n payload: \%s\n
", TOPIC_DATA, topic_msg.payload);
```

```
rc = IOT_MQTT_Publish(pclient, TOPIC_DATA, &topic_msg);
EXAMPLE_TRACE("rc = IOT_MQTT_Publish() = %d", rc);
```

b) Execute the make command using the path `iotkit-embedded` to compile the device SDK. The statement format is as follows:

```
make distclean
```

```
make
```

c) Runs the updated sample program using the path:`iotkit-embedded/output/release/bin`.

d) Check if the cloud applications can successfully receive the message. If the message is successfully received, you can obtain the following data from the message callback of the SDK.

Parameter	Description
messageId	A 19-bit message ID generated by IoT Platform .
topic	The topic from which the message is sent. In this example, the topic is `/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/data`. If the message is a device status change notification, the topic format is `/as/mqtt/status/${productKey}/${deviceName}`.
payload	• The binary data that a Pro Edition device publishes to a topic. • If the message is a device status change notification, the format is as follows: `{ "status":"online" (or offline), //the device status "productKey":"xxxxxxxxxxx", //In the above example, the ProductKey is a1wmrZPO8o9 "deviceName":"xxxxxxxxxx", //In the above example, the DeviceName is cbgiotkJ4O4WW59ivysa "time":"2018-08-31 15:32:28.205", //The time when the notification is sent "utcTime":"2018-08-31T07:32:28.205Z",//The UTC time when the notification is sent "lastTime":"2018-08-31 15:32:28.195", //The time when the last message communication occurred before the status change "utcLastTime":"2018-08-31T07:32:28.195Z",//The UTC time when the last message communication occurred before the status change "clientIp":"xxx.xxx.xxx.xxx" //the public IP address of the device. }`
generateTime	The timestamp of the message generation, in millisecond.

Parameter	Description
qos	• 0: The message is only pushed one time. • 1: The message is pushed at least one time.

1.4 Cloud services receive data from devices

In this step, your devices can report data to the cloud, and cloud applications can get messages from devices by monitoring the MNS queues.

Procedure

1. In the console, configure the service subscriptions for products in order to enable automatic message forwarding from the platform to the MNS queues.
   a) Click **Products**, and then click **View** to view the product that you created.
   b) On the left-side navigation pane, click **Subscriptions**, click **Configure**, and then select the **Device Upstream Notification** and **Device Status Update Notification** check boxes.
      - The Device Upstream Notification option indicates that the platform automatically forwards the data reported by devices to the MNS service.
      - The Device Status Update Notification option indicates that the platform will automatically push a notification to the MNS service when the device status (online or offline) changes .

   Once you subscribe to the MNS service, the MNS service will automatically create a message queue in the China (Shanghai) region. For example, the name is `aliyun-iot-a1wmrZPO8o9` , where `a1wmrZPO8o9` is the productKey of the device.

   The configuration will take effect after one minute.
2. The cloud applications receive device data by monitoring the MNS queues. In this example, assuming that your cloud applications access the MNS service using the Java SDK, and subscribe to topics in queue mode. The process involves the following information:
   a) A project file pom.xml has the following dependencies:

```
<dependency>
          <groupId>com.aliyun.mns</groupId>
          <artifactId>aliyun-sdk-mns</artifactId>
          <version>1.1.5</version>
```

```
</dependency>
```

b) You need to specify the following information when receiving a message:

```
CloudAccount account = new CloudAccount ($AccessKeyId, $AccessKeyS
ecret, $AccountEndpoint);
```

- The AccessKeyId and AccessKeySecret are required for you to access APIs. You can find them in your profile by clicking the Alibaba Cloud account avatar.
- The AccountEndpoint can be found in the MNS console. Select the **China (Shanghai)** region, locate the message queue that IoT Platform has created, and click **Get Endpoint**.

c) You will also need to specify information such as the IoT Message Queue, for example:

```
MNSClient client = account.getMNSClient();
CloudQueue queue = client.getQueueRef("aliyun-iot-a1wmrZPO8o
9"); //Input the queue that is automatically created as the
parameter.


while (true) {
//Get messages
Message popMsg = queue.popMessage(10);  //The timeout for long
polling is 10 senconds
if (popMsg ! = null) {
System.out.println("PopMessage Body: "
+ popMsg.getMessageBodyAsRawString()); //Get the original messages
.
queue.deleteMessage(popMsg.getReceiptHandle()); //Delete messages
from the queue.
} else {
System.out.println("Continuing");
}
}
```

d) Start to monitor the MNS queues.

Note:

For more information about how to monitor the MNS queues, see *MNS*.

3. Upload test messages and run sample programs to verify that the cloud services can receive messages by monitoring the MNS queues.

a) Upload a test message from a simulated device to the topic `/a1wmrZPO8o9/cbgiotkJ404WW59ivysa/data`. For example, send a `hello world!` message. You need to edit the message to be uploaded using the path: `iotkit-embedded/sample/mqtt/mqtt-example.c` with the C SDK on a Linux device. The code example is as follows:

```
#define TOPIC_DATA "/"PRODUCT_KEY"/"DEVICE_NAME"/data"
/* Initialize topic information */
```

```
memset(&topic_msg, 0x0, sizeof(iotx_mqtt_topic_info_t));
strcpy(msg_pub, "message: hello world!") ;
topic_msg.qos = IOTX_MQTT_QOS1;
topic_msg.retain = 0;
topic_msg.dup = 0;
topic_msg.payload = (void *)msg_pub;
topic_msg.payload_len = strlen(msg_pub);
EXAMPLE_TRACE("\n publish message: \n topic: %s\n payload: \%s\n
", TOPIC_DATA, topic_msg.payload);
rc = IOT_MQTT_Publish(pclient, TOPIC_DATA, &topic_msg);
EXAMPLE_TRACE("rc = IOT_MQTT_Publish() = %d", rc);
```

b) Execute the make command using the path `iotkit-embedded` to compile the device SDKs. The command is as follows:

```
make distclean
```

```
make
```

c) Runs the updated sample program using the path: `iotkit-embedded/output/release/bin`.

The output log of sent data is as follows:

```
mqtt_client|246::
publish message:
topic: /a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/data
payload: message: hello world!
```

d) Check if the cloud applications can successfully monitor the above message. If the monitoring is successful,

- the cloud applications will receive a message as follows:

```
{
"messageid":" ", //Message ID
"messagetype":"status",
"topic":"/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/data",
"payload": {hello world},
"timestamp": //the timestamp
}
```

  - messageid: The message ID generated by IoT Platform, which is 64-bit size.
  - messagetype: the message types, including notification, status and upload.
  - Topic: the topic the monitored messages are from. The type of topic is identified by messagetype. When the messagetype = status, it is an empty topic. When the messagetype = upload, it is a specific topic. In this example, the topic is `/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/data`.

- payload: **Base64 Encode data** . The type of data is identified by messagetype. When the messagetype = status, it indicates that IoT Platform sends a notification. When the messagetype = upload, it indicates that the device publishes data to IoT Platform.
- Timestamp: The timestamp, expressed in Epoch time.

- The following is a message example:

```
data=
{
"status":"online" (or offline), //the device status
"productKey":"xxxxxxxxxxx", //In this document, the ProductKey
is: a1wmrZPO8o9
"deviceName":"xxxxxxxxxx", //In this document, the DeviceName:
cbgiotkJ4O4WW59ivysa
"time":"2017-10-11 10:11:12.234", //the time that the
notification is sent
"lastTime":"2017-10-11 10:11:12.123" //the point in time that
 the device communicates with IoT Platform before the status
change.
"clientIp":"xxx.xxx.xxx.xxx" //the public IP address of the
device.
}
```

1.5 Devices receive commands from the cloud

In this step, the cloud application uses the PUB API to send messages to the topic. By subscribing to the topic, the device receives commands from the cloud.

Procedure

1. Modify the device SDK to subscribe to the topic: `/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/get`.

   a) Under the `iotkit-embedded/sample/mqtt/mqtt-example.c` directory, subscribe to the topic in the device SDK. The following is example code:

   ```
   /* Subscribe to a specific topic */
   rc = IOT_MQTT_Subscribe(pclient, TOPIC_GET, IOTX_MQTT_QOS1,
   _demo_message_arrive, NULL);
   if (rc < 0) {
   IOT_MQTT_Destroy(&pclient);
   EXAMPLE_TRACE("IOT_MQTT_Subscribe() failed, rc = %d", rc);
   rc = -1;
   goto do_exit;
   }
   ```

   b) The following is an example function that receives and processes data:

   ```
   static void _demo_message_arrive(void *pcontext, void *pclient,
   iotx_mqtt_event_msg_pt msg)
   {
   iotx_mqtt_topic_info_pt ptopic_info = (iotx_mqtt_topic_info_pt)
   msg->msg;
   /* print topic name and topic message */
   ```

```
EXAMPLE_TRACE("----");
EXAMPLE_TRACE("Topic: '%.*s' (Length: %d)",
ptopic_info->topic_len,
ptopic_info->ptopic,
ptopic_info->topic_len);
EXAMPLE_TRACE("Payload: '%.*s' (Length: %d)",
ptopic_info->payload_len,
ptopic_info->payload,
ptopic_info->payload_len);
EXAMPLE_TRACE("----");
}
```

c) Under the `iotkit-embedded` directory, run the make command to compile the SDK.

```
make distclean
```

```
make
```

d) The sample program generated is under the `iotkit-embedded/output/release/bin` directory. Run the sample program to complete topic subscription.

```
cd output/release/bin
```

```
./mqtt-example
```

2. The cloud application issues commands to the devices.

The cloud application calls the PUB API to send the data `hello world!` to the topic subscribed by the device. The following is an example of the Java code:

```
import com.aliyuncs.iot.model.v20170420. PubRequest
PubRequest request = new PubRequest();
request.setProductKey("a1wmrZPO8o9");
request.setMessageContent(Base64.encodeBase64String("hello world!".
getBytes()));
request.setTopicFullName("/a1wmrZPO8o9/cbgiotkJ4O4WW59ivysa/get");
Request. setqos (0); // Currently supports QoS0 and QoS1
PubResponse response = client.getAcsResponse(request);
System.out.println(response.getSuccess());
System.out.println(response.getErrorMessage());
```

Note:

- For a detailed description of how to issue commands from the cloud, see *Platform API*.
- The device has already subscribed to the topic information. Therefore, the platform automatically sends the corresponding message to the device when the cloud application issues this command.

3. Once the device receives the message from the cloud, the log shows the following:

```
[dbg] iotx_mc_cycle(1277): PUBLISH
```

```
[dbg] iotx_mc_handle_recv_PUBLISH(1091): Packet Ident : 00053506
[dbg] iotx_mc_handle_recv_PUBLISH(1092): Topic Length : 28
[dbg] iotx_mc_handle_recv_PUBLISH(1096): Topic Name : /a1wmrZPO8o9/
cbgiotkJ4O4WW59ivysa/get
[dbg] iotx_mc_handle_recv_PUBLISH(1099): Payload Len/Room : 19 / 990
[dbg] iotx_mc_handle_recv_PUBLISH(1100): Receive Buflen : 1024
[dbg] iotx_mc_handle_recv_PUBLISH(1111): delivering msg ...
[dbg] iotx_mc_deliver_message(866): topic be matched
_demo_message_arrive|151 :: ----
_demo_message_arrive|152 :: packetId: 53506
_demo_message_arrive|156 :: Topic: '/a1wmrZPO8o9/cbgiotkJ4O
4WW59ivysa/get' (Length: 28)
_demo_message_arrive|160 :: Payload: 'data: hello word!' (Length: 19
)
_demo_message_arrive|161 :: ----
```

2 Quick start for IoT Platform Pro

This topic describes how to use the features provided by IoT Platform Pro. You can define a device using Thing Specification Language (TSL) in the cloud and develop device firmware based on the TSL definition to implement bi-directional communication between a device and the cloud.

Prerequisites

- You have a registered Alibaba Cloud account.
- Your Alibaba Cloud IoT Platform service is activated.

  If you have not activated this service yet, go to the *Alibaba Cloud IoT Platform* page, click **Activate Now**. Follow the instructions on the page to activate this service.

Scenario

In this scenario, in order to understand the features provided by IoT Platform Pro, follow these steps: Add a product named *`Smart Irrigation`*, add a device named *`SK-1`* to the product, configure features including *`Power Switch`*, *`Automatic Sprinkler`*, and *`Fault Report`* for the product, and perform online debugging.

2.1 Create a product

This chapter provides step by step instructions on how to create an IoT product and a Thing Specialized Language (TSL).

2.1.1 Create a product category

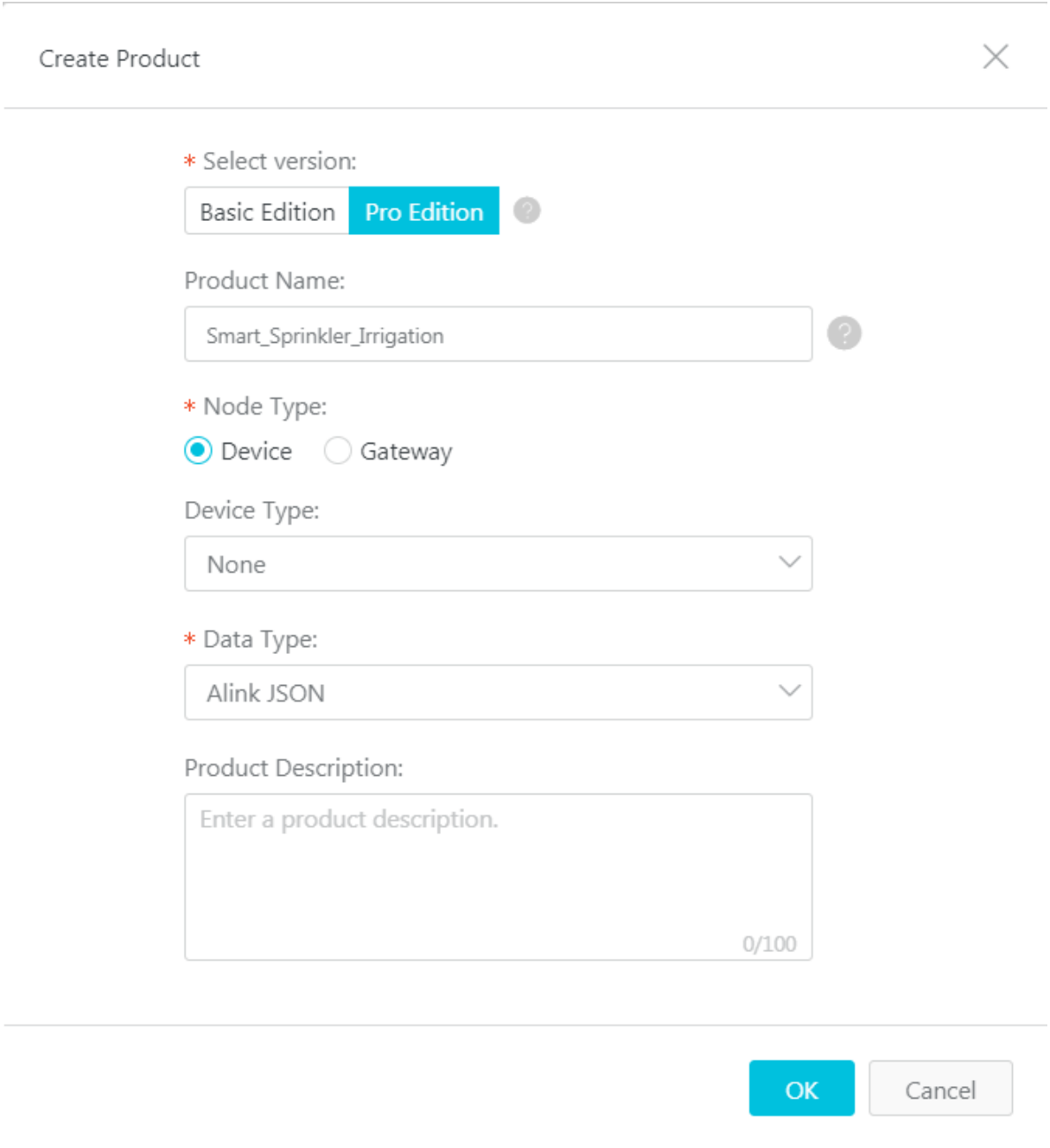
This chapter provides step by step instructions on how to create a product category and set product properties.

Procedure

1. Log on to the *IoT Platform console* using your Alibaba Cloud account.
2. Select **Products**, click **Create Product**.

   The **Create Product**dialog box appears, as shown in the following figure *Figure 2-1: Create a product*.

Figure 2-1: Create a product



3. Set product properties. A product is a collection of specified devices. The product can be used for device management.

   Here is an example of the *`Smart Sprinkler Irrigation`* product.

   - Select Edition: select the **`Pro Edition`** here.
   - Product Name: Enter **`Smart_Sprinkler_Irrigation`**as the product name. You can also customize the product name with a unique identifier and ensure that it is functioning under your account.

- Node Type: Select **Device** here.
- Device Type: A standard template with a set of predefined features.

  For example, standard features, such as power usage, voltage, ampere, total power consumption are predefined for smart meters. When you select **Smart Meter** for the device type, the standard features are automatically created based on the device type. You can modify features in the standard template, or add additional user-defined features to the template.

  If the device type is set to **None**, no standard features are created, but you will still be able to create user-defined features for your product.

  Here we will set it to **None**.
- Data Format: The format of the uploaded or downloaded device data, you can select **Alink JSON** or **Passthrough/Custom**. You can only choose one format to use. The two formats cannot be mixed.
  - Alink JSON is a data exchange protocol between devices and the cloud. It is provided in JSON format in the IoT Platform Pro Edition for developers.
  - If you want to customize the serial data format, you can choose **Passthrough/Custom** , and then convert the user-defined data to Alink JSON script, and configure the data parsing script in the cloud.

- Product Description: User-defined, it can be empty.

4. Click **OK**.

2.1.2 Define Thing Specification Language models

This section describes the procedure of setting features for a product that you have created.

Context

The system automatically creates standard features in the feature list for the product after you specify the **Device Type** when creating the product in the IoT Platform Pro.

Take **Smart Irrigation**as an example, to configure the properties, services, and events, follow these steps:

- Property: Power Switch
- Service: Automatic Sprinkler

- Event: Fault Report

Procedure

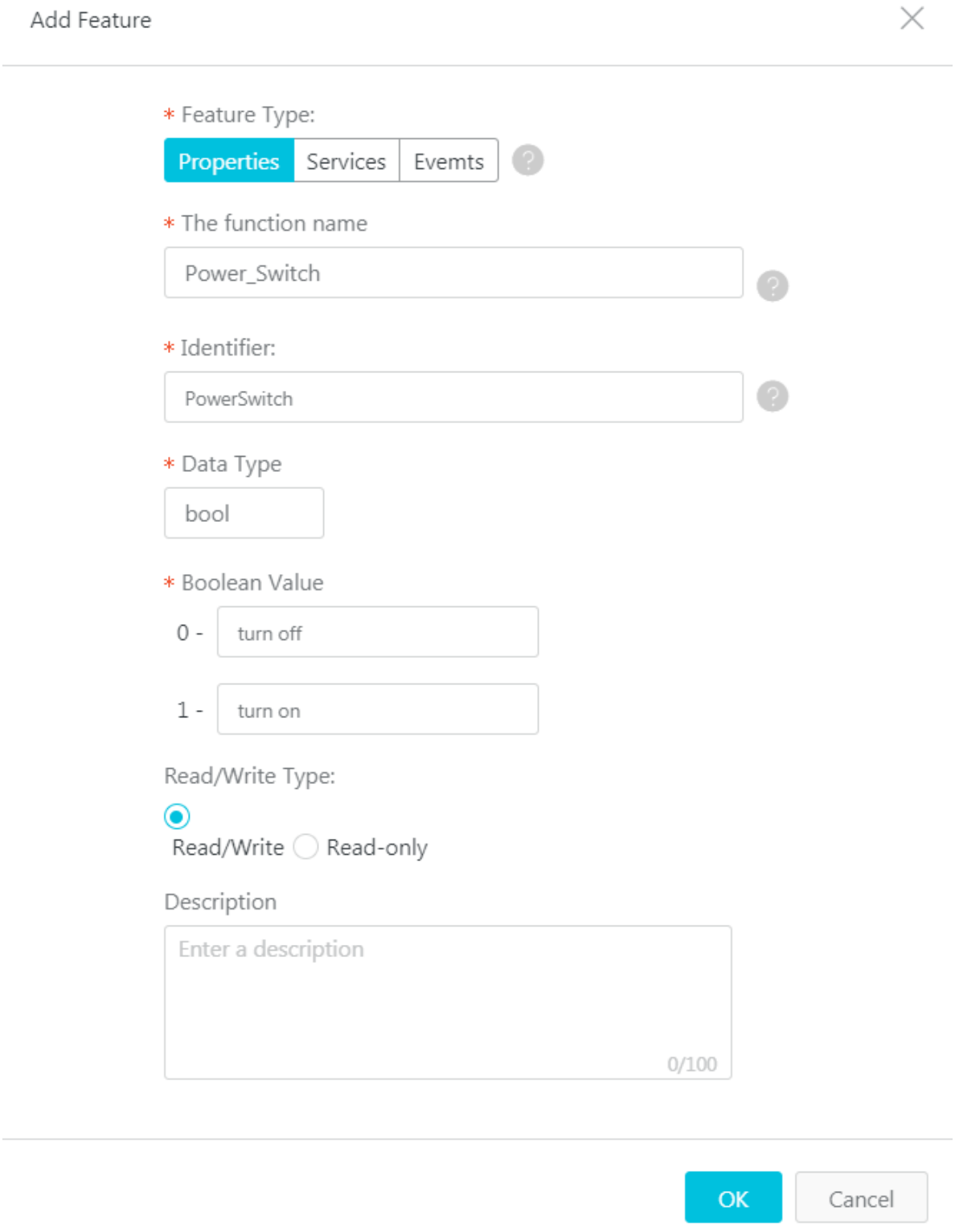
1. On the **Products** page, click **View** next to **`Smart Irrigation`**.

   The system displays the **Product Information** page.

2. Select **Define Feature**, and click **Add**.

   The system displays the **Add Feature** page, as shown in *Figure 2-2: Define a property*.

Figure 2-2: Define a property



3. Configure the **`Power Switch`** property.
   - Feature Type: select **`Property`**.

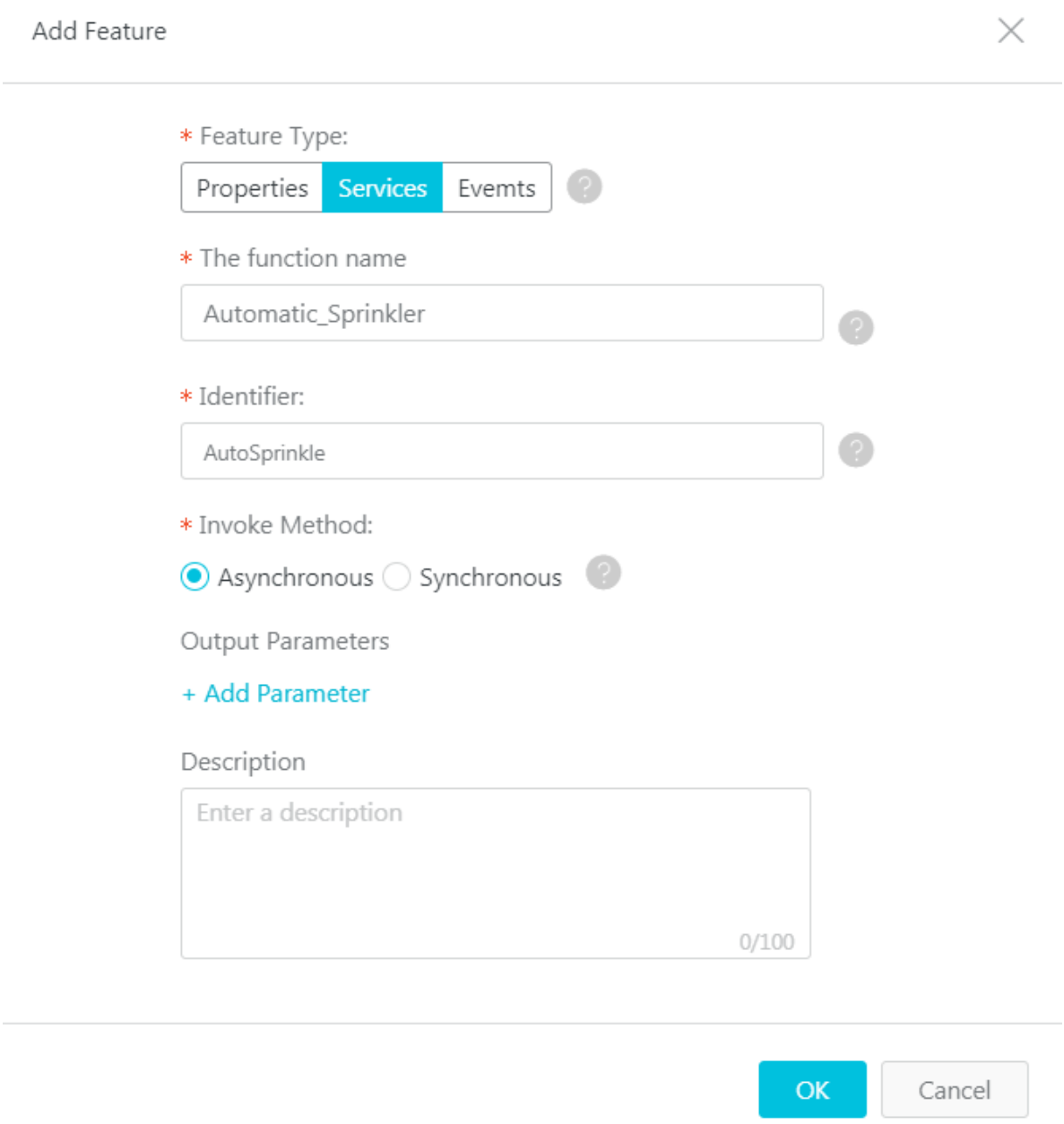
- Feature Name: enter **`Power Switch`** or another name.
- Identifier: enter **`PowerSwitch`** or another identifier.

  Note:
  The feature names and identifiers under the same product must be unique.
- Data Type: select Bool.
- Boolean: enter the description of the selected boolean value.

  When the device reports the property of the power switch, value 0 indicates that the power switch is turned off, and value 1 indicates that the power switch is turned on.
- Read/Write Type: select **`Read/Write`** to enable switch status reporting and remote control.

4. On the **Define Feature** page, click **Add** to configure the service of **`Automatic Sprinkler`**, as shown in *Figure 2-3: Automatic Sprinkler*.

Figure 2-3: Automatic Sprinkler



a) Set basic information about the service.

- Feature Type: select **`Service`**.
- Feature Name: enter **`Automatic Sprinkler`** or another name.
- Identifier: enter **`AutoSprinkle`** or another identifier.

> **Note:**
> The feature names and identifiers under the same product must be unique.

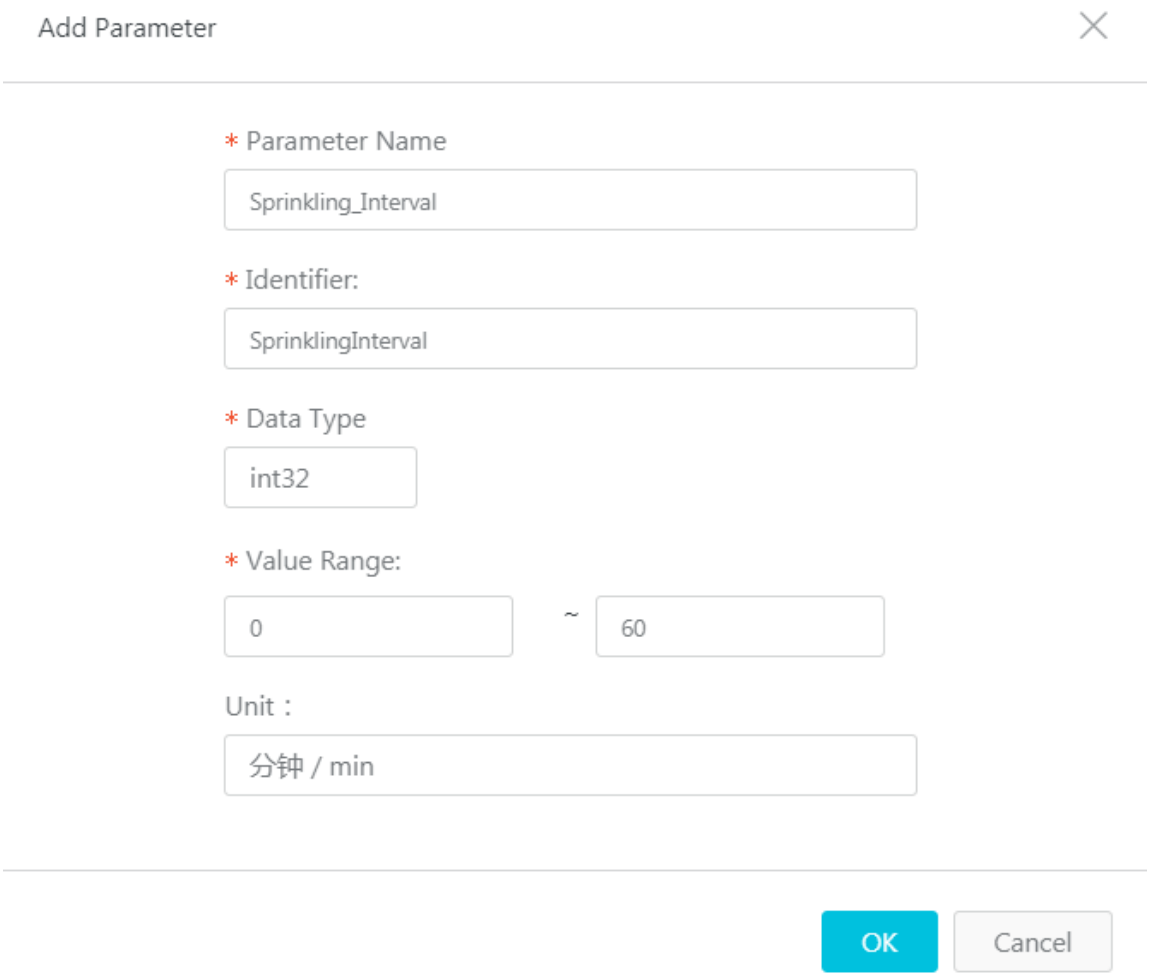
- Invoke Method: select **Asynchronous**.

b) Click **Add Parameter** next to **Input Parameters**.

You need to add two parameters for the **Automatic Sprinkler** service, including **Sprinkling Interval** and **Sprinkling Amount**. After you use the **Automatic Sprinkler** service, specify the sprinkling interval and amount to enable accurate sprinkling.

The system displays the **Add Parameter** page, as shown in *Figure 2-4: Sprinkling interval*.

Figure 2-4: Sprinkling interval



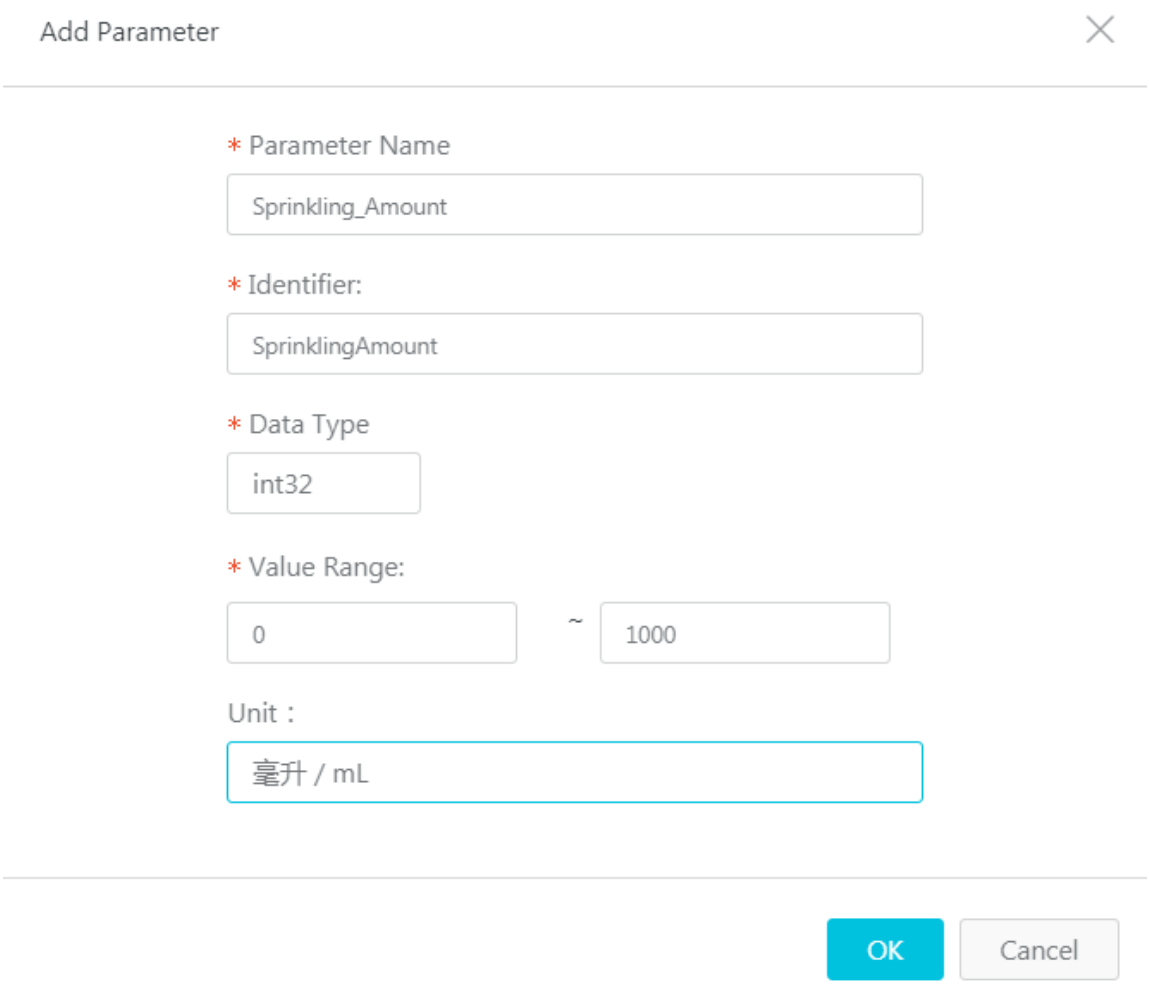
c) Set the Sprinkling Interval parameter.

- Parameter Name: enter **Sprinkling Interval** or another name.
- Identifier: enter **SprinkleInterval** or another identifier.
- Data Type: select **int32**.

- Value Range: this is set to 0 to 60. You can also customize your own range.
- Unit: select **min** to start sprinkling at an interval of 0 to 60 minutes.

d) Set the Sprinkling Amount parameter, as shown in *Figure 2-5: Sprinkling amount*.

Figure 2-5: Sprinkling amount



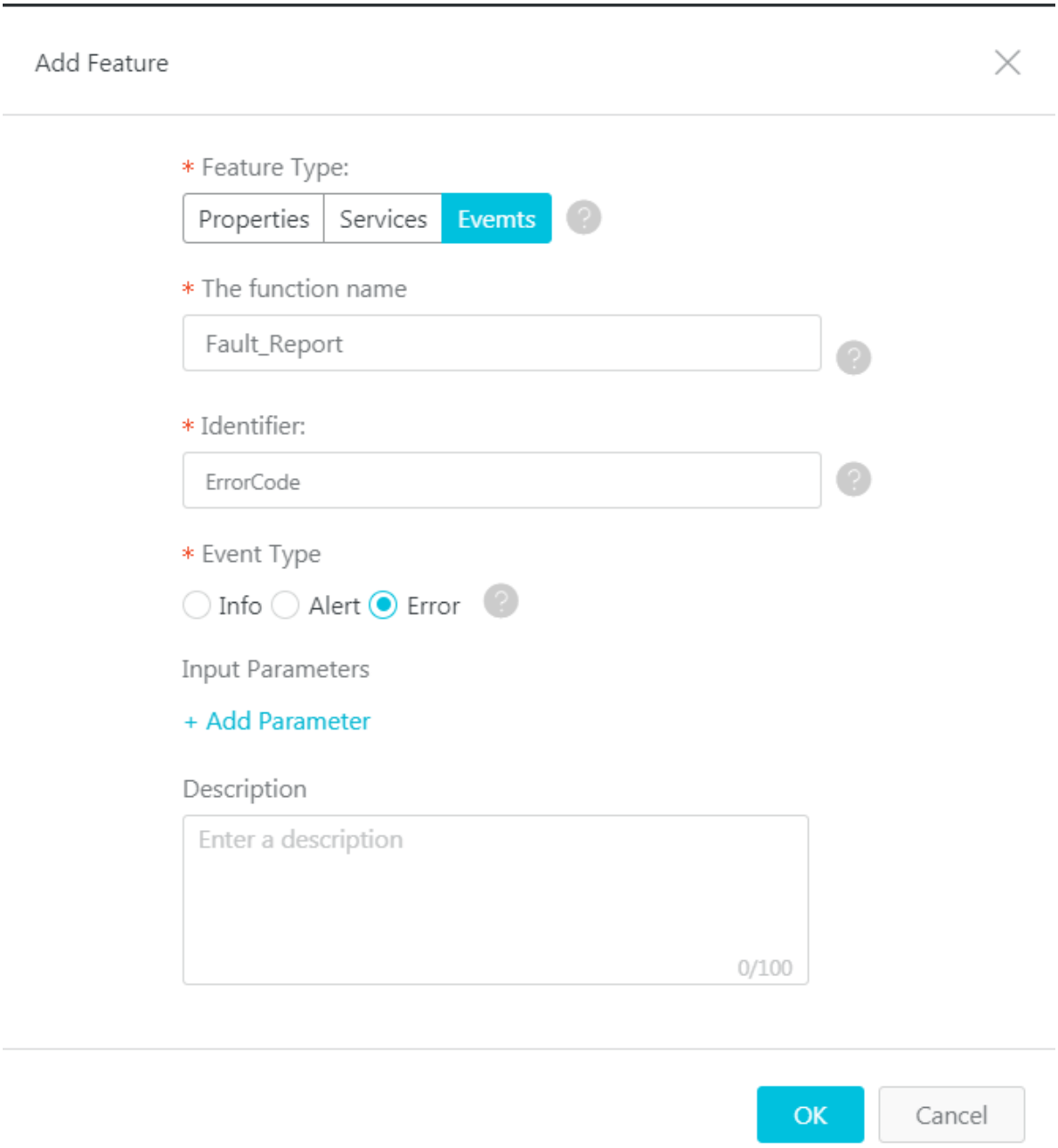
- Parameter Name: enter **Sprinkling Amount** or another name.
- Identifier: enter **SprinkleVolume** or another identifier.
- Data Type: select **int32**.
- Value Range: use the range from 0 to 1000 or another range.
- Unit: select **ml** to set the sprinkling amount from 0 to 1,000 ml each time.

e) Click **OK**.

The service supports response parameters. You can click **Add Parameter** next to **Output Parameters** to add a response parameter if required.

5. On the **Define Feature** page, click **Add** to configure the **`Fault Report`** event, as shown in *Figure 2-6: Fault reporting*.

Figure 2-6: Fault reporting

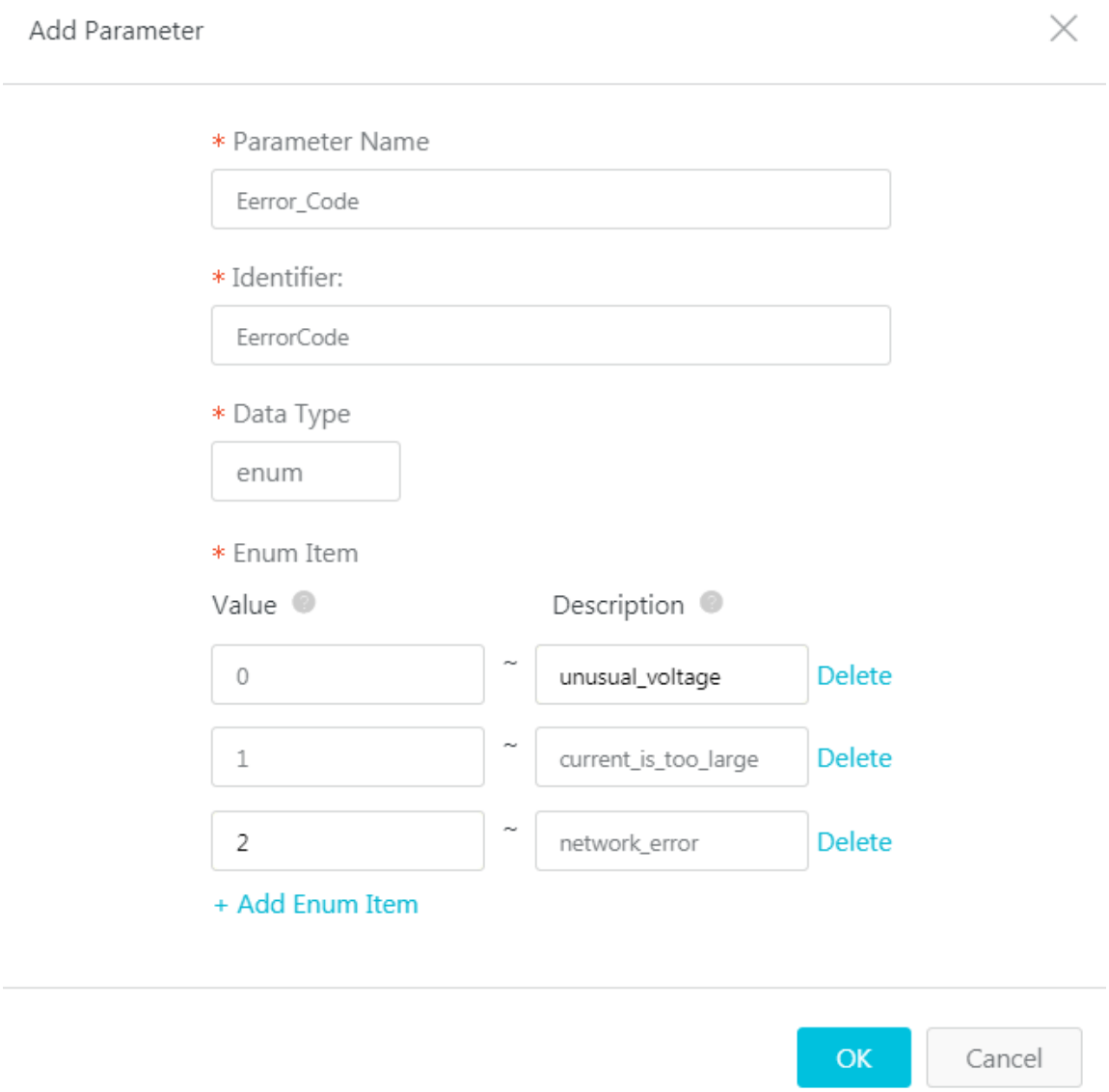


a) Set the basic parameters for fault reporting.

- Feature Type: select **`Event`**.
- Feature Name: enter **`Fault Report`**.
- Identifier: enter **`ErrorCode`** or another identifier.
- Event Type: select **`Fault`**.

b) Click **Add Parameter** next to **`Output Parameters`** to create a fault report event that contains an error code, as shown in *Figure 2-7: Error code*.

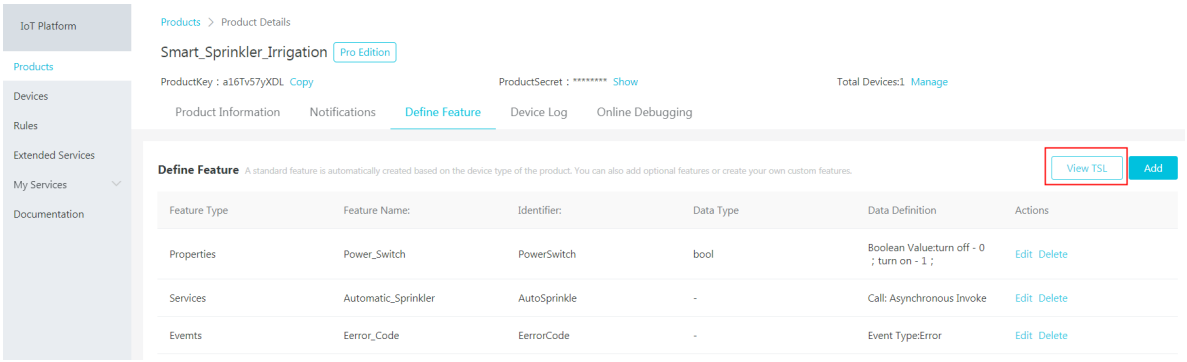
Figure 2-7: Error code



- Parameter Name: enter **`Error code`** or another name.
- Identifier: enter **`ErrorCode`**.
- Data Type: select **`enum`**.
- Enum item: customize the response parameters that are returned when different faults occur. For example, 0 indicates unusual voltage, 1 indicates that the current is too large, and 2 indicates a network error.

6. Return to the **Define Feature** page to view features that have been defined for Automatic Sprinkler, as shown in *Figure 2-8: Define a feature*.

Figure 2-8: Define a feature



7. Click **View TSL** to view TSL that is automatically generated according to the related feature definitions on IoT Platform.
8. Click **Export TSL File** to export the TSL file in the *model.json* format to a local directory. You may use this TSL file when configuring related devices.

2.2 Add devices

This topic describes how to add devices to a product after the product has been added.

Context

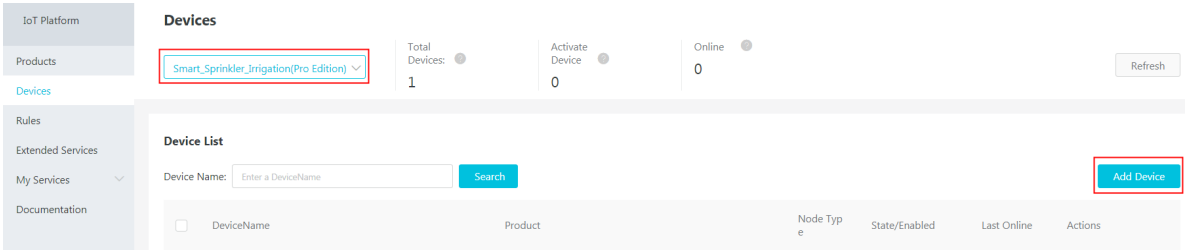
This example shows how to add a device named **SK-1** to Smart Irrigation.

Procedure

1. From the left-side navigation pane in the **IoT Platform** console, select **Devices**.

   The **Devices** page appears, as shown in *Figure 2-9: Devices*.

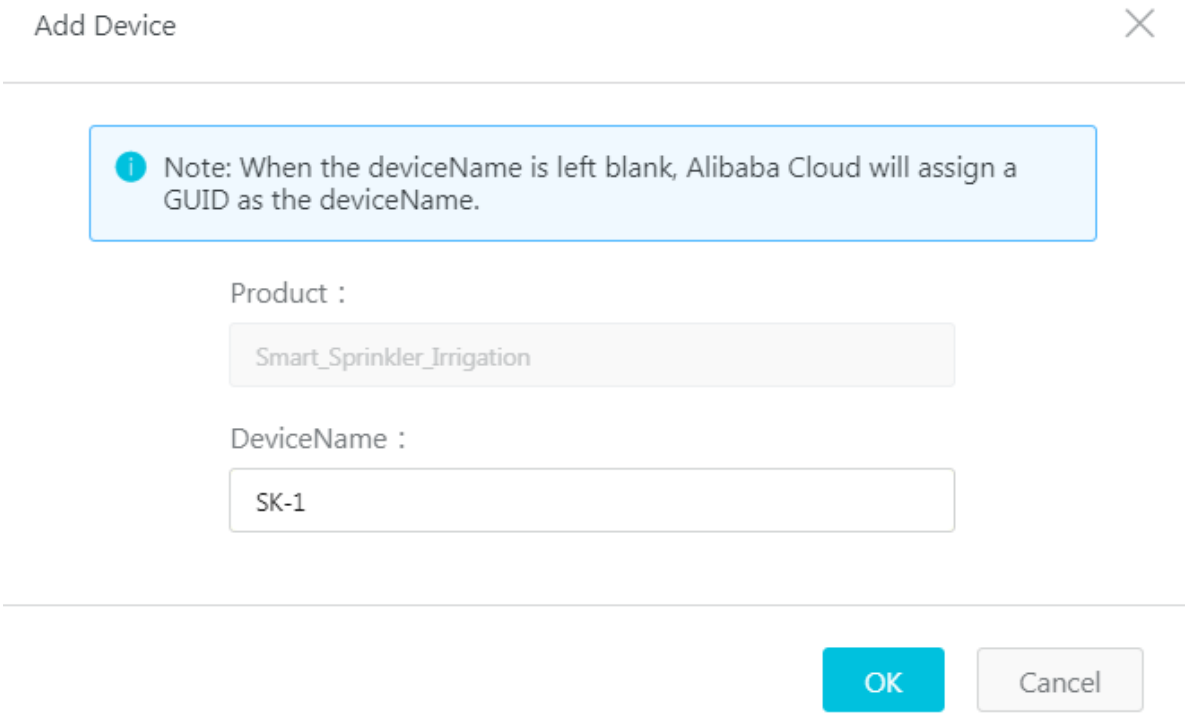
Figure 2-9: Devices



2. From the dropdown list of the product in the upper-left side of the page , select **Smart Irrigation**, and click **Add Device**.

The **Add Device** page appears, as shown in *Figure 2-10: Add Device*.

Figure 2-10: Add Device



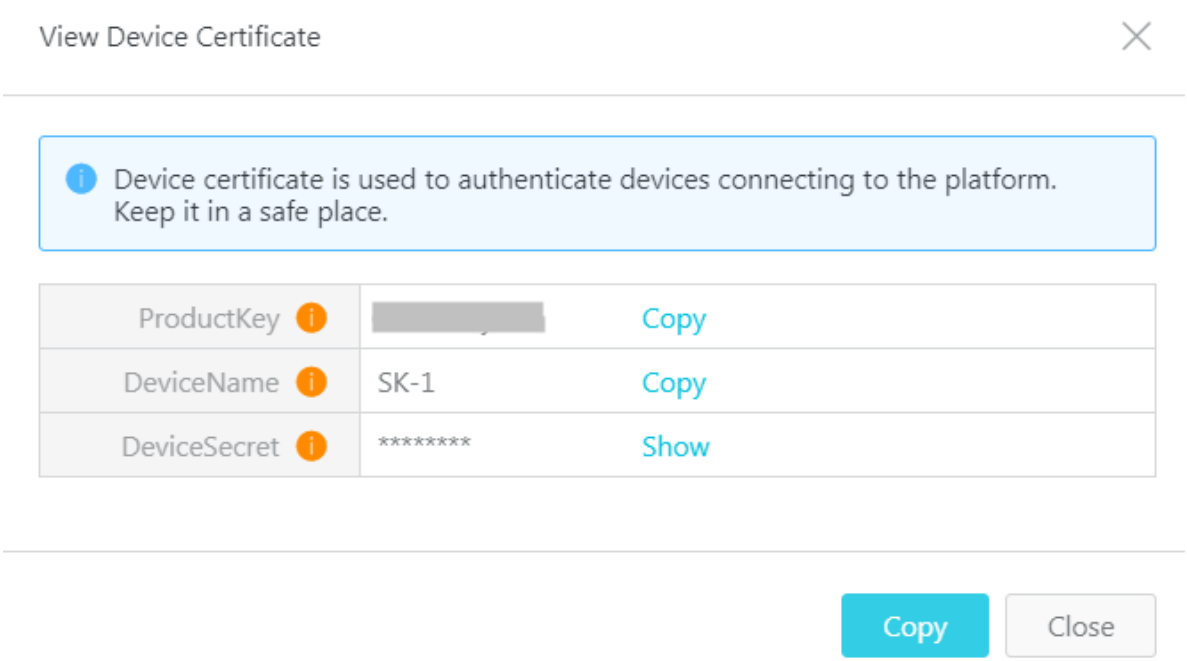
3. Set the device name to SK-1.

> **Note:**
> A device name uniquely identifies a device under a product and can be used to communicate with IoT Hub.

4. Click **OK**.

The **Added** page appears, as shown in *Figure 2-11: Added*.

Figure 2-11: Added



5. Click **Copy** to save the three key fields (ProductKey, DeviceName, and DeviceSecret) of the device.

   The three key fields will be added to the device SDK and used for authentication when the device connects to IoT Platform.

```
"product_key":"a1ekkixgSv1",
"device_name":"SK-1",
"device_secret":"Mqfov2L55E3IwTvy49ikhqs69FfQkScF",
```

   - product_key: Identifies a product category.
   - device_name: Identifies a device in a category.
   - device_secret: Device key.

6. Click **Close**.

2.3 Develop devices

Alibaba Cloud IoT Platform provides device SDKs to allow devices to communicate with the platform. This topic describes how to download the SDKs.

Prerequisites

This example uses the C SDK for Linux . We recommend that you use Ubuntu16.04 to compile the SDK.

Context

This task shows how to download an SDK for Smart Irrigation.

To develop a device, follow these steps:

1. Download the SDK.
2. Set the three key fields (ProductKey, DeviceName, and DeviceSecret) for authentication.
3. Configure a TSL model.

Procedure

1. Log on to a Linux VM instance.
2. Run the following command to install the software that is required for the SDK development and compilation environment.

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

   The software includes 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.
3. In the root directory, run the following command to rehost the source code of the SDK to GitHub.

   ```
   git clone https://github.com/aliyun/iotkit-embedded
   ```
4. Run the following commands to view all available SDK versions.

   ```
   cd iotkit-embedded/
   ```

   ```
   git branch -r
   ```

   The system displays the following versions:

   ```
   origin/HEAD -> origin/master
   origin/RELEASED_V1_0_1_20170629
   origin/RELEASED_V2_00_20170818
   origin/RELEASED_V2_01_20171010
   origin/RELEASED_V2_02_20171130
   origin/RELEASED_V2_03_20180131
   origin/RELEASED_V2_10_20180331
   origin/master
   ```
5. Run the following command to update the current SDK to the latest version. SDK files are named by release time. In this example, the latest version is **origin/RELEASED_V2_10_20180331**.

   ```
   git checkout SDK version
   ```

6. Run the following command to modify the *make.setting* file to configure the TSL model of the device, which is supported by IoT Platform Pro.

   Set both **FEATURE_CMP_ENABLED** and **FEATURE_DM_ENABLED** to **y**.

   ```
   vi iotkit-embedded/make.settings
   ```

   The system displays the following features:

   ```
   FEATURE_MQTT_COMM_ENABLED = y
   FEATURE_MQTT_DIRECT = y
   FEATURE_MQTT_DIRECT_NOTLS = n
   FEATURE_MQTT_DIRECT_NOITLS = y
   FEATURE_COAP_COMM_ENABLED = n
   FEATURE_HTTP_COMM_ENABLED = y
   FEATURE_SUBDEVICE_ENABLED = n
   FEATURE_CMP_ENABLED = y
   FEATURE_DM_ENABLED = y
   FEATURE_SERVICE_OTA_ENABLED = y
   ```

7. Run the following command to modify the ProductKey, DeviceName, and DeviceSecret in the *iotkit-embedded/sample/linkkit/samples/linkkit_sample.c* file.

   ```
   vi iotkit-embedded/sample/linkkit/samples/linkkit_sample.c
   ```

   Modify the ProductKey, DeviceName, and DeviceSecret of Smart Irrigation saved in *5* as follows:

   ```
   #define DM_PRODUCT_KEY_1 "a1a3Xryxx97"
   #define DM_DEVICE_NAME_1 "SK-1"
   #define DM_DEVICE_SECRET_1 "Dud7czC0DZgIo9pUlGgZ9zg9raoruMTC"
   ```

8. Modify the TSL model in the *iotkit-embedded/sample/linkkit/samples/linkkit_sample.c* file.

   You can use the following methods to obtain the TSL model:

   - During the operation, the device publishes a message to a topic to run the `/sys/{productKey}/{deviceName}/thing/dsltemplate/get` command to request the TSL model from IoT Platform.

     This method is memory-intensive and will produce a certain amount of data traffic. A TSL model consumes approximately 20 KB of memory and produces approximately 10 KB of network traffic. The actual amounts of memory and network traffic vary by the complexity of a TSL model.

- Manually export the TSL model from IoT Platform, and save the TSL model to the *linkkit_sample.c* file to preprovision a TSL model for the device. Make sure that the TSL model has been defined before you develop your device. Once the development starts, do not edit the TSL model. Otherwise, you must synchronize every modification to the preprovisioned TSL model on the device.

This example uses the TSL model downloaded from *8*.

Modify the following information:

```
#if !( WIN32)
const char TSL_STRING[] = "{\"schema\":\"https://iot-tsl.oss-cn-
shanghai.aliyuncs.com/schema.json\",\"profile\":{\"productKey\":
\"a1a3Xryxx97\"},\"services\":[{\"outputData\":[],\"identifier\":
\"set\",\"inputData\":[{\"identifier\":\"PowerSwitch\",\"dataType\":
{\"specs\":{\"0\":\"Off\",\"1\":\"On\"},\"type\":\"bool\"},\"name\":
\"Power Switch\"}],\"method\":\"thing.service.property.set\",\"name
\":\"set\",\"required\":true,\"callType\":\"sync\",\"desc\":\"Set
 Properties\"},{\"outputData\":[{\"identifier\":\"PowerSwitch\",
\"dataType\":{\"specs\":{\"0\":\"Off\",\"1\":\"On\"},\"type\":\"bool
\"},\"name\":\"Power Switch\"}],\"identifier\":\"get\",\"inputData
\":[\"PowerSwitch\"],\"method\":\"thing.service.property.get\",
\"name\":\"get\",\"required\":true,\"callType\":\"sync\",\"desc\":
\"Get Properties\"},{\"outputData\":[],\"identifier\":\"AutoSprinkle
\",\"inputData\":[{\"identifier\":\"SprinkleTime\",\"dataType
\":{\"specs\":{\"unit\":\"min\",\"min\":\"0\",\"unitName\":
\"minute\",\"max\":\"60\"},\"type\":\"int\"},\"name\":\"Sprinkling
 Interval\"},{\"identifier\":\"SprinkleVolume\",\"dataType\":
{\"specs\":{\"unit\":\"mL\",\"min\":\"0\",\"unitName\":\"milliliter
\",\"max\":\"1000\"},\"type\":\"int\"},\"name\":\"Sprinkling
 Amount\"}],\"method\":\"thing.service.AutoSprinkle\",\"name\":
\"Automatic Sprinkler\",\"required\":false,\"callType\":\"async
\"}],\"properties\":[{\"identifier\":\"PowerSwitch\",\"dataType\":
{\"specs\":{\"0\":\"Off\",\"1\":\"On\"},\"type\":\"bool\"},\"name\":
\"Power Switch\",\"accessMode\":\"rw\",\"required\":false}],\"events
\":[{\"outputData\":[{\"identifier\":\"PowerSwitch\",\"dataType\":
{\"specs\":{\"0\":\"Off
\",\"1\":\"On\"},\"type\":\"bool\"},\"name\":\"Power Switch\"}],
\"identifier\":\"post\",\"method\":\"thing.event.property.post\",
\"name\":\"post\",\"type\":\"info\",\"required\":true,\"desc\":
\"Report Properties\"},{\"outputData\":[{\"identifier\":\"ErrorCode
\",\"dataType\":{\"specs\":{\"0\":\"Voltage Anomaly\",\"1\":
\"Overcurrent\",\"2\":\"Network Fault\"},\"type\":\"enum\"},\"name
\":\"Fault Code\"}],\"identifier\":\"ErrorCode\",\"method\":
\"thing.event.ErrorCode.post\",\"name\":\"Fault Report\",\"type\":
\"error\",\"required\":false}]}";
```

Note:

The downloaded TSL model of Smart Irrigation is originally in *.json* format. When you change the format to the C language, be careful with the format and escape characters.

2.4 Connect a device to IoT Platform

This topic describes how to connect a device to IoT Platform.

Procedure

1. Run the following commands to compile the SDK to generate a sample program.

   ```
   cd iotkit-embedded

   make distclean

   make
   ```

   A sample file is generated and saved to the *output/release/bin/mqtt-example* directoty.

2. Run the following command to connect the device to the platform.

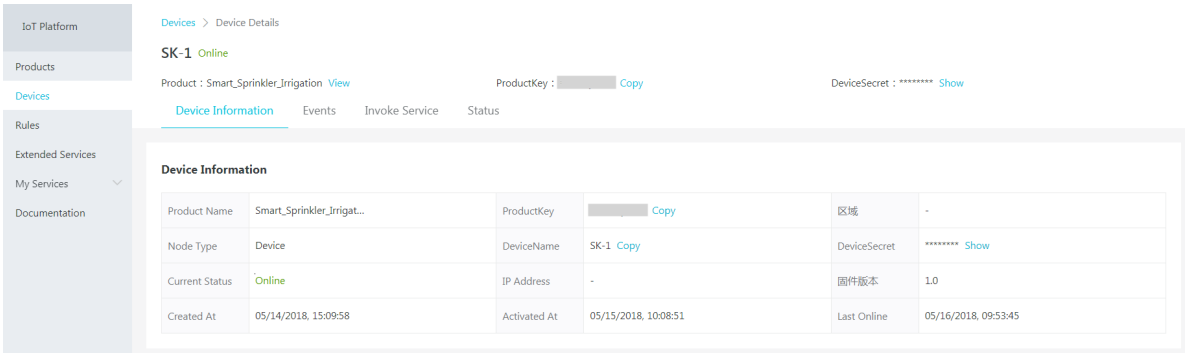
   ```
   ./output/release/bin/mqtt-example
   ```

   The system displays the following messages, indicating that the device has successfully connected to the platform.

   ```
   [dbg]
    guider_print_dev_guider_info(248): .....................................
   [dbg] guider_print_dev_guider_info(249): ProductKey : ***********
   [dbg] guider_print_dev_guider_info(250): DeviceName : SK-1
   [dbg] guider_print_dev_guider_info(251): DeviceID : ***********.
   device-test[dbg]
    guider_print_dev_guider_info(253): .....................................
   [dbg] guider_print_dev_guider_info(254): PartnerID
    Buf : ,partner_id=example.demo.partner-id
   [dbg] guider_print_dev_guider_info(255): ModuleID
    Buf : ,module_id=example.demo.module-id
   [dbg] guider_print_dev_guider_info(256): Guider URL :
   [dbg] guider_print_dev_guider_info(258): Guider SecMode : 2
   [dbg] guider_print_dev_guider_info(260): Guider Timestamp :
    2524608000000
   [dbg]
    guider_print_dev_guider_info(261): .....................................
   [inf] iotx_mc_connect(2035): mqtt connect success!
   ```

3. Log on to the IoT Platform console, select **Devices**, select **Smart Irrigation** from the product list, select **SK-1** from the device list, and then click **View**. Verify that the device is in **Online** status, as shown in *Figure 2-12: Online status*.

Figure 2-12: Online status



2.5 Online debugging

This topic describes how to debug devices online in the console, including debugging and verifying the upstream and downstream features of devices.

Prerequisites

The target device is running with normal access to the cloud.

Debugging scenario

Debug and verify the upstream and downstream features between the Automatic Sprinkler SK-1 device and the cloud using properties, services, and events.

2.5.1 Preparation

Procedure

1. Log on to the IoT Platform console.
2. Select **Devices**, and click **View** that follows the **`Auto Sprinkler`** product to enter the **Device Details** page.
3. Click **Device Debugging** to enter the **Online Debugging** page.
4. Select **`SK-1`** from the **`Real-time Log`** drop-down list box.

2.5.2 Property debugging

This section describes the detailed debugging steps for property reporting and property settings.

Procedure

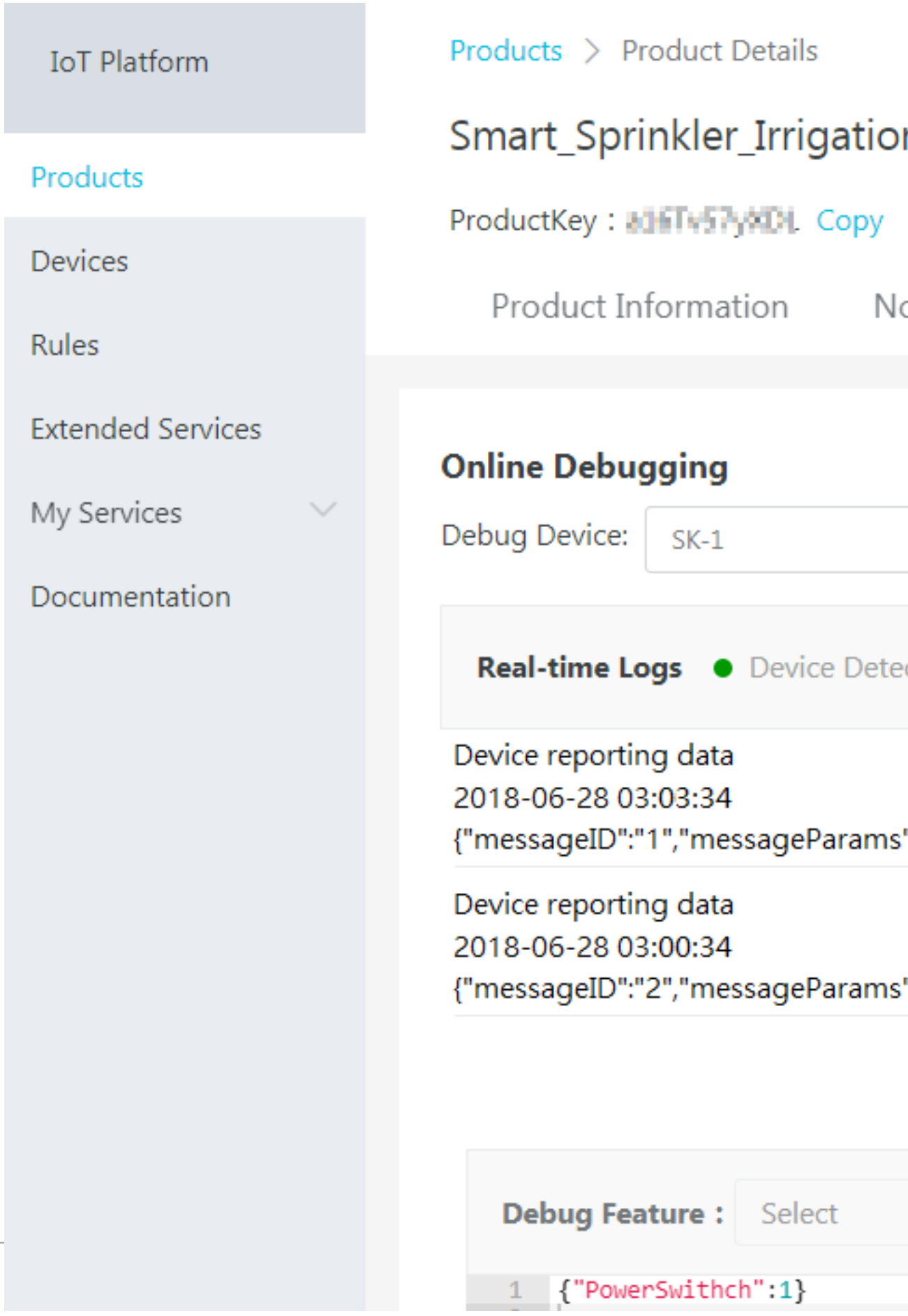
1. Select **`Debug Mode`**, and then select **`Power Switch`**.
2. Debug the property reporting feature.

a) Set the method to **`Get`**.

b) Click **Dispatch Command**.

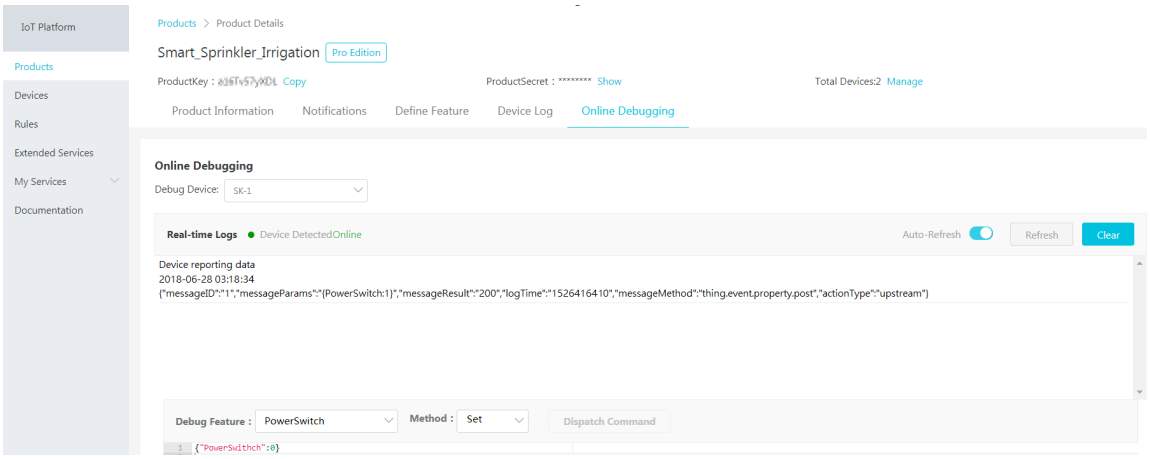
The status of the property on the current device appears in the parameters window. At the same time, the Real-time Log area displays the data reported by the device in real time, as shown in *Figure 2-13: Get*.

Figure 2-13: Get



3. Debug the property setting feature.
   a) Set the method to **Set**.
   b) Specify properties in JSON format in the parameters window.
   c) Click **Dispatch Command**. Then, the device receives the corresponding set command, as shown in *Figure 2-14: Set*.

Figure 2-14: Set



2.5.3 Debug services

This topic describes how to debug a remotely triggered device service.

Context

In this task, the sprinkling interval is 50 minutes, and the sprinkling amount is 600 milliliters.

Procedure

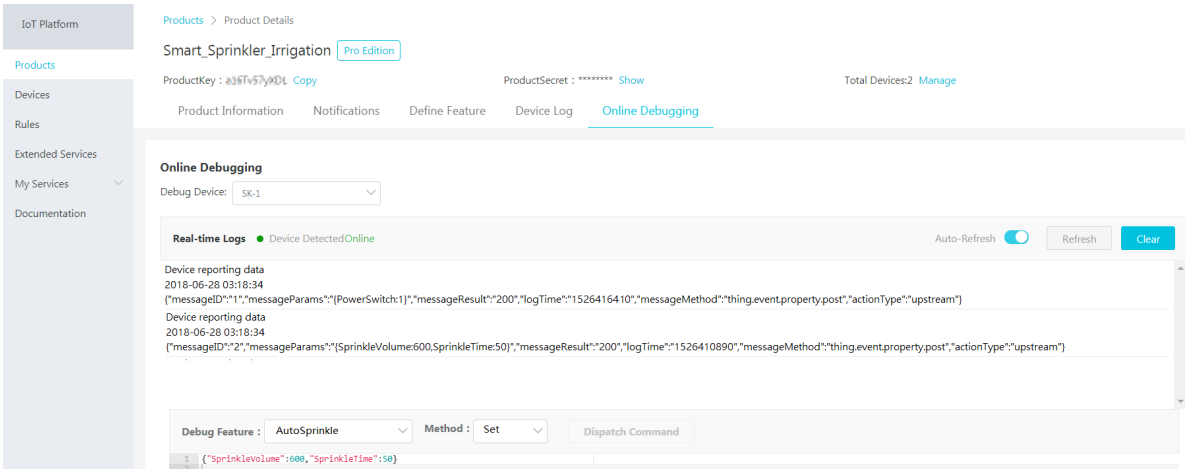
1. Select **Automatic Sprinkler** as the service to be debugged.
2. Enter valid parameter values.

   Example:

   ```
   {"SprinkleTime":50,"SprinkleVolume":600}
   ```

3. Click **Dispatch Command**. The device will receive the command from the cloud, as shown in *Figure 2-15: Automatic sprinkler*.

Figure 2-15: Automatic sprinkler



Note:

- Make sure that the parameters are in the exact format that was specified when you created the product. Otherwise, the cloud cannot send the command.
- Make sure that the parameter values are valid. Otherwise, the device cannot parse the command and returns an error.

2.5.4 Event debugging

This section describes how to debug the process of obtaining the latest events from your device.

Procedure

1. Set the feature that you want to debug to **`Fault Report`**, and set the method to **`Get`**.
2. Click **Dispatch Command** to search the latest event that the device has reported to the cloud, as shown in *Figure 2-16: Fault reporting*.

Figure 2-16: Fault reporting