Alibaba Cloud
物联网平台

Device Access

Issue: 20191126

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, consequent**

ial, 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 contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, 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 this document 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.**

Document conventions

Style	Description	Example
	A danger notice 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.
	A warning notice 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 restart an instance.
	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 **Notice:** If the weight is set to 0, the server no longer receives new requests.
	A note indicates supplemental instructions, best practices, tips, and other content.	 **Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click Settings > Network > Set network type.
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands.	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *`Instance_ID`*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`

Style	Description	Example
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Contents

1 Download device SDKs

IoT Platform provides multiple device SDKs to help you develop your devices and connect them to IoT Platform.

Prerequisites

Before you develop a device SDK, you must complete all the required configurations in the IoT Platform console and obtain all necessary information (such as the device certificate information and topic information). For more information about console configurations, see the related documents in the User Guide.

Device SDKs provided by IoT Platform

You can install and configure an SDK provided by IoT Platform on your device, and then your device can connect to IoT Platform by using the SDK.

Download device SDK from *C SDK*

If the provided SDK does not meet your business requirement, contact Alibaba Cloud by sending an email to `linkkitSDK-query@list.alibaba-inc.com`. The email content must include the following information:

- **Subject:** `Consultation on IoT Platform SDKs`
- **The body of the email must include:**

```
Company name
Contact
Phone Number
The programming language or software that you use to develop your
device
Your business requirements
Your device amount and development schedule
```

Develop your own SDK based on the Alink protocol

If you have specific development requirements that cannot be met by the provided SDK, you can develop your own SDK based on the Alink protocol. For more information, see *Alink protocol*.

Generic protocol SDK

You can use the *generic protocol SDK* **provided by IoT Platform to build a bridge to connect your devices or platforms with IoT Platform, so that they can communicate with each other.**

2 Create a product

The first step when you start using IoT Platform is to create products. A product is a collection of devices that typically have the same features. For example, a product can refer to a product model and a device is then a specific device of the product model.

Context

This topic describes how to create products in the IoT Platform console.

Procedure

1. **Log on to the** *IoT Platform console*.
2. **In the left-side navigation pane, click Devices > Product, and then click Create Product.**
3. **Enter all the required information and then click OK.**

   The parameters are as follows.

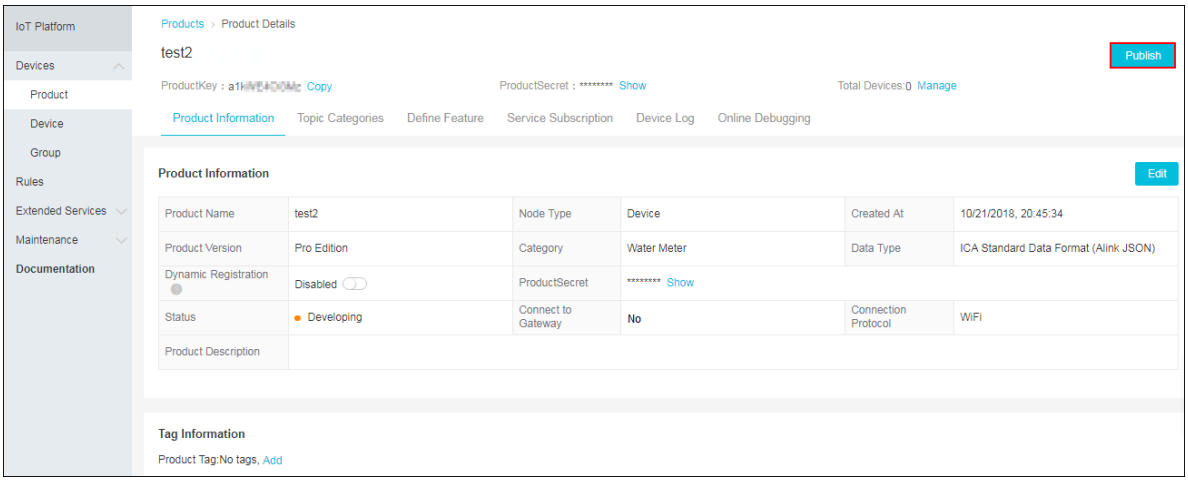
Parameter	Description
Product Name	The name of the product that you want to create. The product name must be unique within the account. For example, you can enter the product model as the product name. A product name is 4 to 30 characters in length, and can contain Chinese characters, English letters, digits, and underscores. A Chinese character counts as two characters.
Node Type	· Direct devices: Indicates that devices of this product connect to IoT Platform directly. · Gateway sub-devices: Indicates that devices of this product cannot connect to IoT Platform directly, but connect to gateway devices and use the communication channels of gateway devices to communication with IoT Platform. · Gateway devices: 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. For more information about gateway devices and sub-devices, see *#unique_7*.

Parameter	Description
Gateway Connection Protocol	Select a protocol for sub-device and gateway communication. · Custom: Indicates that you want to use another protocol as the connection protocol for sub-device and gateway communication. · Modbus: Indicates that the communication protocol between sub-devices and gateways is Modbus. · OPC UA: Indicates that the communication protocol between sub-devices and gateways is OPC UA. · ZigBee: indicates that the communication protocol between sub-devices and gateways is ZigBee. · BLE: indicates that the communication protocol between sub-devices and gateways is BLE.
Network Connection Method	Select a network connection method for the devices: · WiFi · Cellular (2g/3g/4G) · Ethernet · Other
Data Type	Select a format in which devices exchange data with IoT Platform. Options are ICA Standard Data Format (Alink JSON) and Do not parse/Custom. · ICA Standard Data Format (Alink JSON): The standard data format defined by IoT Platform for device and IoT Platform communication. · Do not parse/Custom: If you want to customize the serial data format, select Do not parse/Custom. Custom formatted data must be converted to Alink JSON script by *#unique_8*so that your devices can communicate with the IoT Platform.
Authentication Mode	Currently, only Device Secret is supported.
Product Description	Describe the product information. You can enter up to 100 characters.

After the product is created successfully, you are automatically redirected to the Products page.

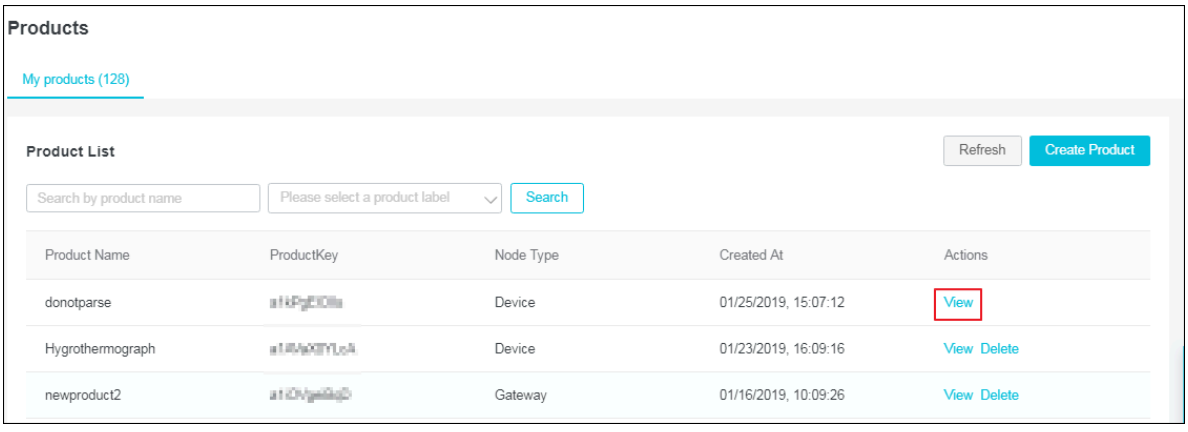
What's next

1. To configure features for a product (such as *Notifications*, *TSL (Define Feature)*, and *Service Subscription*), go to the product list, find the target product and then click its corresponding View button.
2. Register devices on IoT Platform.
3. Develop your physical devices by referring to *Developer Guide (Devices)*.
4. To publish a product, go to the product details page and click Publish.



Note that before you publish a product, you must make sure that you have configured all the correct information for the product, have completed debugging the features, and have verified that it meets the criteria for being published.

When the product status is Published, you can view the product information but cannot modify or delete the product.



To cancel the publishing of a product, click Cancel Publishing.

3 Create devices

3.1 Create multiple devices at a time

A product is a collection of devices. After you create products, you can create specific devices for the product models. You can create one device or multiple devices at a time. This topic explains how to create multiple devices at a time.

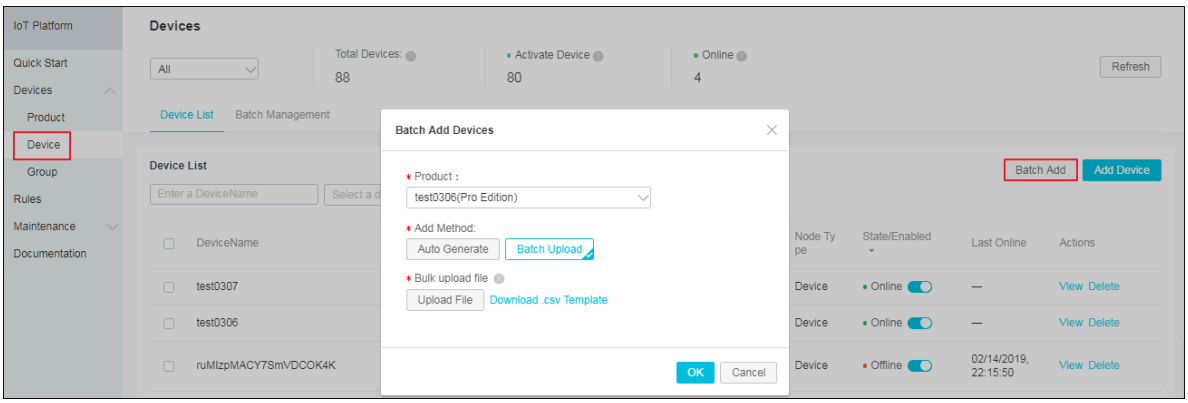
Procedure

1. **Log on to the *IoT Platform console*.**
2. **In the left-side navigation pane, click Devices > Device, and then click Batch Add.**
3. **Select a product that you have created. The devices to be created will be assigned with the features of the selected product.**
4. **Select how the devices are to be named. Two methods:**
   - **Auto Generate: You do not specify names for the devices that you want to create. You only specify the number of devices, and the system automatically generates names for the devices.**
   - **Batch Upload: You specify a name for each device you want to create. Under Upload File, click Download .csv Template to download the naming template. Enter device names in the template table and save the file. Then, click Upload File to upload the naming file.**

Note:

- **Device names must be 4-32 characters in length, and can contain English letters, digits, hyphens, underscores, @ symbols, dots, and colons.**
- **Each device name must be unique in the product.**
- **A file can include up to 1,000 names.**

- The size of the file cannot exceed 2 MB.



5. Click OK to start batch device creation.
6. After the devices are successfully created, click Download Device Certificate to download the file containing the information of created devices.

Result

On the Batch Management tab page of Devices page, you can:

- Click View Details to view the detailed information of the devices.
- Click Download CSV to download the certificates of the devices.

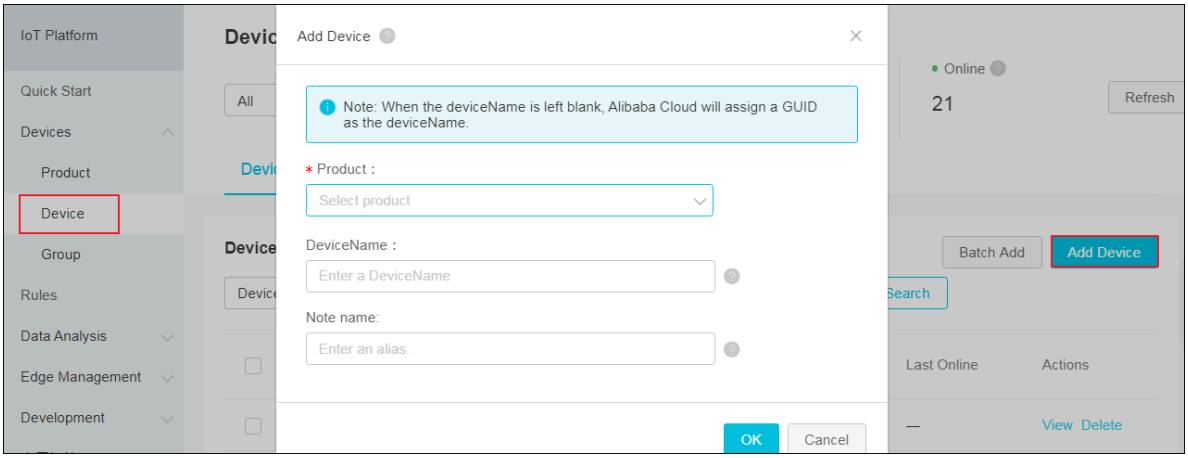
3.2 Create a device

A product is a collection of devices. After you have created a product, you must register devices under the product with IoT Platform. You can create devices individually or create multiple devices at one time. This topic describes how to create devices individually.

Procedure

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, choose Devices > Device, and then click Add Device.

3. In the Add Device dialog box, enter the device information and click OK.



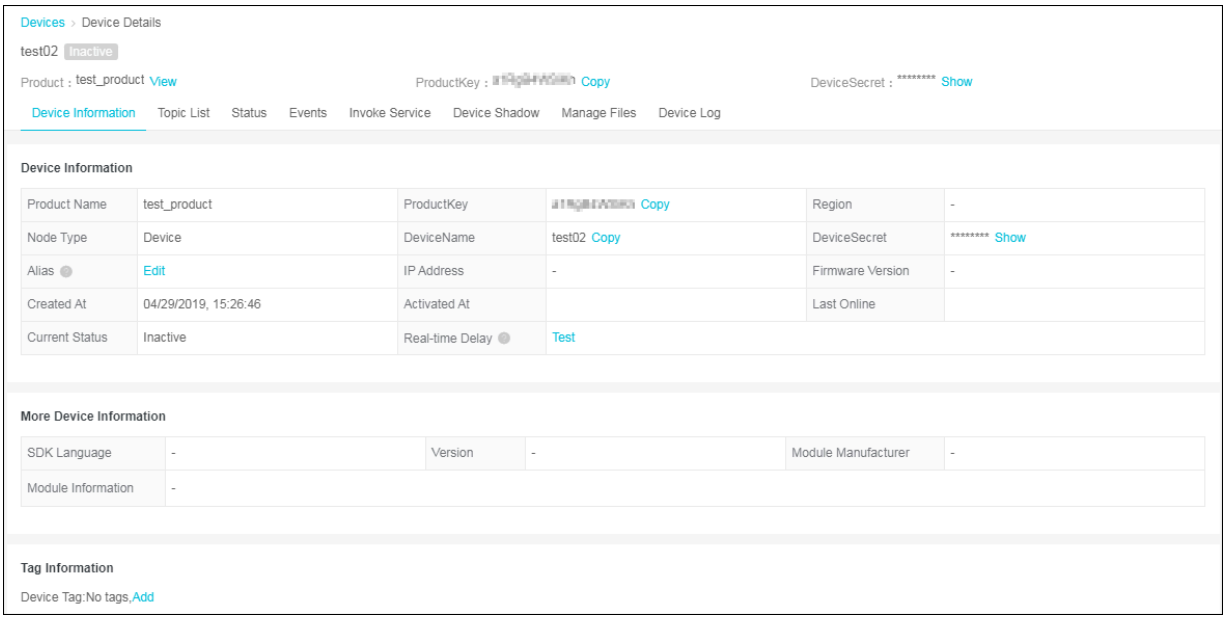
Parameter	Description
Product	Select a product. The device to be created will be assigned the features and properties of the selected product. **Note:** If the product is associated with another platform, make sure that your account has sufficient activation codes to create the device.
DeviceName	Set the device name. If you left this parameter empty, the system automatically generates a device name that contains numbers and letters. · The device name is unique within the product. · The device name must be 4 to 32 characters in length and can contain letters, numbers, and special characters. The supported special characters are hyphens (-), underscores (_), at signs (@), periods (.) , and colons (:).
Note name	Set the alias. The alias must be 4 to 64 characters in length and can contain Chinese characters, letters, numbers, and underscores (_). One Chinese character is counted as two characters.

Result

After the device is created, the View Device Certificate dialog box appears automatically. You can view and copy the device certificate information. A device certificate is the authentication certificate of a device when the device is communicating with IoT Platform. It contains *three key fields*: ProductKey, DeviceName, and DeviceSecret.

Parameter	Description
ProductKey	The key of the product to which the device belongs. It is the GUID that is issued by IoT Platform to the product.
DeviceName	The unique identifier of the device within the product. A device uses the DeviceName and the ProductKey as the device identifier to authenticate to and communicate with IoT Platform.
DeviceSecret	The device key issued by IoT Platform for device authentication and encryption. It must be used in pairs with the DeviceName.

You can also click View next to the newly created device on the Device List page. On the Device Details page, click the Device Information tab to view device information.



What's next

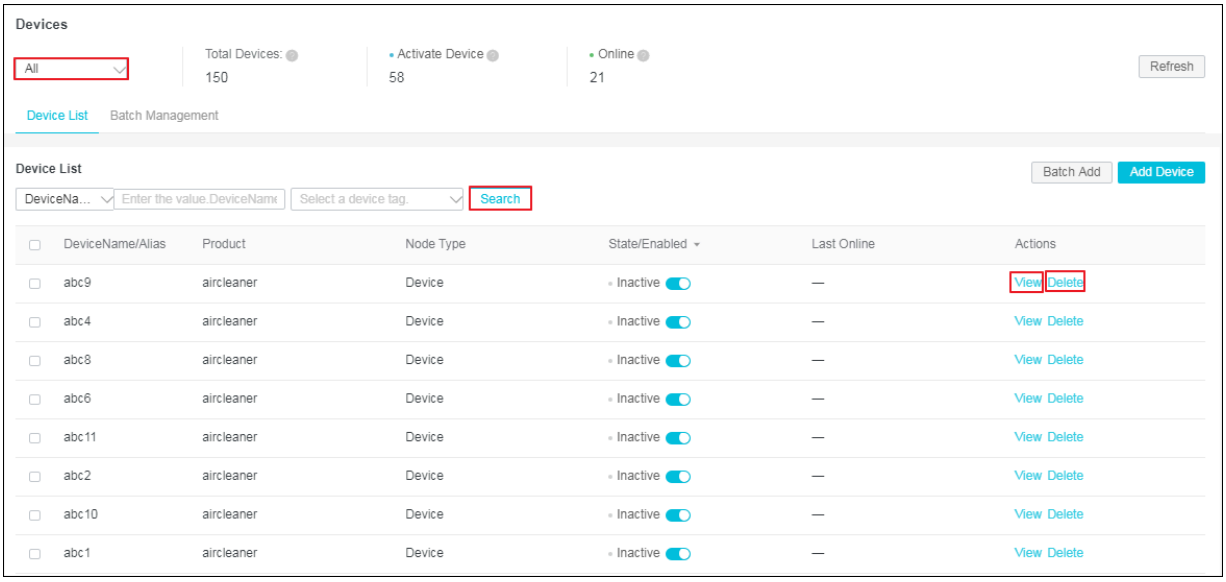
Follow instructions in *Device development documentation* to develop the device SDK.

3.3 Manage devices

After you create a device in IoT Platform, you can manage or view device information in the IoT Platform console.

Manage devices of an account

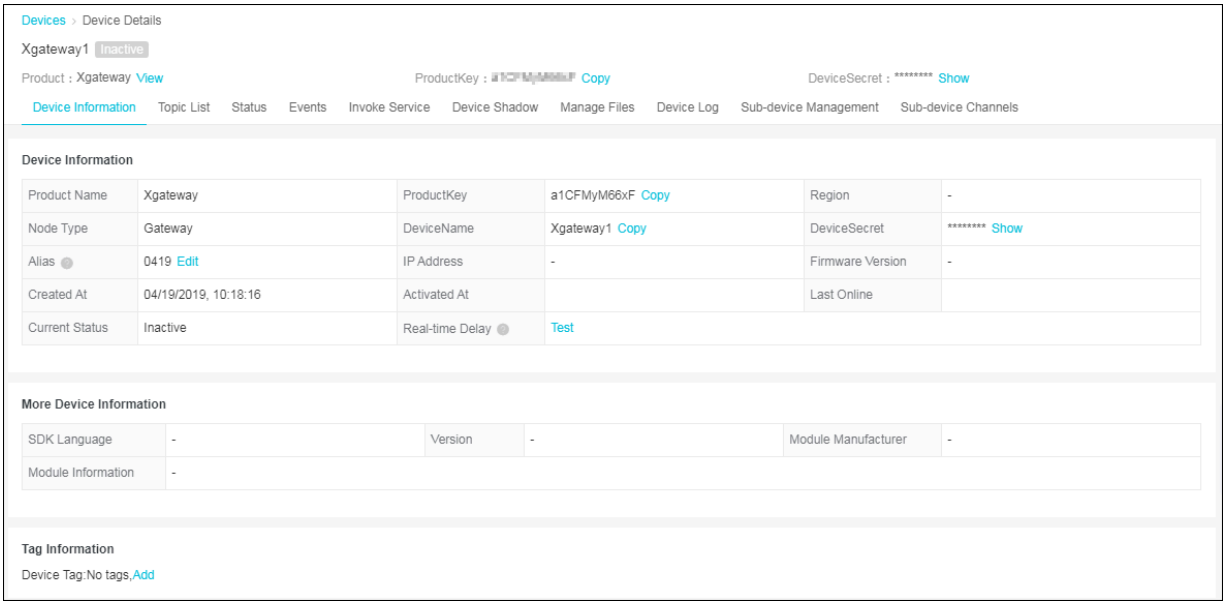
From the left-side navigation pane, choose Devices > Device. The Devices page appears.



Task	Procedure
View devices under a specific product	Select a product in the upper-left corner of the page.
Search for a device	Enter a device name, note name, or device tag to search for a device. Fuzzy search is supported.
View detailed information about a device	Click View next to the corresponding device.
Delete a device	Click Delete next to the corresponding device. **Note:** After a device is deleted, the device certificate becomes invalid and the data about this device in IoT Platform is deleted.

View detailed information about a device

In the device list, click View next to the corresponding device. The Device Details page appears.



Task	Procedure
Activate the device	The Inactive status indicates that the device is not connected to IoT Platform. To develop the device and activate the device, see *Download device SDKs*.
View device information	View the basic information about the device, including device certificate information, *firmware information*, *extended information*, and tag information.
View device data	· On the Status tab page, view the latest values, data records, and desired values of properties. · On the Events tab page, view the records about device reported events. · On the Invoke Service tab page, view the service call records.
View device log	On the Device Log tab page, click Read Now to view the device log information. The information include device activities, upstream messages, downstream messages, TSL data, and QoS=1 message contents. For more information about device logs, see *Device log*.

4 Authenticate devices

To secure devices, IoT Platform provides certificates for devices, including product certificates (ProductKey and ProductSecret) and device certificates (DeviceName and DeviceSecret). A device certificate is a unique identifier used to authenticate a device. Before a device connects to IoT Hub through a protocol, the device reports the product certificate or the device certificate, depending on the authentication method. The device can connect to IoT Platform only when it passes authentication. IoT Platform supports various authentication methods to meet the requirements of different environments.

IoT Platform supports the following authentication methods:

- **Unique-certificate-per-device authentication: Each device has been installed with its own unique device certificate.**
- **Unique-certificate-per-product authentication: All devices under a product have been installed with the same product certificate.**
- **Sub-device authentication: This method can be applied to sub-devices that connect to IoT Platform through the gateway.**

These methods have their own advantages in terms of accessibility and security. You can choose one according to the security requirements of the device and the actual production conditions. The following table shows the comparison among these methods.

Table 4-1: Comparison of authentication methods

Items	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
Information written into the device	ProductKey, DeviceName, and DeviceSecret	ProductKey and ProductSecret	ProductKey
Whether to enable authentication in IoT Platform	No. Enabled by default.	Yes. You must enable dynamic register.	Yes. You must enable dynamic register.

Items	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
DeviceName pre-registration	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes.
Certificate installation requirement	Install a unique device certificate on every device. The safety of every device certificate must be guaranteed.	Install the same product certificate on all devices under a product. Make sure that the product certificate is safely kept.	Install the same product certificate into all sub-devices. The security of the gateway must be guaranteed.
Security	High	Medium	Medium
Upper limit for registrations	Yes. A product can have a maximum of 500,000 devices.	Yes. A product can have a maximum of 500,000 devices.	Yes. A maximum of 200 sub-devices can be registered with one gateway.
Other external reliance	None	None	Security of the gateway.

4.1 Authenticate devices

To secure devices, IoT Platform provides certificates for devices, including product certificates (ProductKey and ProductSecret) and device certificates (DeviceName and DeviceSecret). A device certificate is a unique identifier used to authenticate a device. Before a device connects to IoT Hub through a protocol, the device reports the product certificate or the device certificate, depending on the authentication method. The device can connect to IoT Platform only when it passes authentication. IoT Platform supports various authentication methods to meet the requirements of different environments.

IoT Platform supports the following authentication methods:

- Unique-certificate-per-device authentication: Each device has been installed with its own unique device certificate.

- **Unique-certificate-per-product authentication: All devices under a product have been installed with the same product certificate.**
- **Sub-device authentication: This method can be applied to sub-devices that connect to IoT Platform through the gateway.**

These methods have their own advantages in terms of accessibility and security. You can choose one according to the security requirements of the device and the actual production conditions. The following table shows the comparison among these methods.

Table 4-2: Comparison of authentication methods

Item	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
Information written into the device	ProductKey, DeviceName, and DeviceSecret	ProductKey and ProductSecret	ProductKey
Whether to enable authentication in IoT Platform	No. Enabled by default.	Yes. You must enable dynamic register.	Yes. You must enable dynamic register.
DeviceName pre-registration	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes. You need to make sure that the specified DeviceName is unique under a product.	Yes.
Certificate installation requirement	Install a unique device certificate on every device. The safety of every device certificate must be guaranteed.	Install the same product certificate on all devices under a product. Make sure that the product certificate is safely kept.	Install the same product certificate into all sub-devices. The security of the gateway must be guaranteed.
Security	High	Medium	Medium
Upper limit for registrations	Yes. A product can have a maximum of 500,000 devices.	Yes. A product can have a maximum of 500,000 devices.	Yes. A maximum of 1500 sub-devices can be registered with one gateway.

Item	Unique-certificate-per-device authentication	Unique-certificate-per-product authentication	Sub-device authentication
Other external reliance	None	None	Security of the gateway.

4.2 Unique-certificate-per-device authentication

Using unique-certificate-per-device authentication method requires that each device has be installed with a unique device certificate in advance. When you connect a device to IoT Platform, IoT Platform authenticates the ProductKey, DeviceName, and DeviceSecret of the device. After the authentication is passed, IoT Platform activates the device to enable data communication between the device and IoT Platform.

Context

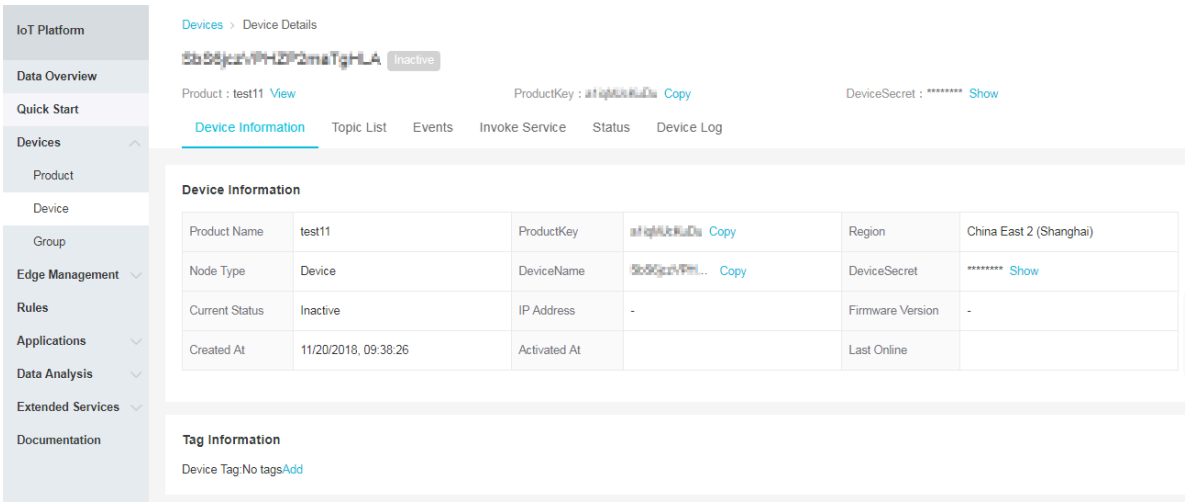
The unique-certificate-per-device authentication method is a secure authentication method. We recommend that you use this authentication method.

Workflow of unique-certificate-per-device authentication:

Procedure

1. **In the *IoT Platform console*, create a product. For more information, see *Create a product* in the User Guide.**

2. Register a device to the product you have created and obtain the device certificate.



3. Install the certificate to the device.

   Follow these steps:

   a) Download a device-side SDK.

   b) Configure the device-side SDK. In the device-side SDK, configure the device certificate (ProductKey, DeviceName, and DeviceSecret).

   c) Develop the device-side SDK based on your business needs, such as OTA development, sub-device connection, TSL-based device feature development, and device shadows development.

   d) During the production process, install the developed device SDK to the device.

4. Power on and connect the device to IoT Platform. The device will initiate an authentication request to IoT Platform using the unique-certificate-per-product method.

5. IoT Platform authenticates the device certificate. After the authentication is passed and the connection with IoT Platform has been established, the device can communicate with IoT Platform by publishing messages to topics and subscribing to topic messages.

4.3 Unique-certificate-per-product authentication

Using unique-certificate-per-product authentication method requires that devices of a product have been installed with a same firmware in which a product certificate (ProductKey and ProductSecret) has been installed. When a device initiates an activation request, IoT Platform authenticates the product certificate

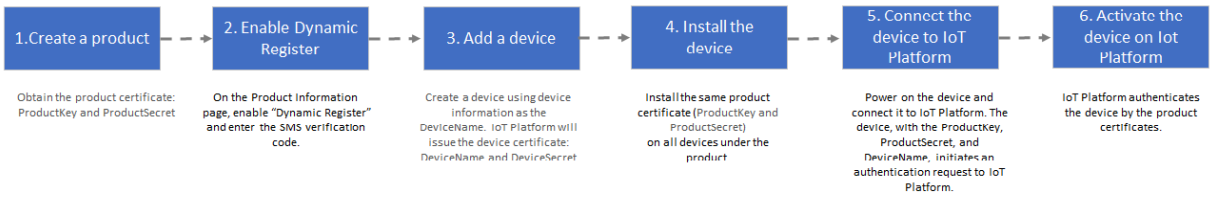
of the device. After the authentication is passed, IoT Platform assigns the corresponding DeviceSecret to the device.

Context

Note:

- This authentication method has risks of product certificate leakage because all devices of a product are installed with the same firmware. On the Product Details page, you can disable `Dynamic Registration` to reject authentication requests from new devices.
- The unique-certificate-per-product method is used to obtain the DeviceSecret of devices from IoT Platform. The DeviceSecret is only issued once. The device stores the DeviceSecret for future use.

Workflow of unique-certificate-per-product authentication:

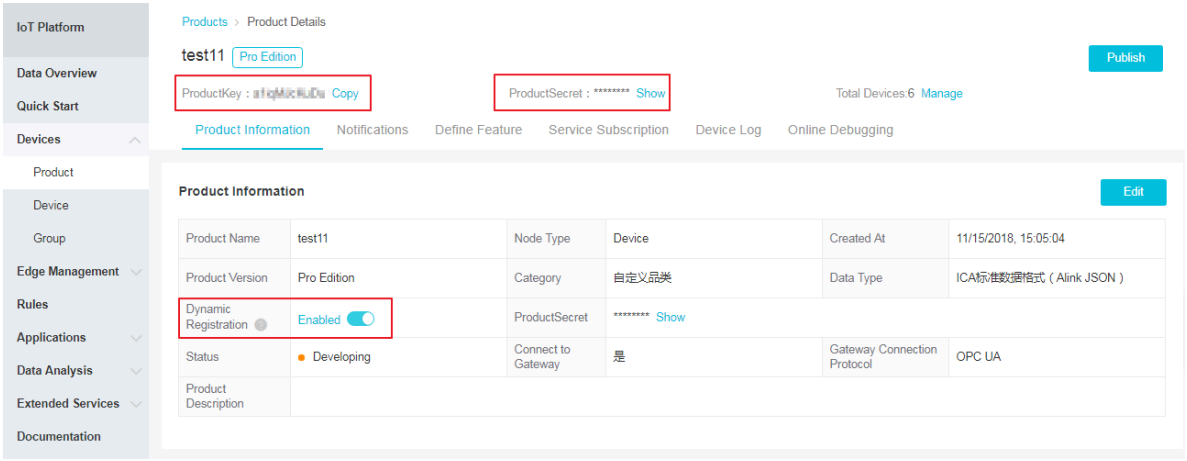


Procedure

1. In the *IoT Platform console*, create a product. For more information, see *Create a product* in the User Guide.
2. On the Product Details page, enable `Dynamic Registration`. IoT Platform sends an SMS verification code to confirm your identity.

Note:

If Dynamic Registration is not enabled when devices initiate activation requests, IoT Platform rejects the activation requests. Activated devices are not affected.



3. **Register a device. The status of a newly registered device is `Inactive`.**

   IoT Platform authenticates the DeviceName when a device initiates an activation request. We recommend that you use an identifier that can be obtained directly from the device, such as the MAC address, IMEI or serial number, as the DeviceName.

4. **Install the product certificate to the device.**

   Follow these steps:

   a) **Download a device-side SDK.**

   b) **Configure the device-side SDK to use the unique-certificate-per-product authentication method. In the device-side SDK, configure the product certificate (ProductKey and ProductSecret).**

   c) **Develop the device-side SDK based on your business needs, such as OTA development, sub-device connection, TSL-based device feature development, and device shadows development.**

   d) **During the production process, install the developed device SDK to the device.**

5. **Power on the device and connect the device to the network. The device sends an authentication request to IoT Platform to perform unique-certificate-per-product authentication.**

6. **After the product certificate has been authenticated by IoT Platform, IoT Platform dynamically assigns the corresponding DeviceSecret to the device. Then, the device has obtained its device certificate (ProductKey, DeviceName, and DeviceSecret) and can connect to IoT Platform. After the connection with**

IoT Platform has been successfully established, the device can communicate with IoT Platform by publishing messages to topics and subscribing to topic messages.

Note:
IoT Platform dynamically assigns DeviceSecret to devices only for the first activation of devices. If you want to reinitialize a device, go to IoT Platform console to delete the device and repeat the procedures to register and activate a device.

5 Topics

The cloud and devices communicate with each other in IoT Platform through topics. The device reports messages to a specified topic and subscribes to messages from the topic. IoT Platform sends commands to topics, and subscribes to specific topics to obtain device information.

5.1 What is a topic?

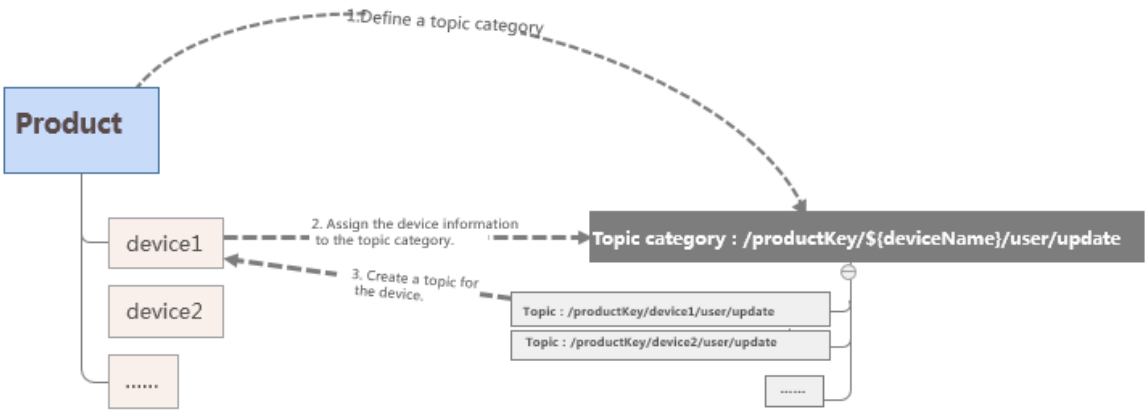
A server and a device communicate with each other in IoT Platform through topics. Topics are associated with devices, and topic categories are associated with products. A topic category of a product is automatically applied to all devices under the product to generate device-specific topics for message communication.

Topic category

To simplify authorization and facilitate communication between devices and IoT Platform, topic categories were introduced. A topic category is a set of topics within the same product. For example, topic category `/${YourProductKey}/${YourDeviceName}/user/update` is a set that contains the following two topics: `/${YourProductKey}/device1/user/update` and `/${YourProductKey}/device2/user/update`.

After a device is created, all topic categories of the product are automatically applied to the device. You do not need to assign topics to each individual device.

Figure 5-1: Automatically create a topic



Descriptions for topic categories:

- **A topic category uses a forward slash (/) to separate elements in different hierarchical levels. A topic category contains the following fixed elements: `${YourProductKey}` indicates the product identifier; `${YourDeviceName}` indicates the device name.**
- **Each element name can contain only letters, numbers, and underscores (_). An element in each level cannot be left empty.**
- **A device can have Pub and Sub permissions to a topic. Pub indicates that the device can publish messages to the topic. Sub indicates that the device can subscribe to the topic.**
- **A device must send a `sub` request to IoT Platform to subscribe to a specified topic. If you want the device to unsubscribe from the topic, you must configure the device to send an `unsub` request.**

Topic

A topic category is used for topic definition rather than communication. Only topics can be used for communication.

- **Topics use the same format as topic categories. The difference is that variable `${YourDeviceName}` in the topic category is replaced by a specific device name in the topic.**
- **A topic is automatically derived from the topic category of the product based on the corresponding device name. A topic contains the device name (`DeviceName`) and can be used for data communication only by the specified device. For example, topic `/${YourProductKey}/device1/user/update` belongs to the device named device1. Only device1 can publish messages and subscribe to this topic. Other devices cannot use this topic.**

Supported wildcards

To use the rules engine data forwarding function to forward device data, you must specify the source topic of the messages when writing an SQL statement. When you specify a topic in *setting a forwarding rule*, you can use the following wildcards. One element can contain only one wildcard.

Wildcard	Description
#	Must be set as the last element in the topic. This wildcard can match any element in the current level and sub-levels. For example, in topic `/${YourProductKey}/device1/user /#`, wildcard `#` is added next to the `/user` element to represent all elements after `/user`. This topic can represent `/${YourProductKey}/device1/user/update` and `/${YourProductKey}/device1/user/update/error`.
+	Matches all elements in the current level. For example, in topic `/${YourProductKey}/+/user/update`, the device name element is replaced by wildcard `+` to represent all devices under the product. This topic can represent `/${YourProductKey}/device1/user/update` and `/${YourProductKey}/device2/user/update`.

System topics and custom topics

IoT Platform supports the following types of topics:

Type	Description
System topics	The system-defined topics. System topics cannot be modified and deleted. System topics include topics used by IoT Platform functions, such as TSL model-related functions and firmware upgrade. For example, topics related to TSL models generally start with `/sys/`. Topics related to firmware upgrade start with `/ota/`. Topics for the device shadow function start with `/shadow/`. **Note:** System topics are not completely displayed in the Topic Categories list and the Topic List. For more information about function-specific topics, see related function documentation.
Custom topics	You can *customize a topic category* on the Topic Categories tab page according to your business requirements. The topic categories you have customized for the product will be automatically applied to all devices under the product.

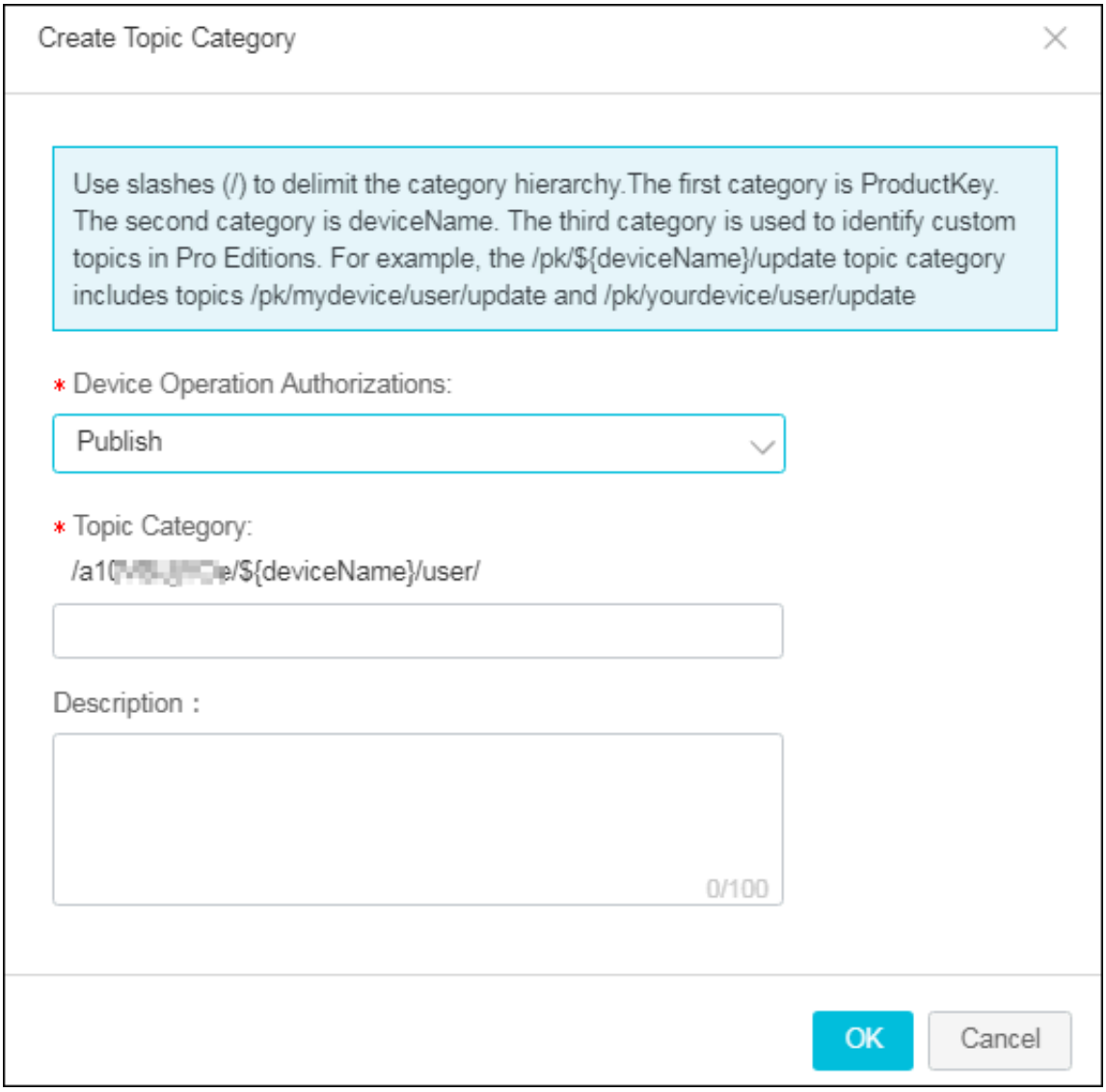
5.2 Create a topic category

This article introduces how to create a topic category for a product. Topic categories will be automatically assigned to devices of the product.

Procedure

1. **Log on to the *IoT Platform console*.**
2. **In the left-side navigation pane, click Devices > Product**
3. **On the Products page, find the product for which you want to create a topic category, and click View in the operation column.**
4. **On the Product Details page, click Topic Categories > Create Topic Category.**

5. **Define a topic category.**



   - **Device Operation Authorizations: Indicates the operations that devices can perform on the topics of this topic category. You can select from Publish, Subscribe, and Publish and Subscribe.**
   - **Topic Category: Enter a custom topic category name according to the `Topic Rule` on the page.**
   - **Description: Describes the topic category. You can leave this box empty.**

6. **Click OK.**

Wildcard characters in topic categories

When you create topic categories, you can use wildcards. For more information about wildcards, see *What is a topic?* Supported wildcards:

- **`#`: Includes the category level you enter and all lower levels in topics.**

- `+`: Includes only one category level in topics, and not lower levels.

Note:

When you want to create topic categories with wildcards, note that:

- **Only topics with** `Device Operation Authorizations` **as** `Subscription` **support wildcards.**
- `#` **can only be at the end of topics.**
- **For topics with wildcard characters, you cannot click Publish to publish messages on the Topic List tab page of devices.**

6 Protocols for connecting devices

6.1 Use CoAP protocol

6.1.1 CoAP standard

Protocol version

IoT Platform supports the Constrained Application Protocol (CoAP) [RFC7252]. For more information, see *RFC 7252*.

Channel security

IoT Platform uses Datagram Transport Layer Security (DTLS) V1.2 to secure channels. For more information, see *DTLS v1.2*.

Open-source client reference

For more information, see *http://coap.technology/impls.html*.

> **Note:**
> **If you use third-party code, Alibaba Cloud does not provide technical support.**

Alibaba Cloud CoAP agreement

- **Do not use a question mark (?) to set a parameter.**
- **Resource discovery is not supported.**
- **Only the User Datagram Protocol (UDP) is supported, and DTLS must be used.**
- **Follow the Uniform Resource Identifier (URI) standard, and keep CoAP URI resources consistent with Message Queuing Telemetry Transport (MQTT)-based topics. For more information, see *MQTT standard*.**

6.1.2 Establish connections over CoAP

IoT Platform supports connections over CoAP. CoAP is suitable for resource-constrained, low-power devices, such as NB-IoT devices. This topic describes how

to connect devices to IoT Platform over CoAP and two supported authentication methods, which are DTLS and symmetric encryption.

Use the symmetric encryption method

1. **Connect to the CoAP server. The endpoint address is** `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`.

   Note:

   - `${YourProductKey}`**: Replace it with the ProductKey value of the device.**
   - `${port}`**: The port number. Set the value to 5682.**

2. **Authenticate the device.**

   Request message:

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"a1NUjcV****","deviceName":"ff1a11e7c0
8d4b3db2b1500d8e0e55","clientId":"a1NUjcV****&ff1a11e7c08d4b3db2b1
500d8e0e55","sign":"F9FD53EE0CD010FCA40D14A9FE******", "seq":"10"}
```

Table 6-1: Parameter description

Parameter	Description
Method	**The request method. The supported method is POST.**
URL	`/auth`**.**
Host	**The endpoint address. The format is** *`${YourProductKey}`*`.coap.cn-shanghai.link.aliyuncs.com`**. Replace** *`${YourProductKey}`* **with the ProductKey value of the device.**
Port	**The port number. Set the value to** `5682`**.**
Accept	**The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.**
Content-Format	**The encoding format of the data that the device sends to IoT Platform. Currently, application/json and application/cbor are supported.**

Parameter	Description
payload	The device information for authentication, in JSON format. For more information, see the following table *payload parameters*.

Table 6-2: payload parameters

Parameter	Required	Description
productKey	Yes	The unique identifier issued by IoT Platform to the product. You can obtain this information on the device details page in the IoT Platform console.
deviceName	Yes	The device name that you specified, or is generated by IoT Platform, when you registered the device. You can obtain this information on the device details page in the IoT Platform console.
ackMode	No	The communication mode. Options: · 0: After receiving a request from the device, the server processes data and then returns the result with an acknowledgment (ACK). · 1: After receiving a request from the device, the server immediately returns an ACK and then starts to process data. After the data processing is complete, the server returns the result. The default value is 0.

Parameter	Required	Description
sign	Yes	Signature. The signature algorithm is `hmacmd5(DeviceSecret,content)`. The value of `content` is a string that is built by sorting and concatenating all the parameters (except `version`, `sign`, `resources`, and `signmethod`) that need to be submitted to the server in alphabetical order, without any delimiters. Signature calculation example: `sign= hmac_md5(mRPVdzSMu2nVBxzK77ER PIMxSYIv****, clientIda1NUjcV****& ff1a11e7c08d4b3db2b1500d8e0e55 deviceNameff1a11e7c08d4b3db2b1 500d8e0e55productKeya1NUjcV**** seq10timestamp1524448722000)`
signmethod	No	The algorithm type. The supported types are hmacmd5 and hmacsha1. The default value is hmacmd5.
clientId	Yes	The device identifier, which can be any string up to 64 characters in length. We recommend that you use the MAC address or the SN code of the device as the clientId.
timestamp	No	The timestamp. Currently, timestamp is not verified.

Response example:

```
{"random":"ad2b3a5eb51d64f7","seqOffset":1,"token":"MZ8m37hp01
w1SSqoDFzo0010500d00.ad2b"}
```

Table 6-3: Response parameters

Parameter	Description
random	The encryption key used for data communication.
seqOffset	The authentication sequence offset.
token	The returned token after the device is authenticated.

3. **The device sends data.**

   Request message:

```
POST /topic/${topic}
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(token), 2089(seq)
```

Table 6-4: Request parameters

Parameter	Required	Description
Method	**Yes**	**The request method. The supported request method is POST.**
URL	**Yes**	**The format is `/topic/${topic}`. Replace the variable *`${topic}`* with the device topic used by the device to publish data.**
Host	**Yes**	**The endpoint address. The format is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`. Replace the variable *`${YourProductKey}`* with the ProductKey value.**
Port	**Yes**	**The port number. Set the value to `5682`.**
Accept	**Yes**	**The encoding format of the data which is to be received by the device . Currently, application/json and application/cbor are supported.**
Content-Format	**Yes**	**The encoding format of the data which is sent by the device. Currently, application/json and application/cbor are supported.**
payload	**Yes**	**The encrypted data that is to be sent. Encrypt the data using the Advanced Encryption Standard (AES) algorithm.**

Parameter	Required	Description
CustomOptions	Yes	The option value can be 2088 and 2089, which are described as follows: · 2088: Indicates the token. The value is the token returned after the device is authenticated. Note: Token information is required every time the device sends data. If the token is lost or expires, initiate a device authentication request again to obtain a new token. · 2089: Indicates the sequence. The value must be greater than the seqOffset value that is returned after the device is authenticated, and must be a unique random number. Encrypt the value with AES. Response message for option `number:2090 (IoT Platform message ID)`

After a message has been sent to IoT Platform, a status code and a message ID are returned.

Establish DTLS connections

1. Connect to the CoAP server. The endpoint address is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com:${port}`.

   Note:

   - *`${YourProductKey}`*: Replace it with the ProductKey value of the device.
   - *`${port}`*: The port number. Set the port number to 5684 for DTLS connections.

2. Download the *root certificate*.

3. **Authenticate the device. Call `auth` to authenticate the device and obtain the device token. Token information is required when the device sends data to IoT Platform.**

   Request message:

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"ZG1EvTEa7NN","deviceName":"NlwaSPXsCp
TQuh8FxBGH","clientId":"mylight1000002","sign":"bccb3d2618afe74b3eab
12b94042f87b"}
```

   For more information about parameters (except for `Port` parameter, where the port for this method is 5684) and payload content, see *Parameter description*.

   Response example:

```
response: {"token":"f13102810756432e85dfd351eeb41c04"}
```

Table 6-5: Return codes

Code	Message	Payload	Description
2.05	Content	The token is contained in the payload if the authentication has passed.	The request is successful.
4.00	Bad Request	no payload	The payload in the request is invalid.
4.01	Unauthorized	no payload	The request is unauthorized.
4.03	Forbidden	no payload	The request is forbidden.
4.04	Not Found	no payload	The requested path does not exist.
4.05	Method Not Allowed	no payload	The request method is not allowed.
4.06	Not Acceptable	no payload	The value of Accept parameter is not in a supported format.

Code	Message	Payload	Description
4.15	Unsupported Content-Format	no payload	The value of Content-Format parameter is not in a supported format.
5.00	Internal Server Error	no payload	The authentication request is timed out or an error occurred on the authentication server.

4. **The device sends data.**

   The device publishes data to a specified topic.

   In the IoT Platform console, on the Topic Categories tab page of the product, you can create topic categories.

   Currently, only topics with the permission to publish messages can be used for publishing data, for example, `/${YourProductKey}/${YourDeviceName}/pub`**. Specifically, if a device name is device, and its product key is a1GFjLP3xxC, the device can send data through the address** `a1GFjLP3xxC.coap.cn-shanghai.link.aliyuncs.com:5684/topic/a1GFjLP3xxC/device/pub`**.**

   Request message:

```
POST /topic/${topic}
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
CustomOptions: number:2088(token)
```

Table 6-6: Request parameters

Parameter	Required	Description
Method	Yes	The request method. The supported request method is POST.
URL	Yes	`/topic/${topic}` Replace the variable `${topic}` with the device topic which will be used to publish data.
Host	Yes	The endpoint address. The format is *`${YourProductKey}`*`.coap.cn-shanghai.link.aliyuncs.com`. Replace *`${YourProductKey}`* with the ProductKey value of the device.

Parameter	Required	Description
Port	Yes	Set the value to 5684.
Accept	Yes	The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.
Content-Format	Yes	The encoding format of the data that the device sends to IoT Platform. Currently, application/json and application/cbor are supported.
CustomOptions	Yes	· Number: 2088. · The value of token is the token information returned after *auth* is called to authenticate the device. **Note:** Token information is required every time the device sends data. If the token is lost or expires, initiate a device authentication request again to obtain a new token.

6.2 Use HTTP protocol

6.2.1 HTTP standard

HTTP protocol versions

- Supports Hypertext Transfer Protocol (HTTP) version 1.0. For more information, see *RFC 1945*
- Supports HTTP version 1.1. For more information, see *RFC 2616*

Channel security

Uses Hypertext Transfer Protocol Secure (HTTPS) to guarantee channel security.

- Does not support passing parameters with question marks (?).
- Resource discovery is currently not supported.
- Only HTTPS is supported.
- The URI standard, the HTTP URI resources, and the MQTT topic must be consistent. See *MQTT standard*.

6.2.2 Establish connections over HTTP

IoT Platform supports HTTP connections, and only the HTTPS protocol is supported. This topic describes how to connect devices to IoT Platform over HTTP.

Restrictions

- **HTTP communications are applicable to simple data report scenarios.**
- **The HTTP server endpoint is** `https://iot-as-http.cn-shanghai.aliyuncs.com`**.**
- **Only the China (Shanghai) region supports HTTP communication.**
- **Only the HTTPS protocol is supported.**
- **The standards for HTTPS-based topics are the same as the standards for MQTT-based topics in** *MQTT standards***. Devices connect to IoT Platform over HTTP and send data to IoT Platform by using** `https://iot-as-http.cn-shanghai.aliyuncs.com/topic/${topic}`**. The value of** `${topic}` **can be the same topics used for MQTT communications. You cannot specify parameters in the format of** `?query_String=xxx`**.**
- **The size of data from devices is limited to 128 KB.**
- **Only POST method is supported.**
- **The value of** `Content-Type` **in the HTTP header of an authentication request must be** `application/json`**.**
- **The value of** `Content-Type` **in the HTTP header of an upstream data request must be** `application/octet-stream`**.**
- **The token returned for the device authentication will expire after a certain period of time. Currently, the token is valid for seven days. Make sure that you understand any negative impact that token expiration will have on your business .**

Procedure

The communication process includes performing device authentication to obtain a device token and using the obtained token for data reporting.

1. **Authenticate the device to obtain the device token.**

   Endpoint: `https://iot-as-http.cn-shanghai.aliyuncs.com`

   Authentication request:

   ```
   POST /auth HTTP/1.1
   Host: iot-as-http.cn-shanghai.aliyuncs.com
   ```

```
Content-Type application/json
body: {"version":"default","clientId":"mylight1000002","signmethod
":"hmacsha1","sign":"4870141D4067227128CBB4377906C3731CAC221C","
productKey":"ZG1EvTEa7NN","deviceName":"NlwaSPXsCpTQuh8FxBGH","
timestamp":"1501668289957"}
```

Table 6-7: Parameters

Parameter	Description
Method	The request method. The supported method is POST.
URL	The URL of the `/auth` request. Only HTTPS is supported.
Host	The endpoint: `iot-as-http.cn-shanghai.aliyuncs.com`.
Content-Type	The encoding format of the upstream data that the device sends to IoT Platform. Only `application/json` is supported. If another encoding format is used, a parameter error is returned.
body	The device information for authentication, in JSON format. For more information, see the following table *Fields in body*.

Table 6-8: Fields in body

Field	Required?	Description
productKey	Yes	The unique identifier of the product to which the device belongs. You can obtain this information from the Device Details page of the IoT platform console.
deviceName	Yes	The device name. You can obtain this information from the Device Details page of the IoT platform console.
clientId	Yes	The client ID, a string of up to 64 characters. We recommend that you use the MAC address or SN code as the client ID.
timestamp	No	The timestamp. A request is valid within 15 minutes after the timestamp is created. The timestamp is in the format of numbers. The value is the number of milliseconds that have elapsed since 00:00, January 1, 1970 (GMT).

Field	Required?	Description
sign	Yes	The signature value. The signature algorithm is in the format of `hmacmd5(deviceSecret,content)`. The value of `content` is a string that contains all the parameters to be reported to IoT Platform except `version`, `sign`, and `signmethod`. These parameters are sorted in alphabetical order and spliced without any separators. Signature example: If `clientId=12345`, `deviceName=device`, `productKey=pk`, `timestamp=789`, `signmethod=hmacsha1`, and `deviceSecret=secret`, then the signature algorithm is `hmacsha1("secret","clientId12345deviceNamedeviceproductKeypktimestamp789").toHexString();`. In this example, binary data will be converted to a case-insensitive hexadecimal string.
signmethod	No	The algorithm type. The type can be hmacmd5 or hmacsha1. If you do not specify this parameter, the default value is hmacmd5.
version	No	The version number. If you do not specify this parameter, the value is "default".

Sample response:

```
body:
{
  "code": 0,// The status code
  "message": "success", // The message
  "Info": {
    "token":  "6944e5bfb92e4d4ea3918d1eda3942f6"
  }
}
```

Note:

- Cache the returned token value locally.
- Token information is required each time when the device reports data to IoT Platform. If the token expires, you must re-authenticate the device to obtain a new token.

Table 6-9: Error codes

Code	Message	Description
10000	common error	Unknown error.
10001	param error	A parameter error occurred.
20000	auth check error	An error occurred while authentica ting the device.
20004	update session error	An error occurred while updating the session.
40000	request too many	Too many requests. The throttling policy limits the number of requests.

2. Send data to IoT Platform.

   The device sends data to a specific topic.

   To send data to a custom topic, you must create a topic category on the Topic Categories tab page of the corresponding product in the IoT Platform console. For more information, see *Create a topic category*.

   For example, a topic category is `/${YourProductKey}/${YourDeviceName}/user/pub`. If the device name is device123, and its ProductKey is a1GFjLPXXXX, the device can send data through `https://iot-as-http.cn-shanghai.aliyuncs.com/topic/a1GFjLPXXXX/device123/user/pub`.

   Upstream data request:

```
POST /topic/${topic} HTTP/1.1
Host: iot-as-http.cn-shanghai.aliyuncs.com
password:${token}
Content-Type: application/octet-stream
body: ${your_data}
```

Table 6-10: Parameter description

Parameter	Description
Method	The request method. The supported method is POST.

Parameter	Description
URL	/topic/${topic}. Replace ${topic} with the topic to which data is sent. Only HTTPS is supported.
Host	The endpoint: iot-as-http.cn-shanghai.aliyuncs.com.
password	This parameter is included in the request header. The value of this parameter is the token returned after calling auth to authenticate the device.
Content-Type	The encoding format of the upstream data that the device sends to IoT Platform. Only application/octet-stream is supported. If another encoding format is used, a parameter error is returned.
body	The data content sent to the target topic.

Sample response:

```
body:
{
  "code": 0, // The status code
  "message": "success", // The message
  "Info": {
    "messageId": 892687627916247040,
  }
}
```

Table 6-11: Error codes

Code	Message	Description
10000	common error	Unknown error.
10001	param error	A parameter error occurred.
20001	token is expired	The token has expired. You must call auth to re-authenticate the device and obtain a new token.
20002	token is null	The request header does not contain any token information.
20003	check token error	An error occurred while obtaining identity information according to the token. You must call auth to re-authenticate the device and obtain a new token.
30001	publish message error	An error occurred while reporting data.

Code	Message	Description
40000	request too many	Too many requests. The throttling policy limits the number of requests.

7 Generic protocol SDK

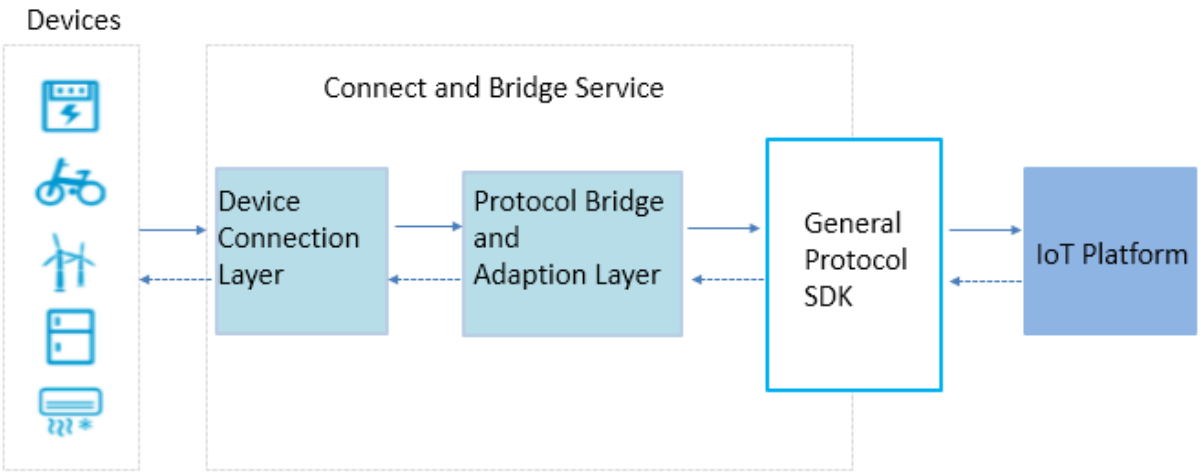
7.1 Overview

Alibaba Cloud IoT Platform supports communication over MQTT, CoAP, or HTTP. Other types of protocols, such as the fire protection agreement GB/T 26875.3-2011, Modbus, and JT808, are not supported. In specific scenarios, some devices may not be able to directly connect to IoT Platform. You must use the generic protocol SDK to build a bridge for your devices or platforms with IoT Platform, so that they can communicate with each other.

Architecture

The generic protocol SDK is a self-adaptive protocol framework. This SDK is used to provide a bridge service for the bi-directional communication between IoT Platform and your devices or platforms.

The following figure shows the architecture.



Scenarios

The generic protocol SDK can be applied to the following scenarios:

- **Your device cannot be directly connected to IoT Platform because of the network or hardware restrictions.**
- **Your device supports only protocols that are not supported by IoT Platform.**

- A connection is already established between the device and your server. You want to connect the device to IoT Platform without modifying the device and protocol.
- The device is directly connected to your server. Additional logic processing is required.

Features

The generic protocol SDK enables the bridge server to communicate with IoT Platform.

Basic features:

- Allows you to manage configurations based on a configuration file.
- Allows you to manage device connections.
- Provides upstream communication capabilities.
- Provides downstream communication capabilities.

Advanced features:

- Allows you to manage configurations based on interfaces.
- Provides interfaces that can be called to report properties, events, and tags.

Terms

Term	Description
device	The device in a real IoT scenario that cannot directly communicate with IoT Platform by using the protocols supported by IoT Platform.
bridge server	The server to which the device is connected. This server uses a specific protocol to communicate with the device and uses the generic protocol SDK to communicate with IoT Platform.
original protocol	The specific protocol used between the device and the bridge server. The generic protocol SDK does not involve the definition and implementation of the original protocol.
original device identifier	The unique identifier used by the device to communicate with the bridge server over the original protocol. Among the generic protocol SDK interface parameters, the `originalIdentity` parameter specifies the identifier of the device's original identity.

Term	Description
device certificate	The device certificate information obtained after you register the device with IoT Platform. The information includes ProductKey, DeviceName, and DeviceSecret. In a scenario that uses the generic protocol, you do not need to install the device certificate on the device. Instead, you must configure the generic protocol SDK file: *devices.conf*. The bridge maps the `originalIdentity` of the device to the device certificate.
bridge certificate	The device certificate information returned after you register the bridge device with IoT Platform. The information includes ProductKey, DeviceName, and DeviceSecret. The bridge certificate uniquely identifies the bridge in IoT Platform.

Development and deployment

1. **Create products and devices.**

   Log on to the IoT Platform console and create products and devices. For more information, see *Create a product* **and** *Create a device* **or** *Create multiple devices at a time*.

   Obtain the device certificate of the bridge. This certificate must be provided when you configure the generic protocol SDK.

   > **Note:**
   > **Bridge is a virtual concept. You can use any device certificate as the certificate information of the bridge.**

2. **Configure the generic protocol SDK.**

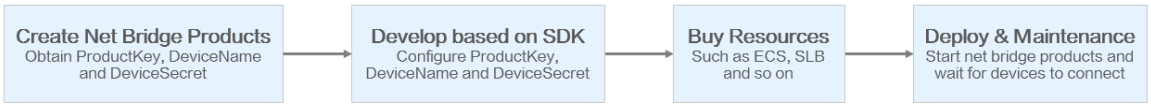
   The generic protocol SDK supports only the Java language. Only JDK 1.8 and later versions are supported.

   For more information about how to configure the generic protocol SDK, see *Use the basic features*.

3. **Deploy the bridge service.**

   You can deploy a developed bridge service on Alibaba Cloud in a scalable manner by using Alibaba Cloud services such as *ECS* and *SLB*. You can also deploy the bridge service in local environment to guarantee secure communication.

   The following figure shows the procedures of using ECS to deploy the bridge service:



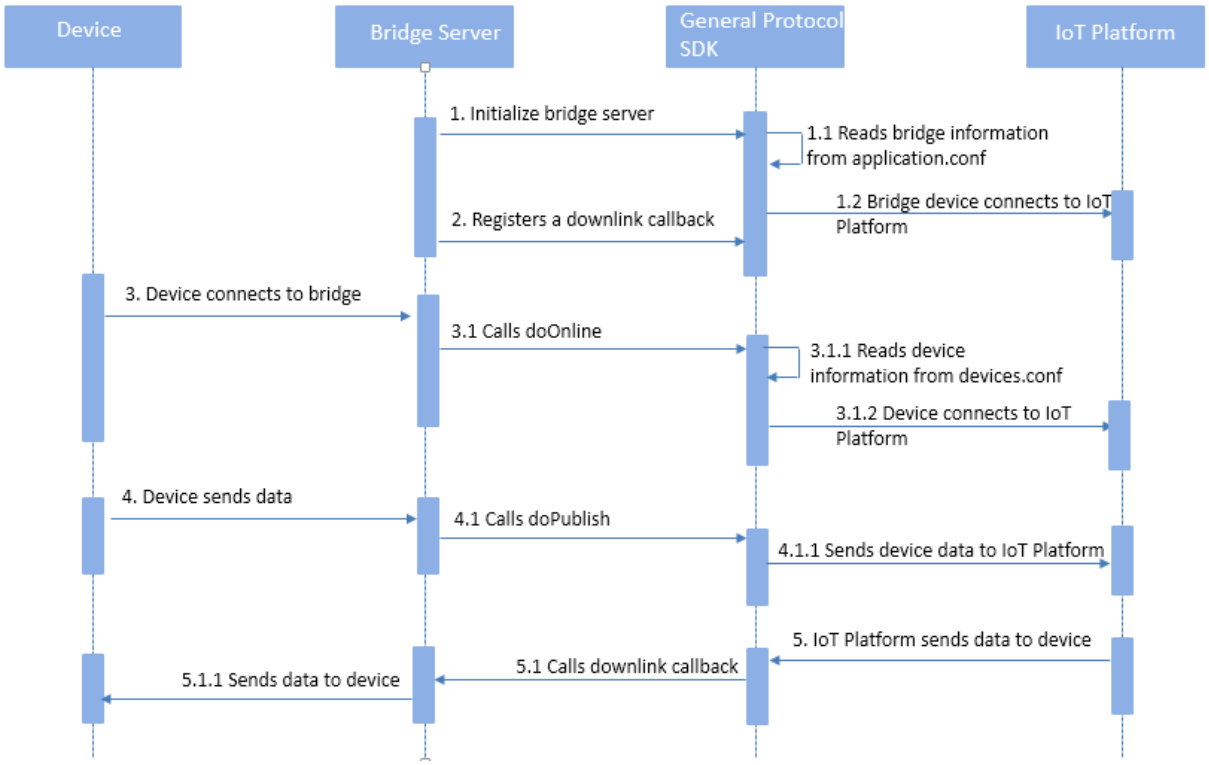
7.2 Use the basic features

Based on the generic protocol SDK, your device can connect to and communicate with Alibaba Cloud IoT Platform by using the bridge service. This topic describes how to configure the generic protocol SDK to implement basic capabilities, including device connection and disconnection and message upstreaming and downstreaming.

See *generic protocol SDK demo* in GitHub.

Flow diagram

The following flow diagram shows the overall process for how to use the generic protocol SDK to connect a device to IoT Platform.



Import the SDK

Add the following dependency in your maven project to import the generic protocol SDK.

```
<dependency>
  <groupId>com.aliyun.openservices</groupId>
  <artifactId>iot-as-bridge-sdk-core</artifactId>
  <version>2.0.0</version>
</dependency>
```

Initialization

- **Initialize the SDK.**

  You must create a `BridgeBootstrap` object and call the `bootstrap` method. After the generic protocol SDK initialization is complete, the SDK reads the bridge information and initiates a request for the bridge to connect to IoT Platform.

  In addition to calling `bootstrap`, you can also register the `DownlinkChannelHandler` callback with the generic protocol SDK to receive downstream messages from IoT Platform.

  Sample code:

```
BridgeBootstrap bridgeBootstrap = new BridgeBootstrap();
bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
    @Override
```

```
    public boolean pushToDevice(Session session, String topic, byte
[] payload) {
        //get message from cloud
        String content = new String(bytes);
        log.info("Get DownLink message, session:{}, {}, {}", session
, topic, content);
        return true;
    }

    @Override
    public boolean broadcast(String topic, byte[] payload) {
        return false;
    }
});
```

- **Configure bridge information.**

  By default, a bridge is configured based on a configuration file. By default, the configuration file is read from `application.conf` under the default resource file path of the Java project (generally `src/main/resources/`). The file is in the format of *HOCON* (JSON superset). The generic protocol SDK uses `typesafe.config` to parse the configuration file.

  You can configure a bridge device either by specifying a bridge device or dynamically registering a bridge device. This topic only describes how to specify a bridge device. For more information about how to dynamically register a bridge device, see *Dynamically register a bridge device*.

  Table 7-1: Bridge configuration parameters

Parameter	Required	Description
productKey	Yes	The key of the product to which the bridge device belongs .

Parameter	Required	Description
deviceName	No	The device name of the bridge device. - You must provide this parameter if you have registered the bridge device in advance and want to configure the device based on the specified device certificate information. - You do not need to provide this parameter if you have not registered the bridge device in advance and want to use the MAC address of the bridge server as the device name to dynamically register a device with IoT Platform.

Parameter	Required	Description
deviceSecret	No	The device secret of the bridge device. - You must provide this parameter if you have registered the bridge device in advance and want to configure the device based on the specified device certificate information. - You do not need to provide this parameter if you choose to dynamically register the bridge device rather than have the bridge device registered in advance.

Parameter	Required	Description
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway service. The bridge device and IoT Platform establish a persistent connection over the HTTP/2 protocol. The endpoint is in the format of `${productKey}.iot-as-http2 .${RegionId}.aliyuncs.com:443`. Replace *${ProductKey}* with the ProductKey of the product to which your bridge device belongs. Replace *${RegionId}* with the ID of the region where your service is located.

Parameter	Required	Description
authEndpoint	Yes	The service URL for device authentication. The device authentication service URL is in the format of `https://iot-auth .${RegionId}.aliyuncs.com/auth/bridge`. Replace *${RegionId}* with the ID of the region where your service is located. For more information about regions, see *Regions and zones*. For example, if the region is China (Shanghai), then the device authentication service address is `https://iot-auth.cn-shanghai.`

Parameter	Required	Description
popClientProfile	No	This parameter must be provided if you use the MAC address of the bridge server to dynamically register the bridge device. For more information, see *Dynamically register a bridge device*.

Use the following format to configure the bridge device certificate:

```
# Server endpoint
http2Endpoint = "https://a1tN7OBmTcd.iot-as-http2.cn-shanghai.
aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/
bridge"

# Gateway device info, productKey & deviceName & deviceSecret
productKey = ${bridge-ProductKey-in-Iot-Plaform}
deviceName = ${bridge-DeviceName-in-Iot-Plaform}
deviceSecret = ${bridge-DeviceSecret-in-Iot-Plaform}
```

Device authentication and connection

- **Configure device connection.**

  The device connection interface in the generic protocol SDK:

```
/**
 * Device authentication
 * @param newSession  Device session information, which is returned
in a downstream callback.
 * @param originalIdentity  The  original identity of the device
 * @return
 */
```

```
public boolean doOnline(Session newSession, String originalIdentity
);
```

When the device is connected to the bridge device, it must pass in a session. When a downstream message is called back, the session is called back to the bridge device. The session contains the original identifier field, so that the bridge device can determine which device the message came from.

In addition, the session also has an optional channel field, which can be designed to store device connection information. For example, your bridge server is built based on Netty. You can use this field to store the channel object correspond ing to the persistent connection of the device. When a message is sent from IoT Platform, the bridge device can directly obtain the channel from the session for subsequent operations. The data type of the channel field is Object. The generic protocol SDK does not process data stored in the channel field. You can also store any device-related information in the channel field according to the scenario.

Sample code for device connection:

```
UplinkChannelHandler uplinkHandler = new UplinkChannelHandler();
//Create a session
Object channel = new Object();
Session session = Session.newInstance(originalIdentity, channel);
//Connect the device to the bridge
boolean success = uplinkHandler.doOnline(session, originalIdentity);
if (success) {
    //If the device is connected, the bridge device accepts new
communication requests from the device.
} else {
    //If the device connection fails, the bridge device rejects
subsequent communication requests, such as disconnection requests.
}
```

- **Map an original identifier to a device certificate.**

  You must configure the mapping between the device certificate and the original identifier of a device. By default, a configuration file is used to configure the mapping. The configuration file is read from *devices.conf* **under the default resource file path of the Java project (generally** *src/main/resources/***). The file is in the format of** *HOCON* **(JSON superset). The generic protocol SDK uses** `typesafe.config` **to parse the configuration file.**

  Use the following format to configure the device certificate information:

```
${device-originalIdentity} {
  prodyctKey : ${device-ProductKey-in-Iot-Plaform}
  deviceName : ${device-DeviceName-in-Iot-Platform}
  deviceSecret : ${device-DeviceSceret-in-Iot-Platform}
```

```
}
```

Parameter	Required	Description
productKey	Yes	The key of the product to which the device belongs.
deviceName	Yes	The device name.
deviceSecret	Yes	The device secret.

Device sends data to IoT Platform

The interface for data upstreaming in the generic protocol SDK:

```
/**
 * Send upstream messages from the device by synchronously calling the
 interface
 * @param originalIdentity  The original identifier of the device
 * @param protocolMsg  The message to be sent, including the topic,
payload, and QoS information
 * @param timeout  The timeout period in seconds
 * @return  Indicates whether the message is sent successfully within
the timeout period
 */
boolean doPublish(String originalIdentity, ProtocolMessage protocolMsg
, int timeout);
/**
 * Send upstream messages from the device by asynchronously calling
the interface
 * @param originalIdentity  The original identifier of the device
 * @param protocolMsg  The message to be sent, including the topic,
payload, and QoS information
 * @return  After this interface is called, CompletableFuture is
returned immediately. The caller can further process this Future.
 */
CompletableFuture<ProtocolMessage> doPublishAsync(String originalId
entity,
                                                  ProtocolMessage
protocolMsg);
```

Sample code:

```
DeviceIdentity deviceIdentity =
    ConfigFactory.getDeviceConfigManager().getDeviceIdentity(
originalIdentity);
ProtocolMessage protocolMessage = new ProtocolMessage();
protocolMessage.setPayload("Hello world".getBytes());
protocolMessage.setQos(0);
protocolMessage.setTopic(String.format("/%s/%s/update",
        deviceIdentity.getProductKey(), deviceIdentity.getDeviceName
()));
//Synchronous sending
int timeoutSeconds = 3;
boolean success = upLinkHandler.doPublish(originalIdentity, protocolMe
ssage, timeoutSeconds);
//Asynchronous sending
```

```
upLinkHandler.doPublishAsync(originalIdentity, protocolMessage);
```

Bridge device pushes data to device

When the bridge device calls the `bootstrap` method, it registers `DownlinkChannelHandler` with the generic protocol SDK. When the generic protocol SDK receives a downstream message, it calls back the `pushToDevice` method in `DownlinkChannelHandler`. You can edit the `pushToDevice` method to configure the bridge device to process downstream messages.

Note:

Do not create a time-consuming logic in the `pushToDevice` method. Otherwise, the thread that receives downstream messages will be blocked. Use the asynchronous transmission if a time-consuming logic or I/O logic exists, for example, sending downstream messages through a persistent connection to the devices.

Sample code:

```
private static ExecutorService executorService  = new ThreadPool
Executor(
    Runtime.getRuntime().availableProcessors(),
    Runtime.getRuntime().availableProcessors() * 2,
    60, TimeUnit.SECONDS,
    new LinkedBlockingQueue<>(1000),
    new ThreadFactoryBuilder().setDaemon(true).setNameFormat("bridge-
downlink-handle-%d").build(),
    new ThreadPoolExecutor.AbortPolicy());
public static void main(String args[]) {
    //Use application.conf & devices.conf by default
    bridgeBootstrap = new BridgeBootstrap();
    bridgeBootstrap.bootstrap(new DownlinkChannelHandler() {
        @Override
        public boolean pushToDevice(Session session, String topic,
byte[] payload) {
            //get message from cloud
            //get downlink message from cloud
            executorService.submit(() -> handleDownLinkMessage(session
, topic, payload));
            return true;
        }
        @Override
        public boolean broadcast(String s, byte[] bytes) {
            return false;
        }
    });
}
private static void handleDownLinkMessage(Session session, String
topic, byte[] payload) {
    String content = new String(payload);
    log.info("Get DownLink message, session:{}, topic:{}, content:
{}", session, topic, content);
    Object channel = session.getChannel();
    String originalIdentity = session.getOriginalIdentity();
```

```
    //for example, you can send the message to device via channel, it
depends on you specific server implementation
}
```

Parameter	Description
Session	A session is transmitted by a device when the device is connecting to the bridge device. A session can be used to identify the device to which the downstream message is sent.
topic	The topic of the downstream message.
payload	The payload of a downstream message in binary format.

Device disconnection

A device is disconnected under the following situations:

- When the bridge device is disconnected from IoT Platform, all connected devices are automatically disconnected from IoT Platform.
- The bridge device reports a disconnection request for a device to IoT Platform.

The interface for bridge device to report device disconnection in the generic protocol SDK:

```
/**
 * Report a disconnection request to IoT Platform for a device
 * @param originalIdentity  The original identifier of the device
 * @return  Indicates whether the message is sent successfully
 */
boolean doOffline(String originalIdentity);
```

Sample code:

```
upLinkHandler.doOffline(originalIdentity);
```

7.3 Use the advanced features

This topic describes how to use the advanced features of the generic protocol SDK. The advanced features include customizing the configuration file path, configuring dynamic bridge registration, calling the data reporting interfaces encapsulated in the generic protocol SDK to report properties, events, and tags.

Customize configurations

By default, the configuration file of a bridge device and the mapping configuration file of the device certificate are read from *application.conf* and *devices.conf*,

respectively, under a fixed path. The generic protocol SDK allows you to customize configurations. Before you call `bootstrap`, call the *`ConfigFactory.init`* method to customize the path of a configuration file. You can also customize an instance to implement the corresponding interface.

Sample code to customize configurations:

```
//Define config
//You can specify the location path of config files
//or you can create an instance and implement the corresponding
interface
//Config.init() must be called before bridgeBootstrap.bootstrap()
ConfigFactory.init(
    ConfigFactory.getBridgeConfigManager("application-self-define.conf
"),
    selfDefineDeviceConfigManager);
bridgeBootstrap.bootstrap();

private static DeviceConfigManager selfDefineDeviceConfigManager = new
 DeviceConfigManager() {
    @Override
    public DeviceIdentity getDeviceIdentity(String originalIdentity) {
        //Suppose you dynamically get deviceInfo in other ways
        return devicesMap.get(originalIdentity);
    }

    @Override
    public String getOriginalIdentity(String productKey, String
deviceName) {
        //you can ignore this
        return null;
    }
};
```

Dynamically register a bridge device

When you need to deploy a bridge application on a large number of servers, it is cumbersome to specify different bridge devices for different bridge servers. You can configure the bridge information file *`application.conf`* to dynamically register bridge devices with IoT Platform. You must provide the `productKey` and `popClientProfile` parameters in the configuration file. The generic protocol SDK will call the IoT Platform API and use the bridge servers' MAC codes as the device names to register bridge devices.

Note:

- **To dynamically register bridge devices, you only need to modify the bridge configuration file. The call code is the same as *Use the basic features*.**

- **If the bridge information is already specified in the bridge configuration file, no device is created. The generic protocol SDK calls the IoT Platform API and uses the bridge server's MAC code as the device name to register a bridge device only if the following conditions are met: The `deviceName` and `deviceSecret` parameters are left empty in the configuration file; all parameters in `popClientprofile` are specified. If a device is already registered using the current MAC code, the device is directly used as the bridge device.**
- **If a bridge is configured by using this method, we recommend that you do not perform debugging on a local client by using the configurations for the production environment. Each time the program is debugged on a local client, the generic protocol SDK uses the MAC code of the client to register a bridge device, and associates all devices in the device configuration file *`devices.conf`* with the bridge. We recommend that you use dedicated devices for testing to perform debugging to avoid interference with the production environment.**

Table 7-2: Configuration parameters

Parameter	Required	Description
productKey	Yes	The ProductKey of the product to which the bridge device belongs.

Parameter	Required	Description
http2Endpoint	Yes	The endpoint of the HTTP/2 gateway service. The bridge device and IoT Platform establish a persistent connection over the HTTP/2 protocol. The endpoint is in the format of `${productKey}.iot-as-http2 .${RegionId}.aliyuncs.com:443`. Replace *${productKey}* with the ProductKey of the product to which your bridge device belongs. Replace *${RegionId}* with the ID of the region where your service is located. For more information about regions, see *Regions and zones*. For example, if the ProductKey of the bridge device is alabcabc123, the region is China (Shanghai), then the HTTP/2 gateway service endpoint is `alabcabc123.iot-as-http2.cn-shanghai.aliyuncs.com:443`.
authEndpoint	Yes	The service URL for device authentication. The device authentication service URL is in the format of `https://iot-auth .${RegionId}.aliyuncs.com/auth/bridge`. Replace *${RegionId}* with the ID of the region where your service is located. For more information about regions, see *Regions and zones*. For example, if the region is China (Shanghai), then the device authentication service address is `https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge`.

Parameter	Required	Description
popClientProfile	Yes	After this parameter is configured, the generic protocol SDK calls the IoT Platform API to automatically register bridge devices. For more information, see the following table: Parameters in popClientProfile.

Table 7-3: Parameters in popClientProfile

Parameter	Required	Description
accessKey	Yes	The AccessKey ID of your Alibaba Cloud account. Log on to the Alibaba Cloud console and click your account avatar to go to the Account Management page. You can create or view the AccessKey information.
accessSecret	Yes	The AccessKey Secret of your Alibaba Cloud account.
name	Yes	The IoT Platform service region to which the bridge device connects. This parameter indicates the region to which the product identified by `productKey` belongs. For more information about regions, see *Regions and zones*.
region	Yes	The ID of the IoT Platform service region to which the bridge device connects. This parameter indicates the region to which the product identified by `productKey` belongs. This parameter is expressed in the same way as the `name` parameter.
product	Yes	The product name. Set the value to `Iot`.

Parameter	Required	Description
endpoint	Yes	The endpoint of the API. The endpoint is in the format of `iot. ${RegionId}.aliyuncs.com`. Replace *${RegionId}* with the ID of the region where your service is located. For more information about regions, see *Regions and zones*. For example, If the region is China (Shanghai), the endpoint is `iot.cn-shanghai.aliyuncs.com`.

Sample code to dynamically register bridge devices:

```
# Server endpoint
http2Endpoint = "https://${YourProductKey}.iot-as-http2.cn-shanghai.
aliyuncs.com:443"
authEndpoint = "https://iot-auth.cn-shanghai.aliyuncs.com/auth/bridge"

# Gateway device info
# You can also specify productKey only, and dynamic register
deviceName & deviceSecret in runtime
productKey = ${YourProductKey}

# If you dynamic register gateway device using your mac address, you
have to specify 'popClientProfile'
# otherwise you can ignore it
popClientProfile = {
    accessKey = ${YourAliyunAccessKey}
    accessSecret = ${YourAliyunAccessSecret}
    name = cn-shanghai
    region = cn-shanghai
    product = Iot
    endpoint = iot.cn-shanghai.aliyuncs.com
}
```

Call interfaces to report TSL data

To facilitate use and reduce your encapsulation operations, the generic protocol SDK encapsulates data reporting interfaces. They are `reportProperty`, `fireEvent`, and `updateDeviceTag`. The device can use these interfaces to report properties, report events, and update device tags.

Prerequisites and usage guidelines:

- **Before you call `reportProperty` and `fireEvent` to report properties and events, log on to the *IoT Platform console* and go to the Product Details page of the corresponding product. Then, click the Define Feature tab and define properties and events. For more information, see *#unique_42*.**

- **If the tag that is specified in `updateDeviceTag` already exists, the tag value is updated. If the tag does not exist, the tag is automatically created. To check the call result, you can log on to the IoT Platform console and go to the Device Details page of the corresponding device.**

Sample code:

```
TslUplinkHandler tslUplinkHandler = new TslUplinkHandler();
//report property
//Property 'testProp' is defined in IoT Platform Web Console
String requestId = String.valueOf(random.nextInt(1000));
tslUplinkHandler.reportProperty(requestId, originalIdentity, "testProp
", random.nextInt(100));

//fire event
//Event 'testEvent' is defined in IoT Platform Web Console
requestId = String.valueOf(random.nextInt(1000));
HashMap<String, Object> params = new HashMap<String, Object>();
params.put("testEventParam", 123);
tslUplinkHandler.fireEvent(originalIdentity, "testEvent", ThingEvent
Types.INFO, params);

//update device tag
//'testDeviceTag' is a tag key defined in IoT Platform Web Console
requestId = String.valueOf(random.nextInt(1000));
tslUplinkHandler.updateDeviceTag(requestId, originalIdentity, "
testDeviceTag", String.valueOf(random.nextInt(1000)));
```

The parameters in this example are described as follows:

Parameter	Description
requestId	The request ID.
originalIdentity	The original identifier of the device.
testProp	The identifier of the property. For this example, make sure that you have defined a property with the identifier as testProp in the IoT Platform console. This sample code indicates to report the value of property testProp.
random.nextInt(100)	The property value to be reported. The value range of the property value is also defined in the IoT Platform console. In this example, use `random.nextInt(100)` to indicate a random number less than 100.
testEvent	The identifier of the event. For this example, make sure that you have defined an event with the identifier as testEvent in the IoT Platform console. This sample code indicates to report event testEvent.

Parameter	Description
ThingEventTypes.INFO	The event type. `ThingEventTypes` specifies the event type. A value of `INFO` indicates that the event type is Info. For this example, make sure that you have selected Info as the event type when you defined event testEvent in the IoT Platform console.
params	The output parameters of the event. The identifier, data type, and value range of output parameters are also defined in the IoT Platform console. In this example, the identifier of the output parameter is testEventParam, and the value is 123.
testDeviceTag	The key of the tag. The data type is String. In this example, the key is testDeviceTag. Set the key of the tag as instructed based on your requirements. For more information, see *#unique_43/unique_43_Connect_42_section_igy_bvb_wdb*.
String.valueOf(random.nextInt(1000))	The value of the tag. The data type is String. In this example, `String.valueOf(random.nextInt(1000))` indicates a random number less than 1000. Set the value of the tag as instructed based on your requirements. For more information, see *#unique_43/unique_43_Connect_42_section_igy_bvb_wdb*.