Alibaba Cloud
物联网平台

Manage Device

Issue: 20191210

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 Device lifecycle management

1.1 Add devices

The device management function of IoT Platform allows you to view and manage the lifecycle of a device. You first add a device in IoT Platform. You can add a device in the IoT Platform console or by calling the API operation.

Use the console to add a device

1. **Log on to the *IoT Platform console*.**
2. **In the left-side navigation pane, choose Devices > Products.**
3. **On the Products page that appears, click Create Product. In the displayed dialog box, enter the required product information to create a product.**

   For more information, see *#unique_5*.
4. **In the left-side navigation pane, click Devices.**
5. **On the Devices page, add a device.**
   - **You can click Batch Add to add multiple devices at a time.**
   - **You can click Add Device to add a single device.**

   For more information, see *#unique_6* and *#unique_7*.

1.2 Connect devices and disconnect devices

When a device is connected to IoT Platform, the device is in the Online state. When a device is disconnected from IoT Platform, the device is in the Offline state.

Connect a device to IoT Platform

Develop a device and connect the device to IoT Platform.

> **Note:**
> **The following section describes how to directly connect a device to IoT Platform. For more information about how to connect sub-devices to IoT Platform, see *Connect sub-devices to IoT Platform*.**

1. **Develop the device.**

   IoT Platform provides device SDKs in multiple programming languages. These SDKs encapsulate protocols for communication between devices and IoT Platform. For more information about using a device SDK, see *#unique_10*.

   When developing a device, configure the identity information of the device. The identity information is used to authenticate the device when it connects to IoT Platform.

   IoT Platform supports the following methods for the authentication of devices that are directly connected:

   - ***#unique_11*: This method requires that each device has a unique device certificate installed in advance. The device certificate includes the ProductKey, DeviceName, and DeviceSecret of the device.**
   - ***#unique_12*: This method allows you to install the same firmware (a product certificate including ProductKey and ProductSecret) on all devices of a product. Then, you can use the product certificate to perform device authentication. To use this method, you need to enable dynamic device registration on the Product Details page of the product. When a device initiates a connection request, IoT Platform verifies the product certificate. After the authentication is passed, IoT Platform assigns the corresponding DeviceSecret to the device.**

2. **Install the device SDK to the device.**
3. **Power on the device, connect the device to the network, and then the device connects to IoT Platform.**

Disconnect a device from IoT Platform

After a device is disconnected from IoT Platform, the status of the device in IoT Platform is Offline. Two types of device disconnection are available.

- **Active disconnection: a device disconnects from IoT Platform.**
- **Forcible disconnection: IoT Platform disconnects from the device. For example , if another device uses the same device certificate to access IoT Platform, the current device is forced to disconnect from IoT Platform. A forcible disconnect ion also occurs when you have deleted or disabled the device in IoT Platform.**

1.3 Disable and Enable devices

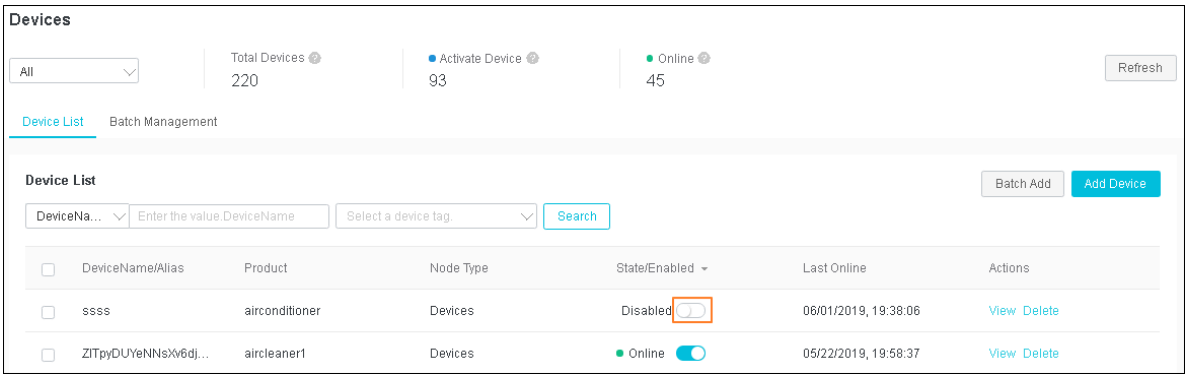
By disabling a device, the device cannot be connected to IoT Platform. By enabling a disabled device, the device can be connected to IoT Platform again. This topic describes how to disable and enable a device.

Disable a device

> **Note:**
> **After a device is disabled, IoT Platform retains the information associated to the device. However, the device cannot be connected to IoT Platform, and you cannot perform operations related to the device.**

To disable a device in the console, follow these steps:

1. **Log on to the *IoT Platform console*.**
2. **In the left-side navigation pane, choose Devices > Devices.**
3. **In the Device List section on the page that appears, find the device that you want to disable. Turn off the Enabled switch.**



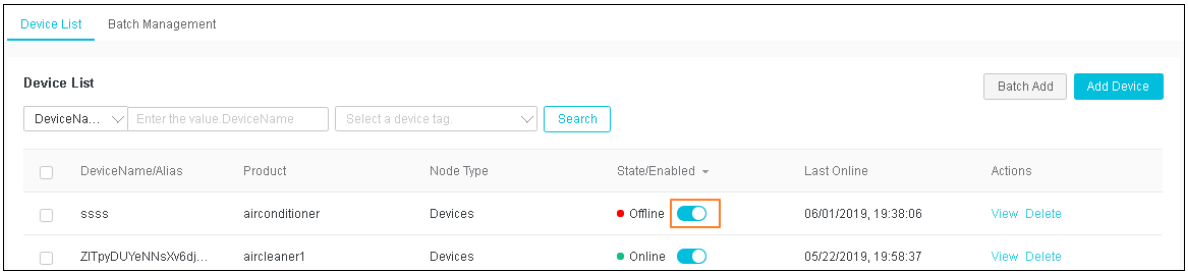
Enable a device

After you disable a device, you can enable it again.

To enable a device in the console, Follow these steps:

1. **In the *IoT Platform console*, open the Devices page. In the Device List section, find the device that you want to enable.**

2. Turn on the Enabled switch.

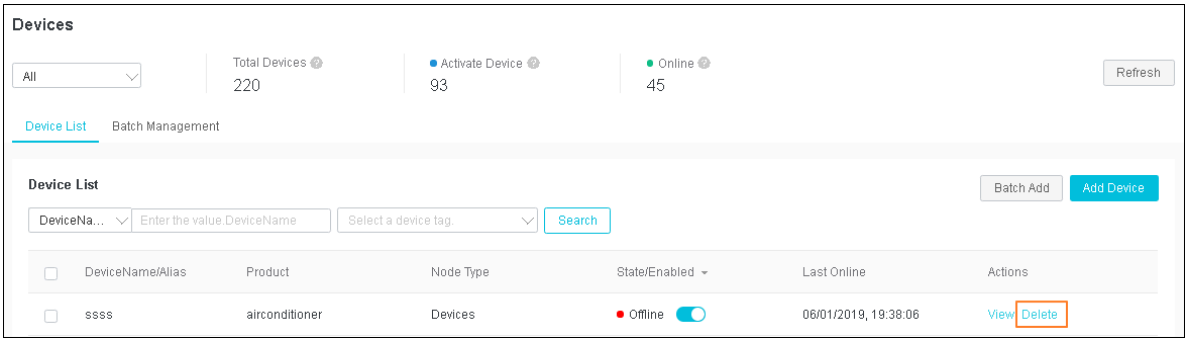


1.4 Delete devices

You can delete devices from IoT Platform. After a device is deleted from IoT Platform, the device ID becomes invalid and all information associated with the device is deleted. You cannot perform operations related to the device by using IoT Platform.

Procedure

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, choose Devices > Devices.
3. In the Device List section on the page that appears, find the device that you want to delete.
4. Click Delete in the Actions column and confirm the deletion.



Result

After the device is deleted, the device certificate becomes invalid and cannot be restored.

2 TSL

2.1 Overview

Thing Specification Language (TSL) is a data model that digitizes a physical entity and constructs the entity data model in IoT Platform. In IoT Platform, a TSL model refers to a set of product features. After you have defined features for a product, the system automatically generates a TSL model of the product. A TSL model describes what a product is, what the product can do, and what services the product can provide.

A TSL model is a file in JSON format. TSL files are the digitized expressions of physical entities, such as sensors, vehicle-mounted devices, buildings and factories . A TSL file describes an entity in three dimensions: property (what the entity is), service (what the entity can do), and event (what event information the entity reports). Defining these three dimensions is to define the product features.

Therefore, the feature types of a product are Properties, Services and Events. You can define these three types of features in the console.

Feature type	Description
Property	Describes a running status of a device, such as the current temperature read by the environmental monitoring equipment. You can use GET and SET methods to send requests to get and set device properties.
Service	Indicates a feature or method of a device that can be used by a user. You can set input parameters and output parameters for a service. Compared with properties, services can implement more complex business logic, for example, a specific task.
Event	Indicates the notifications of a type of event occurred when a device is running. Events typically indicate notifications that require actions or attention, and they may contain multiple output parameters. For example, events can be notifications about the completion of tasks, system failures, or temperature alerts. You can subscribe to events or push events to a message receiving target.

The TSL format

The format of Thing Specification Language (TSL) is JSON. This article introduces the JSON fields of TSL.

In the Define Feature tab of your target product, click View TSL.

The following section details each JSON field.

```
{
    "schema":"TSL schema of a thing",
    "link":"System-level URI in the cloud, used to invoke services and
 subscribe to events",
    "profile":{
        "productKey":" Product ID",
    },
    "properties":[
        {
            "identifier":"Identifies a property. It must be unique
under a product",
            "name":"Property name",
            "accessMode":"Read/write type of properties, including
Read-Only and Read/Write",
            "required":"Determines whether a property that is required
 in the standard category is also required for a standard feature",
            "dataType":{
                "type":"Data type: int (original), float (original),
double (original), text (original), date (UTC string in milliseconds
), bool (integer, 0 or 1), enum (integer), struct (supports int, float
, double, text, date, enum, and bool), array (supports int, double,
float, struct, and text)",
                "specs":{
                    "min":"Minimum value, available only for the int,
float, and double data types",
                    "max":"Maximum value, available only for the int,
float, and double data types",
                    "unit":"Property unit",
                    "unitName":"Unit name",
                    "size":"Array size, up to 128 elements, available
only for the array data type",
                    "item":{
                        "type":"Type of an array element"
                    }
                }
            }
        }
    ],
    "events":[
        {
            "identifier":"Identifies an event that is unique under a
product, where "post" are property events reported by default",
            "name":"Event name",
            "desc":"Event description",
            "type":"Event types, including info, alert, and error",
            "required":"Whether the event is required for a standard
feature",
            "outputData":[
                {
                    "identifier":"Uniquely identifies a parameter",
                    "name":"Parameter name",
                    "dataType":{
```

```
                        "type":"Data type: int (original), float
 (original), double (original), text (original), date (UTC string
in milliseconds), bool (integer, 0 or 1), enum (integer), struct (
supports int, float, double, text, date, enum, and bool), array (
supports int, double, float, struct, and text)",
                        "specs":{
                            "min":"Minimum value, available only for
the int, float, and double data types",
                            "max":"Maximum value, available only for
the int, float, and double data types",
                            "unit":"Property unit",
                            "unitName":"Unit name",
                            "size":"Array size, up to 128 elements,
available only for the array data type",
                            "item":{
                                "type":"Type of an array element"
                            }
                        }
                    }
                }
            ],
            "method":"Name of the method to invoke the event,
generated according to the identifier"
        }
    ],
    "services":[
        {
            "identifier":"Identifies a service that is unique under a
product (set and get are default services generated according to the
read/write type of the property)",
            "name":"Service name",
            "desc":"Service description",
            "required":"Whether the service is required for a standard
 feature",
            "inputData":[
                {
                    "identifier":"Uniquely identifies an input
parameter",
                    "name":"Name of an input parameter",
                    "dataType":{
                        "type":"Data type: int (original), float
 (original), double (original), text (original), date (UTC string
in milliseconds), bool (integer, 0 or 1), enum (integer), struct (
supports int, float, double, text, date, enum, and bool), array (
supports int, double, float, struct, and text)",
                        "specs":{
                            "min":"Minimum value, available only for
the int, float, and double data types",
                            "max":"Maximum value, available only for
the int, float, and double data types",
                            "unit":"Property unit",
                            "unitName":"Unit name",
                            "size":"Array size, up to 128 elements,
available only for the array data type",
                            "item":{
                                "type":"Type of an array element"
                            }
                        }
                    }
                }
            ],
            "outputData":[
                {
```

```
                    "identifier":"Uniquely identifies an output
parameter",
                    "name":"Name of an output parameter",
                    "dataType":{
                        "type":"Data type: int (original), float
 (original), double (original), text (original), date (UTC string
in milliseconds), bool (integer, 0 or 1), enum (integer), struct (
supports int, float, double, text, date, enum, and bool), array (
supports int, double, float, struct, and text)",
                        "specs":{
                            "min":"Minimum value, available only for
the int, float, and double data types",
                            "max":"Maximum value, available only for
the int, float, and double data types",
                            "unit":"Property unit",
                            "unitName":"Unit name",
                            "size":"Array size, up to 128 elements,
available only for the array data type",
                            "item":{
                                "type":"Type of an array element,
available only for the array data type"
                            }
                        }
                    }
                }
            ],
            "method":"Name of the method to invoke the service, which
is generated according to the identifier"
        }
    ]
}
```

If the product is connected to a gateway as a sub-device and the connection protocol is Modbus or OPC UA, you can view the TSL extension configuration.

```
{
"profile": {
"productKey": "Product ID",
  },
"properties": [
    {
"identifier": "Identifies a property. It must be unique under a
product",
"operateType": "(coilStatus/inputStatus/holdingRegister/inputRegister
)",
"registerAddress": "Register address",
"originalDataType": {
"type": "Data type: int16, uint16, int32, uint32, int64, uint64, float
, double, string, customized data(returns hex data according to big-
endian)",
"specs": {
"registerCount": "The number of registers, available only for string
and customized data",
"swap16": "swap the first 8 bits and the last 8 bits of the 16 bits of
 the register data(for example, byte1byte2 -> byte2byte10). Available
for all the other data types except string and customized data",
"reverseRegister": "Ex: Swap the bits of the original 32 bits data (
for example, byte1byte2byte3byte4 ->byte3byte4byte1byte2”. Available
for all the other data types except string and customized data"
        }
      },
"scaling": "Scaling factor",
```

```
"pollingTime": "Polling interval. The unit is ms",
"trigger": "The trigger of data report. Currently, two types of
triggering methods are supported: 1: report at the specified time; 2:
report when changes occurred"
    }
  ]
}
```

Use TSL

1. **In the IoT Platform console,** *Define features* **or** *Import Thing Specification Language (TSL)*.
2. **Develop the SDK.**

   See the documentations of *Link Kit SDK* **for help information.**
3. **Connect the SDK to IoT Platform. Then, devices can report properties and events to IoT Platform, and in IoT Platform, you can set properties and call device services.**

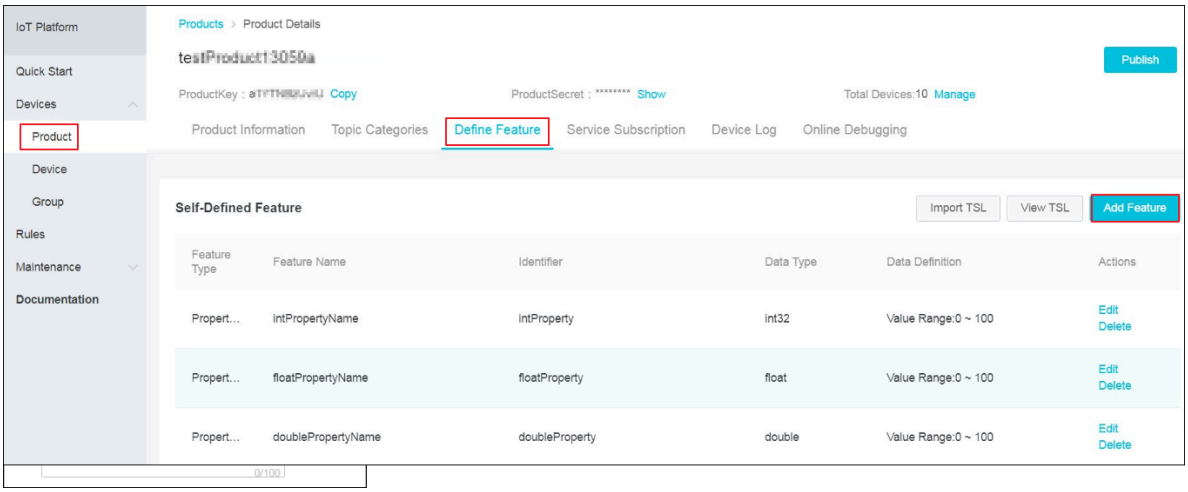
2.2 Define features

Defining features for products is to define Thing Specification Language (TSL), including defining properties, services, and events. This article describes how to define features in the IoT Platform console.

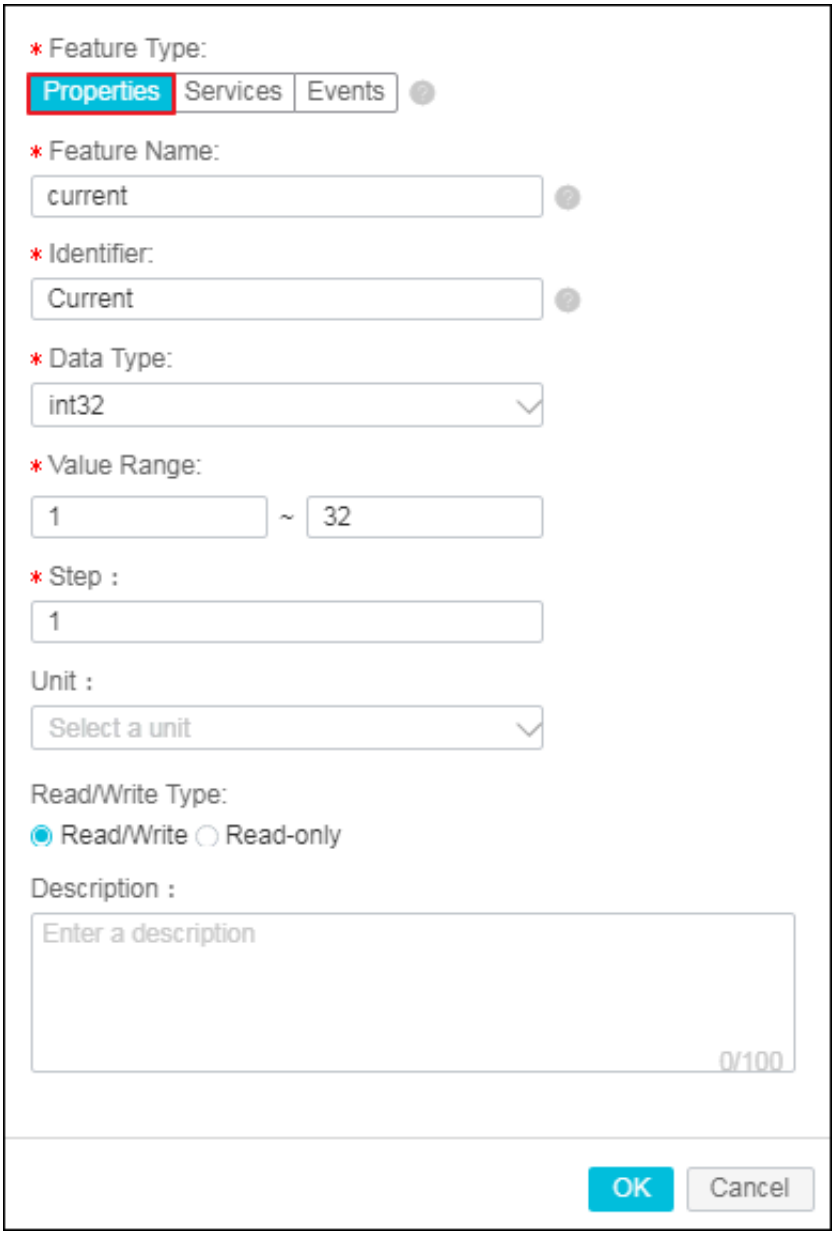
Procedure

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 define features and click View.**
4. **Click Define Feature.**

5. **Add self-defined features. Click the Add Feature button corresponding to Self-defined Feature to add custom features for the product. You can define properties, services and events for the product.**



- **Define a property. In the Add Self-defined Feature dialog box, select Properties as the feature type. Enter information for the property and then clickOK.**



The parameters of properties are listed in the following table.

Parameter	Description
The function name	Property name, for example, Power Consumption. Each feature name must be unique in the product. A feature name must start with a Chinese character, an English letter, or a digit, can contain Chinese characters, English letters, digits, dashes(-) and underscores (_), and cannot exceed 30 characters in length.

Parameter	Description
Identifier	Identifies a property. It must be unique in the product. It is the parameter `identifier` in Alink JSON TSL, and is used as the key when a device is reporting data of this property. Specifically, IoT Platform uses this parameter to verify and determine whether or not to receive the data. An identifier can contain English letters, digits, and underscores (_), and cannot exceed 50 characters in length. For example, PowerConsumption.  Note: An identifier cannot be any one of the following words: set, get, post, time, and value, because they are system parameter names.

Parameter	Description
Data Type	- `int32`: 32-bit integer. If you select int32, you are required to define the value range, step, and unit. - `float`: Float. If you select float, you are required to define the value range, step, and unit. - `double`: Double float. If you select double, you are required to define the value range, step, and unit. - `enum`: Enumeration. You must specify enumeration items with values and descriptions. For example,1 indicates heating mode and 2 indicates cooling mode. - `bool`: Boolean. You must specify the Boolean values. Values include 0 and 1. For example, you can use 0 to indicate disabled and 1 to indicate enabled. - `text`: Text string. You must specify the data length. The maximum value is 2048 bytes. - `date`: Timestamp. A UTC timestamp in string type, in milliseconds. - `struct`: A JSON structure. Define a JSON structure, and add new JSON parameters. For example, you can define that the color of a lamp is a structure composed of three parameters: red, green, and blue. Structure nesting is not supported. - `array`: Array. You must select a data type for the elements in the array from int32, float, double, text and struct. Make sure that the data type of elements in an array is the same and that the length of the array does not exceed 128 elements. **Note:** When the gateway connection protocol is Modbus, you do not set this parameter.
Step	The smallest granularity of changes of properties, events, and input and output parameter values of services. If the data type is int32, float, or double, step is required.
Unit	You can select None or a unit suitable.

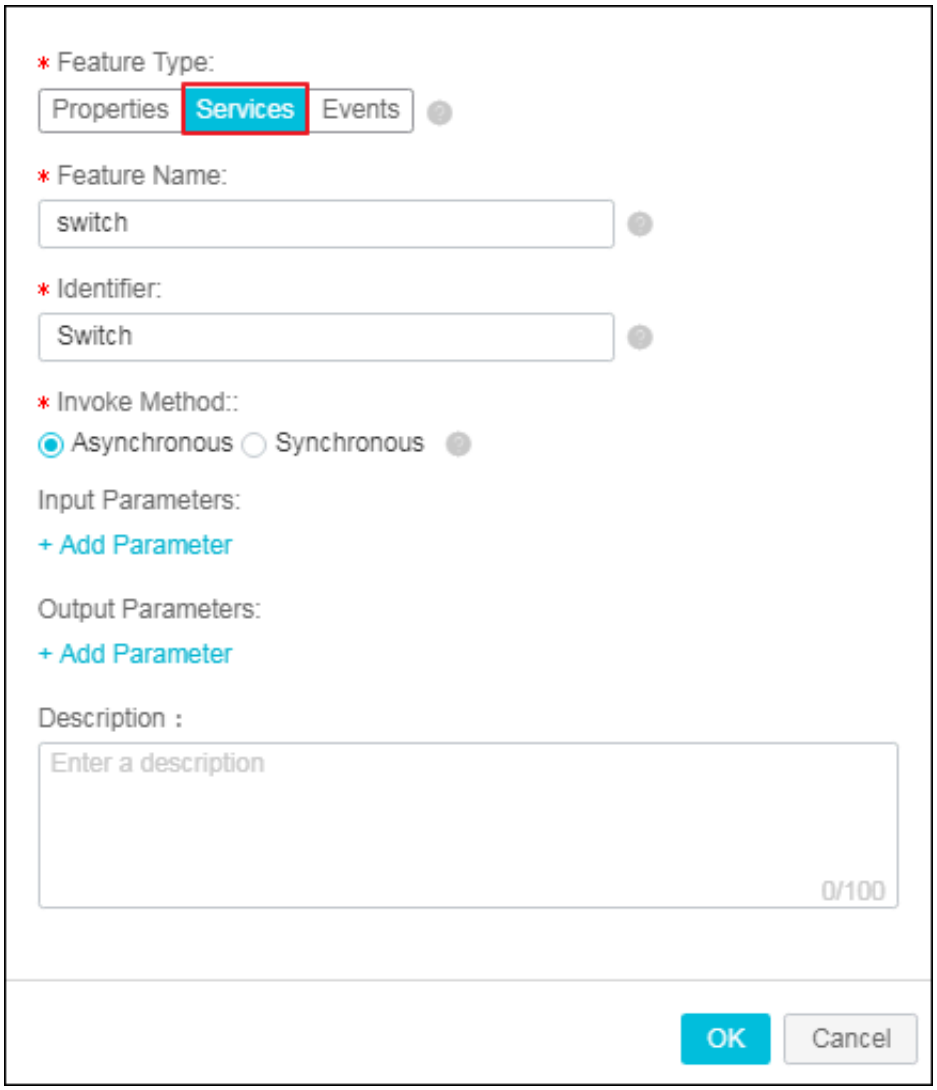
Parameter	Description
Read/Write Type	- `Read/Write`: GET and SET methods are supported for Read/Write requests. - `Read-only`: Only GET is supported for Read-only requests.  **Note:** When the gateway connection protocol is Modbus, you do not set this parameter.
Description	Enter a description or remarks about the property. You can enter up to 100 characters.

Parameter	Description
Extended Information	When the gateway connection protocol is Custom, Modbus, or OPC UA, you can configure extended information. - When the gateway connection protocol is Custom, add custom configurations. The custom configurations must be written in JSON format, and can contain up to 1,024 characters. - When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the property. - When the gateway connection protocol is Modbus, configure the following parameters. ■ Operation Type: ■ Coil Status (read-only, 01) ■ Coil Status (read and write, 01-read, 05-write) ■ Coil Status (read and write, 01-read, 0F-write) ■ Discrete Input (read-only, 02) ■ Holding Registers (read-only, 03) ■ Holding Registers (read and write, 03-read, 06-write) ■ Holding Registers (read and write, 03-read, 10-write) ■ Input Registers (read-only, 04) ■ Register Address: Enter a hexadecimal address beginning with 0x. The range is 0x0 - 0xFFFF. For example, 0xFE. ■ Original Data Type: Multiple data types are supported, including int16, uint16, int32, uint32, int64, uint64, float , double, string, bool, and customized data (raw data). ■ Switch High Byte and Low Byte in Register: Swap the first 8 bits and the last 8 bits of the 16-bit data in the register. Options: ■ `true` ■ `false` ■ Switch Register Bits Sequence: Swap the bits of the original 32-bit data. Options: ■ `true` ■ `false` ■ Zoom Factor: The zoom factor is set to 1 by default. It can be set to negative numbers, but cannot be set to 0. ■ Collection Interval: The time interval of data collection. It is in milliseconds and the value cannot be lower than 10. ■ Data Report: The trigger of data report. It can be either `At Specific Time` or `Report Changes`.

- **Define a service. In the Add Self-defined Feature dialog box, select Services as the feature type. Enter information for the service and then click OK.**

   **Note:**

When the gateway connection protocol is Modbus, you cannot define any service for the product.



The parameters of services are as follows.

Parameter	Description
The function name	Service name. A feature name must start with an English letter, Chinese character, or a number. It can contain English letters, Chinese characters, digits, dashes (-), and underscores (_), and cannot exceed 30 characters in length. If you have selected a category with feature template when you were creating the product, the system displays the standard services from the standard feature library for you to choose. Note: When the gateway connection protocol is Modbus, you cannot define custom services for the product.
Identifier	Identifies a service. It must be unique within the product. The parameter `identifier` in Alink JSON TSL. It is used as the key when this service is called. An identifier can contain English letters, digits, and underscores (_), and cannot exceed 30 characters in length. Note: Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value.
Invoke Method	- `Asynchronous`: For an asynchronous call, IoT Platform returns the result directly after the request is sent, and does not wait for a response from the device. - `Synchronous`: For a synchronous call, IoT Platform waits for a response from the device. If no response is received, the call times out.

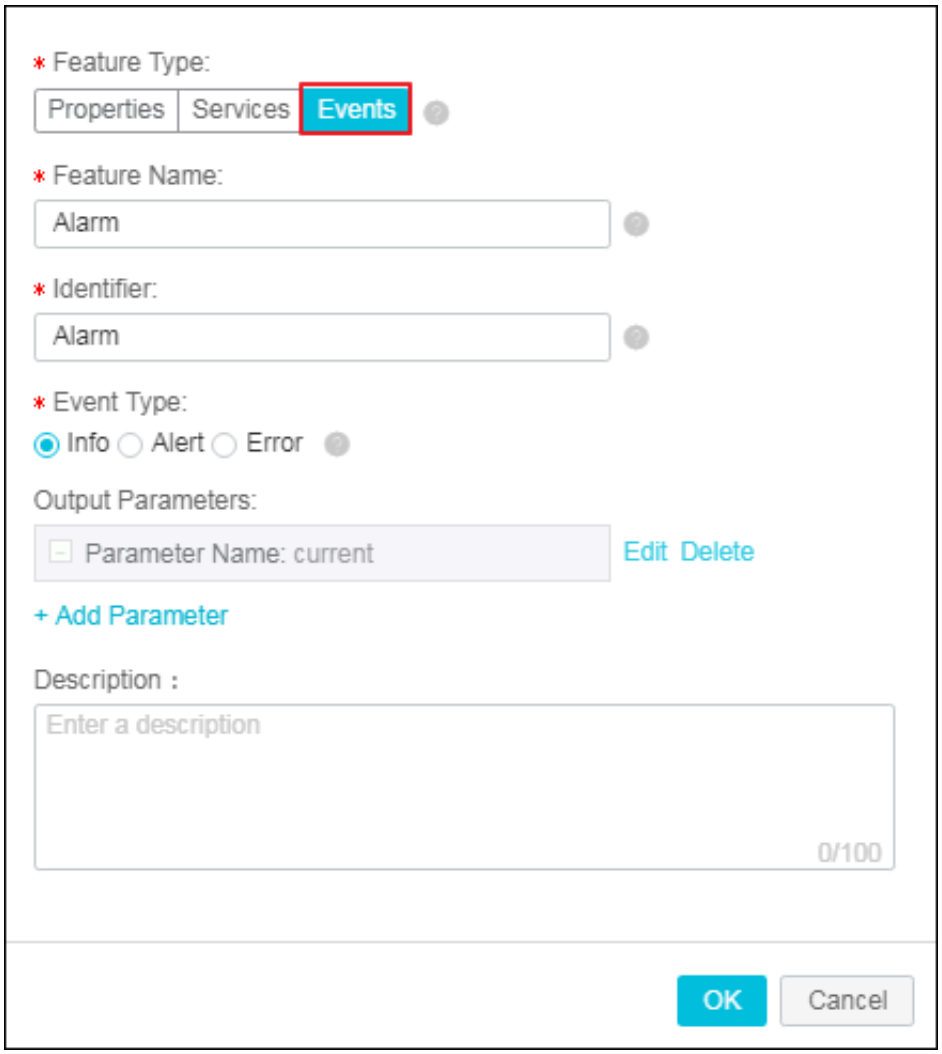
Parameter	Description
Input Parameters	(Optional) Set input parameters for the service. Click Add Parameter, and add an input parameter in the dialog box that appears. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. Note: - Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value. - You can either use a property as an input parameter or define an input parameter. For example, you can specify the properties `Sprinkling Interval` and `Sprinkling Amount` as the input parameters of the `Automatic Sprinkler` service feature. Then, when Automatic Sprinkler is called, the sprinkler automatically starts irrigation according to the sprinkling interval and amount. - You can add up to 20 input parameters for a service.
Output Parameters	(Optional) Set output parameters for the service. Click Add Parameter, and add an output parameter in the dialog box that appears. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. Note: - Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value. - You can either use a property as an output parameter or define an output parameter. For example, you can specify the property `SoilHumidity` as an output parameter. Then, when the service Automatic Sprinkler is called, IoT Platform returns the data about soil humidity. - You can add up to 20 output parameters for a service.

Parameter	Description
Extended Information	- When the gateway connection protocol is Custom, add custom configurations. The custom configurations must be written in JSON format, and can contain up to 1,024 characters. - When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the service.
Description	Enter a description or remarks about the service. You can enter up to 100 characters.

- Define an event. In the Add Self-defined Feature dialog box, select Events as the feature type. Enter information for the parameter and then click OK.

Note:

When the gateway connection protocol is Modbus, you cannot define any event for the product.



The parameters of events are as follows.

Parameter	Description
The function name	**Event name.** **A feature name must start with a Chinese character, an English letter, or a digit, can contain Chinese characters, English letters, digits, dashes(-) and underscores (_), and cannot exceed 30 characters in length.** **Note:** **When the gateway connection protocol is Modbus, you cannot define events.**

Parameter	Description
Identifier	Identifies an event. It must be unique in the product. It is the parameter `identifier` in Alink JSON TSL, and is used as the key when a device is reporting data of this event, for example, ErrorCode. **Note:** Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value.
Event Type	- `Info`: Indicates general notifications reported by devices, such as the completion of a specific task. - `Alert`: Indicates alerts that are reported by devices when unexpected or abnormal events occur. It has a high priority. You can perform logic processing or analytics depending on the event type. - `Error`: Indicates errors that are reported by devices when unexpected or abnormal events occur. It has a high priority. You can perform logic processing or analytics depending on the event type.
Output Parameters	The output parameters of an event. Click Add Parameter, and add an output parameter in the dialog box that appears. You can either use a property as an output parameter or define an output parameter. For example, you can specify the property `Voltage` as an output parameter. Then, devices report errors with the current voltage value for further fault diagnosis. When the gateway connection protocol is OPC UA, you must set the parameter index that is used to mark the order of the parameters. **Note:** - Identifiers of input parameters cannot be any one of the following words: set, get, post, time, and value. - You can add up to 50 output parameters for an event.

Parameter	Description
Extended Information	- When the gateway connection protocol is Custom, add custom configurations. The custom configurations must be written in JSON format, and can contain up to 1,024 characters. - When the gateway connection protocol is OPC UA, set a node name. Each node name must be unique under the event.
Description	Enter a description or remarks about the event. You can enter up to 100 characters.

2.3 Import Thing Specification Language (TSL)

This article introduces how to import an existing TSL for a product.

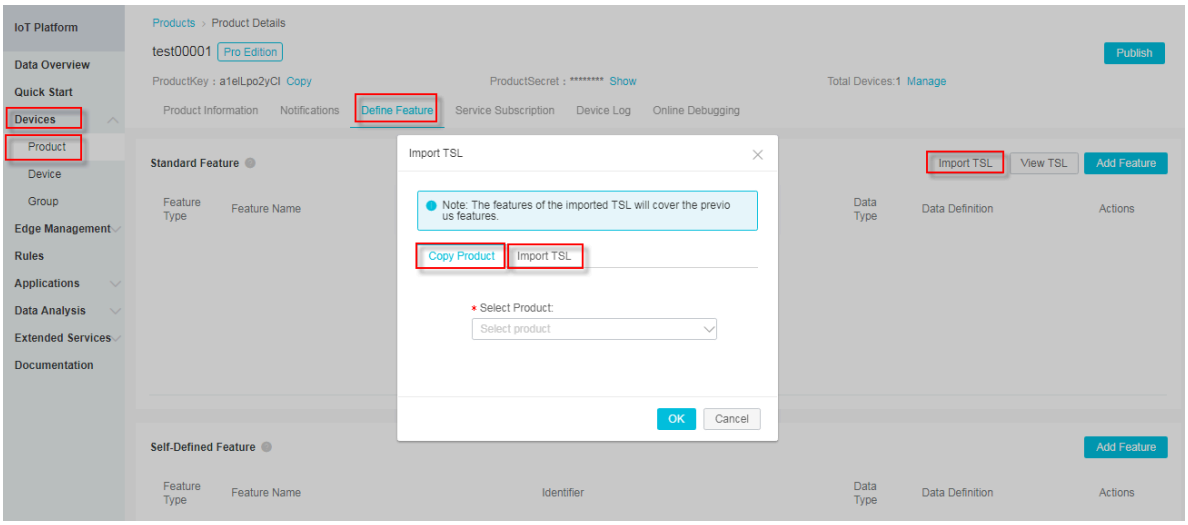
Procedure

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 import TSL and click View.**
4. **Click Define Feature > Import TSL.**

Note:

- **The previously defined features of the product will be overwritten, once you have imported a new TSL for the product. Therefore, this function must be used with caution.**

- You cannot import a TSL for a product whose gateway connection protocol is defined as Modbus.



You can import TSL in two ways:

- Copy Product: Copy the TSL of another product. Select an existing product and click OK to import the TSL of the selected product to this product.

  If you want to modify some features, click Edit corresponding to the features on the Define Feature tab page.
- Import TSL: Paste your self-defined TSL script into the edit box and click OK.

  Note:

  The size of the imported file cannot exceed 256 KB.

  The value of ProductKey in the TSL script must be the ProductKey of the target product.

3 Data parsing

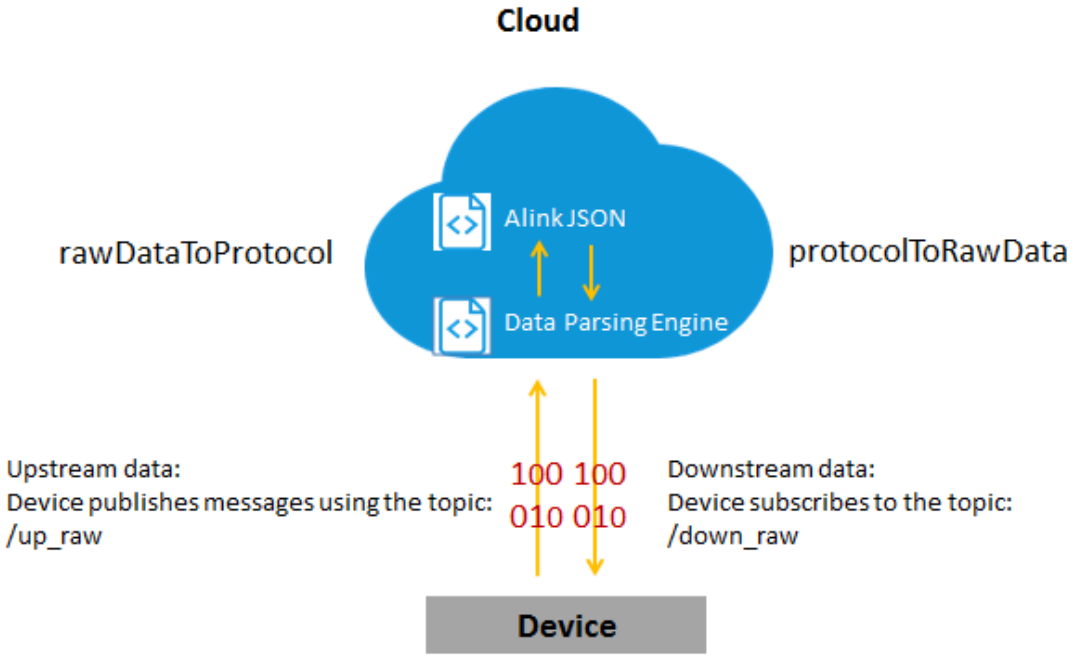
3.1 Data parsing

Devices with low configurations and limited resources or devices that have high requirements for network traffic can send raw data to IoT Platform. This prevents the devices from directly sending data to IoT Platform in Alink JSON format. You must write a data parsing script in the IoT Platform console to parse upstream and downstream data to be in standard Alink JSON format and the custom data format, respectively.

About data parsing

When receiving raw data from a device, IoT Platform runs the parsing script to convert the raw data to the Alink JSON data for business processing. When sending data to the device, IoT Platform also runs the parsing script to convert the Alink JSON data to the device custom formatted data.

Data parsing process:



For more information about sending data upstream and downstream, see "Devices report properties or events" and "Call device services or set device properties" in *Communications over Alink protocol*.

Script format

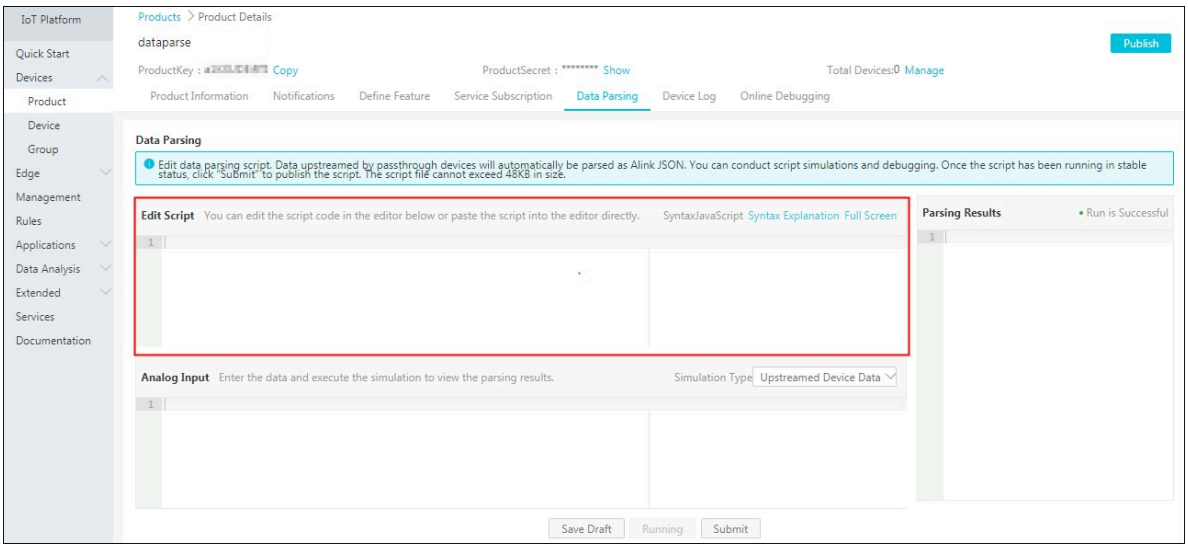
```
/**
 * Convert data in Alink JSON format to data format that can be
identified by the device. This feature is called when IoT Platform
sends data to a device.
 *  Input: jsonObj   Object  Required
 *  Output: rawData byte[]   Array  Required
 *
 */
function protocolToRawData(jsonObj) {
    return rawdata;
}

/**
 * Convert the custom formatted data to Alink JSON data. This function
 is called when a device reports data to IoT Platform.
 * Input: rawData byte[]   Array   Required
 * Output: jsonObj   Object   Required
 */
function rawDataToProtocol(rawData) {
    return jsonObj;
}
```

Edit and verify scripts

Only JavaScript is supported to edit scripts. IoT Platform provides an online script editor that allows you to edit and submit scripts, and simulate data parsing for testing.

1. **Log on to the IoT Platform console.**
2. **From the left-side navigation pane, choose Devices > Product.**
3. **Click Create Product to create a product and set the data type to Do not parse/ Custom. For more information, see *#unique_5*.**

4. On the Product Details page, click the Data Parsing tab. Edit your data parsing script in the editor. Only JavaScript is supported. For more information, see *Example: Edit a script*.



When you edit the script, you can perform the following operations:

- Click Full Screen to view or edit the script in full screen. Click Exit Full Screen to exit the full screen.
- ClickSave Draft at the bottom of the page to save the content that you have edited. The next time you access the Data Parsing page, you will be notified that you have a draft. You can then choose to restore edit or delete draft.
  - A saved draft script will not be published to the running platform and will not affect a published script.
  - A new draft will overwrite any previously saved draft.

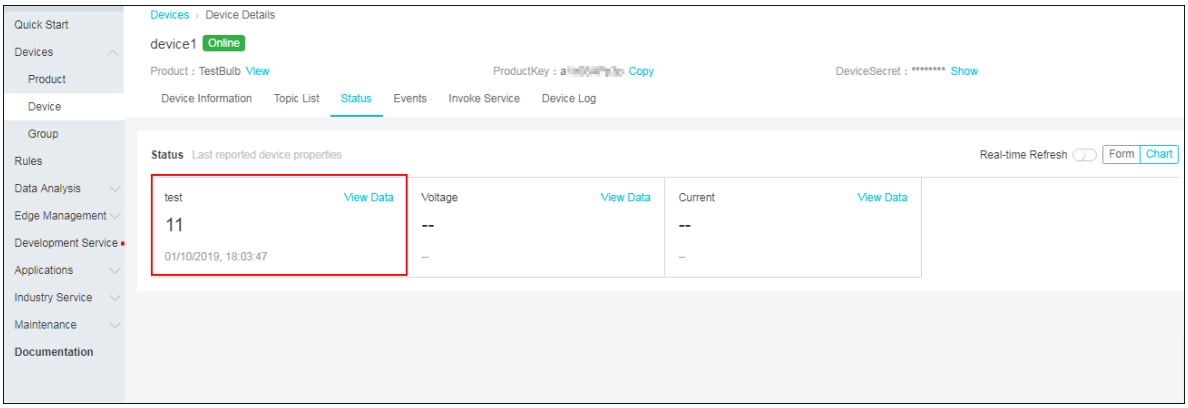
5. After you finish editing the script, you can enter analog data in the Analog Input box. Click Run to test whether the script can be used to parse data correctly. For more information about analog data and parsing results, see *Verify a data parsing script*.
6. If you confirm that the script is correct and can parse data correctly, click Submit to submit the script to the running platform. When data is exchanged between IoT Platform and the device, the system will automatically call the corresponding function in the script to convert data.

7. **Perform a test by sending data to IoT Platform from a real device.**

   a. **Register a device, and develop the *device SDK***

   b. **The device connects to IoT Platform and reports data to IoT Platform.**

   c. **In the IoT Platform console, go to the Device Details page of the device. Click the Status tab to view the device property data.**



Example: Edit a script

The following describes the data parsing script format and content of a product. In this example, the device data is in hexadecimal notation, and the product has three properties: prop_float, prop_int16, and prop_bool.

1. **Create a product and select Do not parse/Custom as the data type. Then, define the following properties. For more information, see *Define features*.**

Identifier	Type	Value range	Read/write
prop_float	float	-100 to 100	Read/write
prop_int16	int32	-100 to 100	Read/write
prop_bool	bool	0: Enabled. 1: Disabled.	Read/write

2. **Define the communication protocol as follows:**

   Table 3-1: Upstream data request

Field	Number of bytes
Frame type	One
Request ID	Four
prop_int16	Two
prop_bool	One

Field	Number of bytes
prop_float	Four

Table 3-2: Upstream data response

Field	Number of bytes
Frame type	One
Request ID	Four
Result code	One

Table 3-3: Property setting request

Field	Number of bytes
Frame type	One
Request ID	Four
prop_int16	Two
prop_bool	One
prop_float	Four

Table 3-4: Property setting response

Field	Number of bytes
Frame type	One
Request ID	Four
Result code	One

3. **Edit the script.**

   You must define the following methods in the script:

   - `protocolToRawData`: **Convert Alink JSON formatted data to custom formatted data.**
   - `rawDataToProtocol`: **Convert custom formatted data to Alink JSON formatted data.**

   A script demo is as follows:

```
var COMMAND_REPORT = 0x00; //Devices report property
var COMMAND_SET = 0x01; //Set property
var COMMAND_REPORT_REPLY = 0x02; //Respond to the reported data
```

```
var COMMAND_SET_REPLY = 0x03; //Respond to the property setting
request
var COMMAD_UNKOWN = 0xff;    //Other command
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post'; //This
is a topic for devices to report property data to IoT Platform.
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'; //This is
a topic for for IoT Platform to send property management commands to
 devices.
var ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set'; //
This is a topic for devices to report property setting results to
IoT Platform.
/*
Sample data:
Upstream data
Input->
    0x000000000100320100000000
Output->
    {"method":"thing.event.property.post","id":"1","params":{"
prop_float":0,"prop_int16":50,"prop_bool":1},"version":"1.0"}

Property setting response
Input->
    0x0300223344c8
Output->
    {"code":"200","data":{},"id":"2241348","version":"1.0"}
*/
function rawDataToProtocol(bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (var i = 0; i < bytes.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = new Object();
    var fHead = uint8Array[0]; // command
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD; //The Alink
JSON formatted data topic for reporting properties
        jsonMap['version'] = '1.0'; //The fixed protocol version
field in the Alink JSON format
        jsonMap['id'] = '' + dataView.getInt32(1); //The request ID
in Alink JSON format
        var params = {};
        params['prop_int16'] = dataView.getInt16(5); //The value of
prop_int16
        params['prop_bool'] = uint8Array[7]; //The value of
prop_bool
        params['prop_float'] = dataView.getFloat32(8); //The value
of prop_float
        jsonMap['params'] = params; //The value for params in Alink
JSON format
    } else if(fHead == COMMAND_SET_REPLY) {
        jsonMap['version'] = '1.0'; //The fixed protocol version
field in the Alink JSON format
        jsonMap['id'] = '' + dataView.getInt32(1); //The request ID
value in Alink JSON format
        jsonMap['code'] = ''+ dataView.getUint8(5);
        jsonMap['data'] = {};
    }

    return jsonMap;
}
/*
Sample data:
Property setting
```

```
Input->
    {"method":"thing.service.property.set","id":"12345","version":"
1.0","params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1
}}
Output->
    0x0100003039014d0142f6e76d

Upstream data response
Input->
    {"method":"thing.event.property.post","id":"12345","version":"1.
0","code":200,"data":{}}
Output->
    0x0200003039c8
*/
function protocolToRawData(json) {
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD) //Set properties
    {
        var params = json['params'];
        var prop_float = params['prop_float'];
        var prop_int16 = params['prop_int16'];
        var prop_bool = params['prop_bool'];
        //Join raw data according to the custom protocol format
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET
)); //The command field
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id
))); //The ID in Alink JSON format
        payloadArray = payloadArray.concat(buffer_int16(prop_int16
)); //The value of prop_int16
        payloadArray = payloadArray.concat(buffer_uint8(prop_bool
)); //The value of prop_bool
        payloadArray = payloadArray.concat(buffer_float32(prop_float
)); //The value of prop_float
    } else if (method ==  ALINK_PROP_REPORT_METHOD) { //Response to
device upstream data
        var id = json['id'];
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET
)); // The command field
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id
))); //The ID in Alink JSON format
        payloadArray = payloadArray.concat(buffer_uint8(code));
    } else { //Other commands that will not be processed
        var id = json['id'];
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET
)); //The command field
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id
))); //The ID in Alink JSON format
        payloadArray = payloadArray.concat(buffer_uint8(code));
    }
    return payloadArray;
}
//The following lists some auxiliary functions:
function buffer_uint8(value) {
    var uint8Array = new Uint8Array(1);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setUint8(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int16(value) {
    var uint8Array = new Uint8Array(2);
    var dv = new DataView(uint8Array.buffer, 0);
```

```
    dv.setInt16(0, value);
    return [].slice.call(uint8Array);
}
function buffer_int32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setInt32(0, value);
    return [].slice.call(uint8Array);
}
function buffer_float32(value) {
    var uint8Array = new Uint8Array(4);
    var dv = new DataView(uint8Array.buffer, 0);
    dv.setFloat32(0, value);
    return [].slice.call(uint8Array);
}
```

Verify a data parsing script

After you edit a sample script, you can verify the correctness of the script. Enter analog data in the Analog Input box, and click Run. The system will call this script to parse the analog data. The parsed result will be displayed in the Parsing Results box at the right side of the page.

- **Parse the device-reported property data**

  Select `Upstreamed Device Data` **as the simulation type, enter the following hexadecimal data, and then click Run.**

  ```
  0x00002233441232013fa00000
  ```

  The data parsing engine will convert the hexadecimal data to JSON data as defined in the script. The result will be displayed in the Parsing Results area.

  ```
  {
      "method": "thing.event.property.post",
      "id": "2241348",
      "params": {
          "prop_float": 1.25,
          "prop_int16": 4658,
          "prop_bool": 1
      },
      "version": "1.0",
  }
  ```

- **Parse downstream data from IoT Platform to the device.**

  Select Received Device Data as the simulation type, enter the following JSON data, and then click Run.

  ```
  {
    "id": "12345",
    "version": "1.0",
    "code": 200,
    "method": "thing.event.property.post",
    "data": {}
  ```

```
}
```

The data parsing engine will convert the JSON data to the following hexadecimal data.

```
0x0200003039c8
```

- **Parse the property setting data from IoT Platform to devices.**

  Select `Received Device Data` as the simulation type, enter the following JSON data, and then click Run.

```
{
    "method": "thing.service.property.set",
    "id": "12345",
    "version": "1.0",
    "params": {
        "prop_float": 123.452,
        "prop_int16": 333,
        "prop_bool": 1
    }
}
```

  The data parsing engine converts JSON data to the following hexadecimal data.

```
0x0100003039014d0142f6e76d
```

- **Parse property setting results returned by the device.**

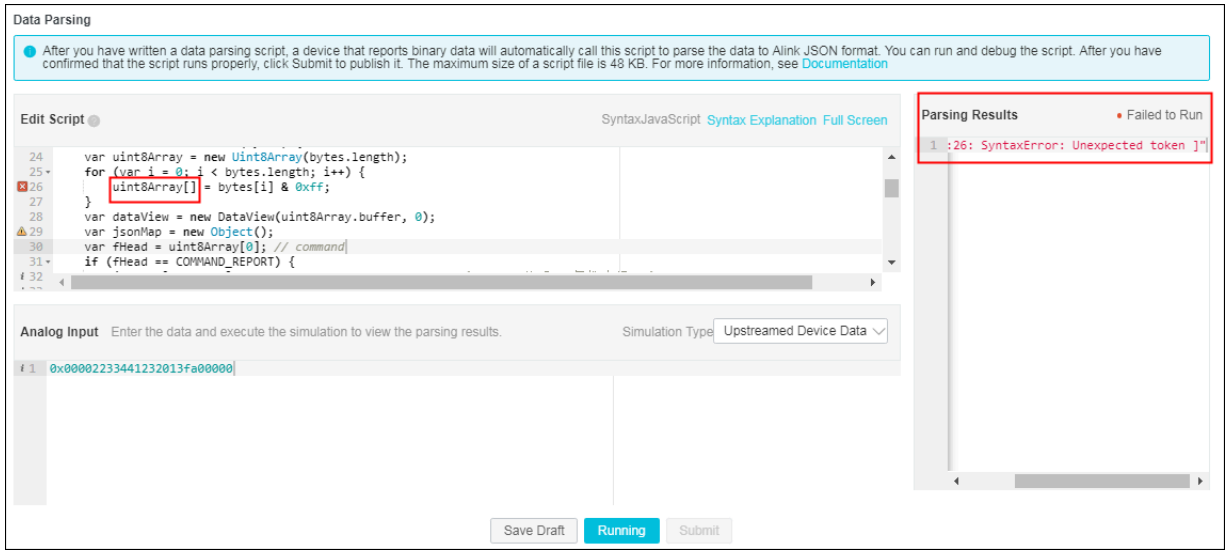
  Select `Upstreamed Device Data` as the simulation type, enter the following hexadecimal data, and then click Run.

```
0x0300223344c8
```

  The data parsing engine will convert the hexadecimal data to the following JSON data.

```
{
  "code": "200",
  "data": {}
  "id": "2241348",
  "version": "1.0",
}
```

If the script is incorrect, an error message is displayed in the Parsing Results area. You must troubleshoot the error according to the error message and modify the script code accordingly.



Debug a data parsing script in a local computer

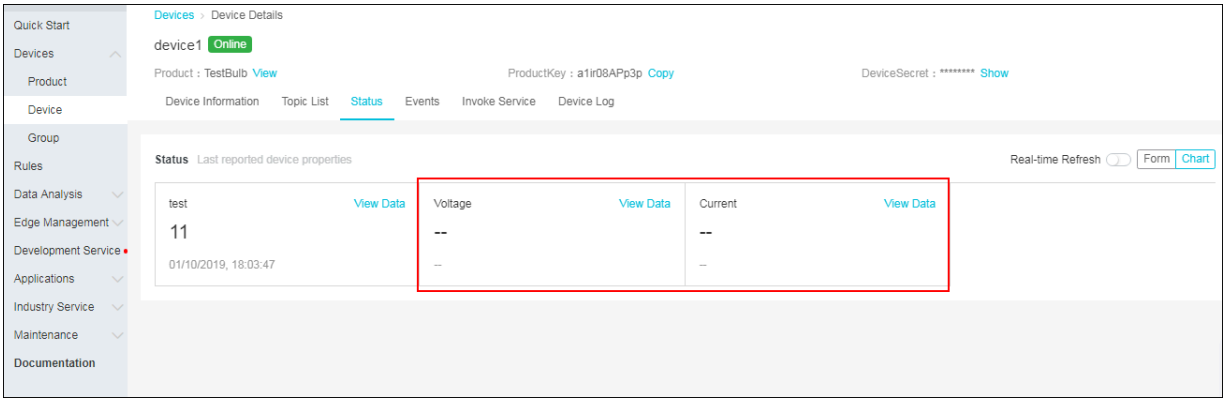
IoT Platform Data Parsing does not support debugging on the running platform. We recommend that you develop and debug the script locally and then paste the finished script into the online editor. You may use the following debugging method.

```
// Test Demo
function Test()
{
    //0x001232013fa00000
    var rawdata_report_prop = new Buffer([
        0x00, //The fixed command header. A value of 0 indicates a
property report message.
        0x00, 0x22, 0x33, 0x44, // The ID fields that identify the
request sequence.
        0x12, 0x32, //Two-byte value of prop_int16
        0x01, //One-byte value of prop_bool
        0x3f, 0xa0, 0x00, 0x00 //Four-byte value of prop_float
    ]);
    rawDataToProtocol(rawdata_report_prop);
    var setString = new String('{"method":"thing.service.property.
set","id":"12345","version":"1.0","params":{"prop_float":123.452, "
prop_int16":333, "prop_bool":1}}');
    protocolToRawData(JSON.parse(setString));
}
Test();
```

Troubleshoot issues

After a device is connected to IoT Platform and reports data, the reported data can be displayed in the IoT console if data parsing functions correctly. To view the data, go to the Device Details page of the device and click the Status tab.

In some occasions, after the device reports data, no data is displayed on the page, as shown in the following figure:



To view device logs: From the left-side navigation pane, choose Maintenance > Device Log and select the corresponding product. On the Device Log page, click the TSL Data Analysis tab. You can view the communication log between the device and IoT Platform.

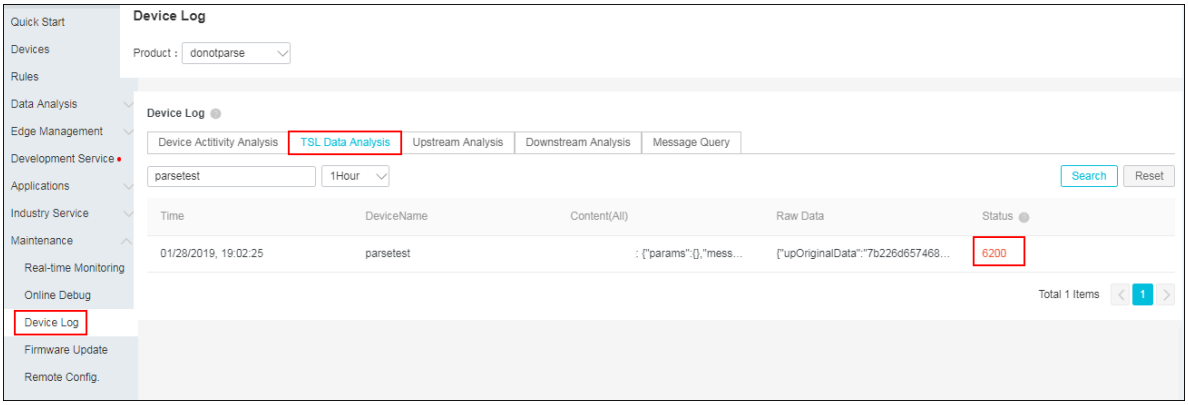
Use the following process to troubleshoot the issue:

1. **View the reported data on the Device Log page. Each log entry records the converted data and the original data.**
2. **Check the error codes according to the descriptions in *Device log*.**
3. **Troubleshoot the issue based on the error code, the script, and the reported data.**

The following lists some errors:

- **The data parsing script is not found.**

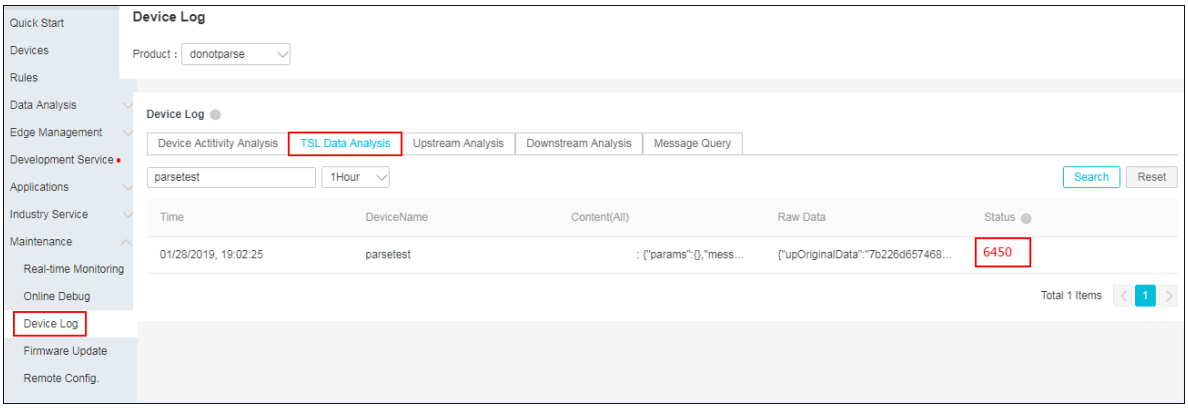
  As shown in the following figure, the error code is 6200. To check the description of the error, see *Device log*. The error code of 6200 indicates that no script was found. Check whether the data parsing script has been submitted in the console.



- **Alink method does not exist.**

  The error code is 6450. This error code is described in *Device log* as follows: The method parameter is not found in Alink data. This error occurs if the method

parameter is not found in the Alink data reported by the device or in the parsed result of Do not parse/Custom data.



You can check the raw data, for example:

```
17:54:19.064, A7B02C60646B4D2E8744F7AA7C3D9567, upstream-error -
bizType=OTHER_MESSAGE,params={"params":{}},result=code:6450,message:
alink method not exist,...
```

In the log, the error message is `alink method not exist`. If this error occurs, you must correct your script.

4 Tags

A tag is a custom identifier you set for a product, a device, or a device group. You can use tags to flexibly manage your products, devices and groups.

IoT often involves the management of a huge number of products and devices. How to distinguish various products and devices, and how to achieve centralized management become a challenge. Alibaba Cloud IoT Platform allows you to use tags to address these issues. The use of tags allows the centralized management of your various products, devices, and groups.

Therefore, we recommend that you create tags for your products, devices and device groups. The structure of a tag is `key: value`.

This article describes how to create product tags, device tags, and group tags in the console.

Note:

Each product, device, or group can have up to 100 tags.

Product tags

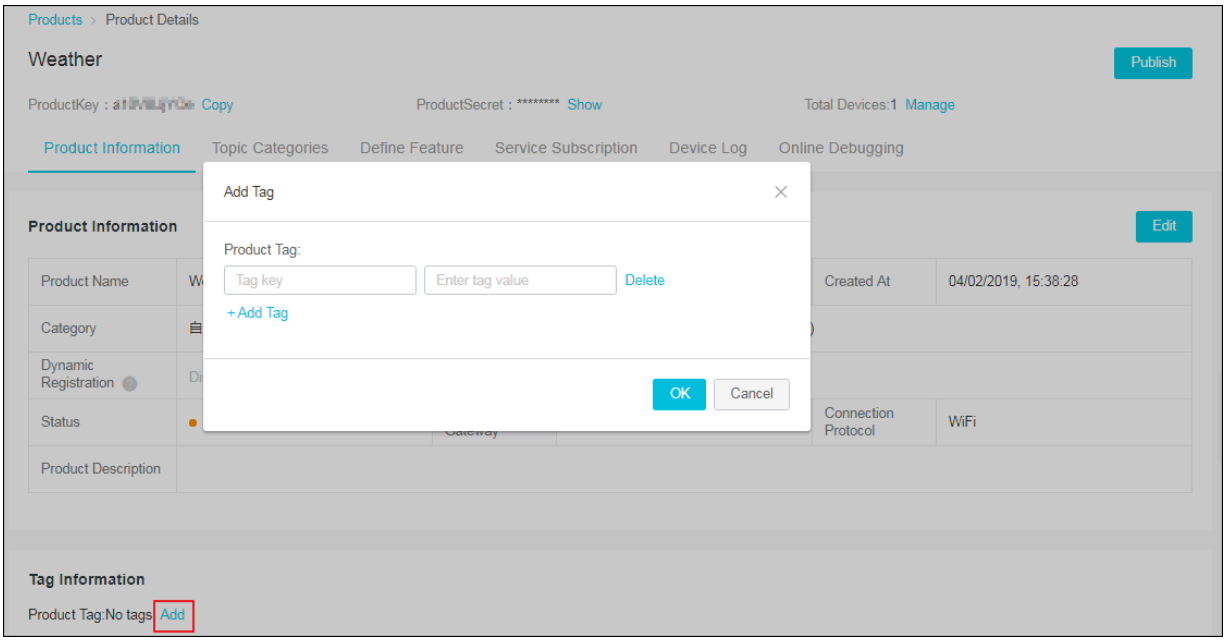
Product tags typically describe the information that is common to all devices of a product. For example, a tag can indicate a specific manufacturer, organization, physical size, or operating system. After a product has been created, you can create tags for it.

To create product tags in the console, follow these steps:

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 tags and click View.
4. Click Add under Tag Information.
5. In the dialog box, enter values for `Tag Key` and `Tag Value`, and then click OK.

Parameter	Description
Tag Key	A tag key can contain English letters, digits and dots (.), and cannot exceed 30 characters.

Parameter	Description
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as two characters.



Device tags

You can facilitate device management by creating unique tags for devices. For example, you can use the device feature information as tags, such as `PowerMeter:room201` for the electricity meter of room 201.

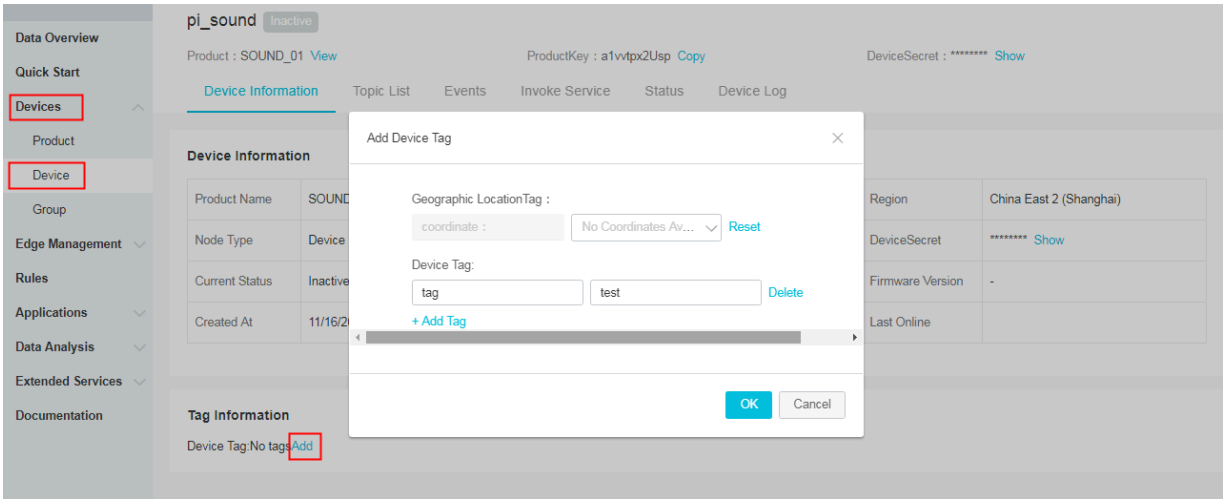
Device tags always follow the devices. You can include tag information in the messages reported to IoT Platform by devices. When you use the rules engine to forward these messages to other Alibaba Cloud services, the tag information is also forwarded to the targets.

To create device tags in the console, follow these steps:

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click Devices > Device.
3. On the Devices page, find the device for which you want to create tags, click View to go to the Device Details page.
4. Click Add under Tag Information.

5. In the dialog box, enter values for `Tag Key` and `Tag Value`, and then click OK.

Parameter	Description
Tag Key	A tag key can contain English letters , digits, and dots (.), and can be 2-30 characters in length.
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as 2 characters.



Group tags

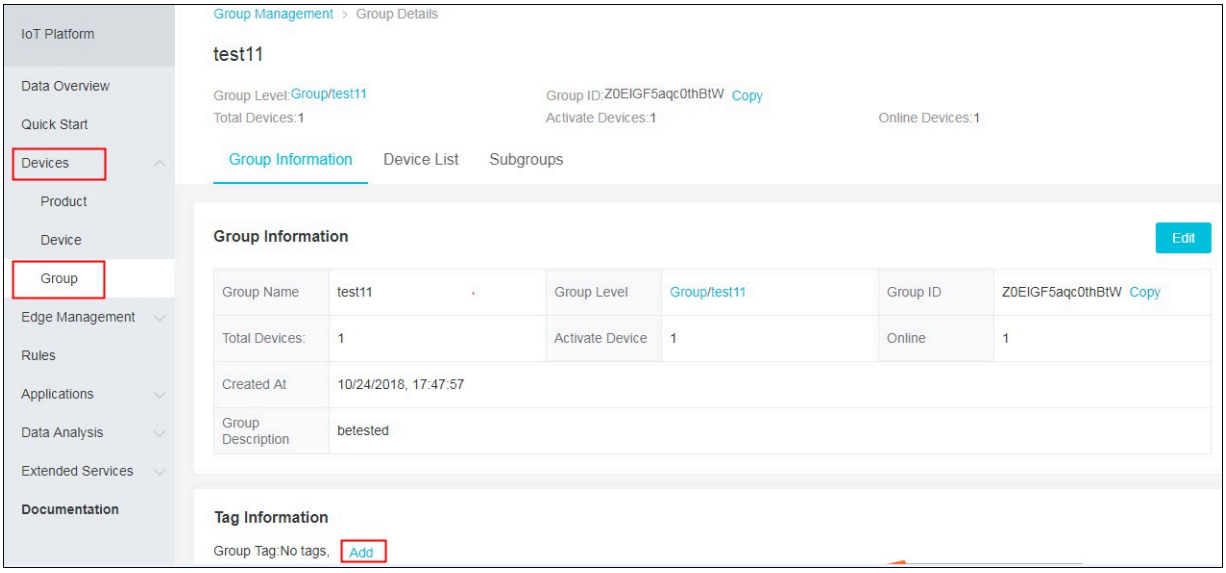
You can manage devices across products by grouping your devices. A group tag typically describe the general information of devices in the group and the sub-groups. For example, you can use region information as a group tag. After you have created a group, you can create tags for it.

To create group tags, follow these steps:

1. Log on to the *IoT Platform console*.
2. In the left-side navigation pane, click Devices > Group.
3. On the Group Management page, find the group for which you want to create tags and click View.
4. Click Add under Tag Information.

5. In the dialog box, enter values for `Tag Key` and `Tag Value`, and then click OK.

Parameter	Description
Tag Key	A tag key can contain English letters , digits, and dots (.), and can be 2-30 characters in length.
Tag Value	A tag value can contain Chinese characters, English letters, digits, underscores (_), hyphens (-), colons (:), and dots (.), and cannot exceed 128 characters. A Chinese character is counted as 2 characters.



Manage tags in batch

In the console, you only can create, modify, and delete tags one by one. IoT Platform provides APIs for managing tags in batch. In addition, IoT Platform provides APIs for querying products, devices, and groups based on tags. For more information about tag related APIs, see the documents in API reference.

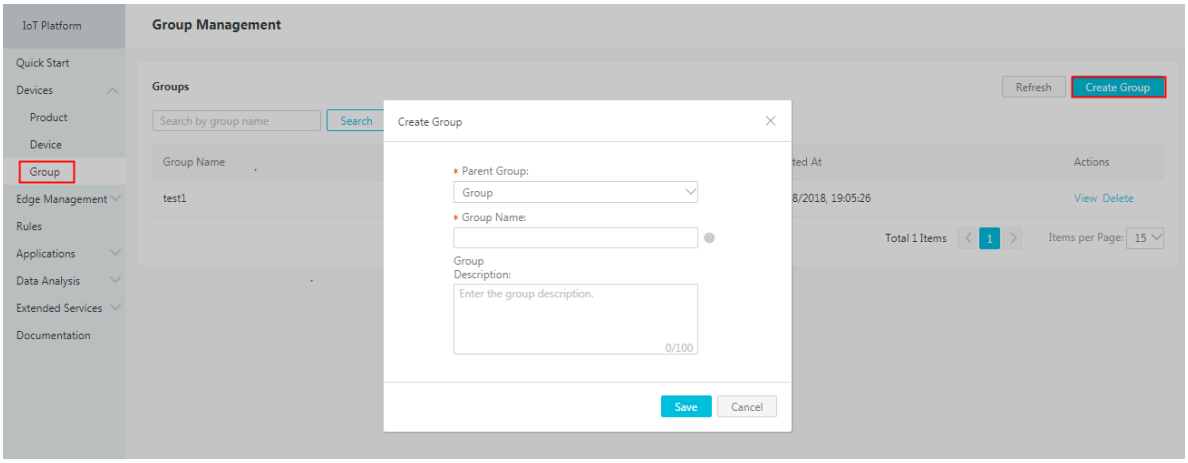
5 Device group

IoT Platform supports device groups. You can assign devices from different products to the same group. This article introduces how to create and manage device groups in the IoT Platform console.

Procedure

1. **Log on to the *IoT Platform console*.**
2. **Click Devices > Group.**
3. **On the group management page, click Create Group, enter group information, and then click Save.**

> **Note:**
> **You can create up to 1,000 groups (including parent groups and subgroups) .**

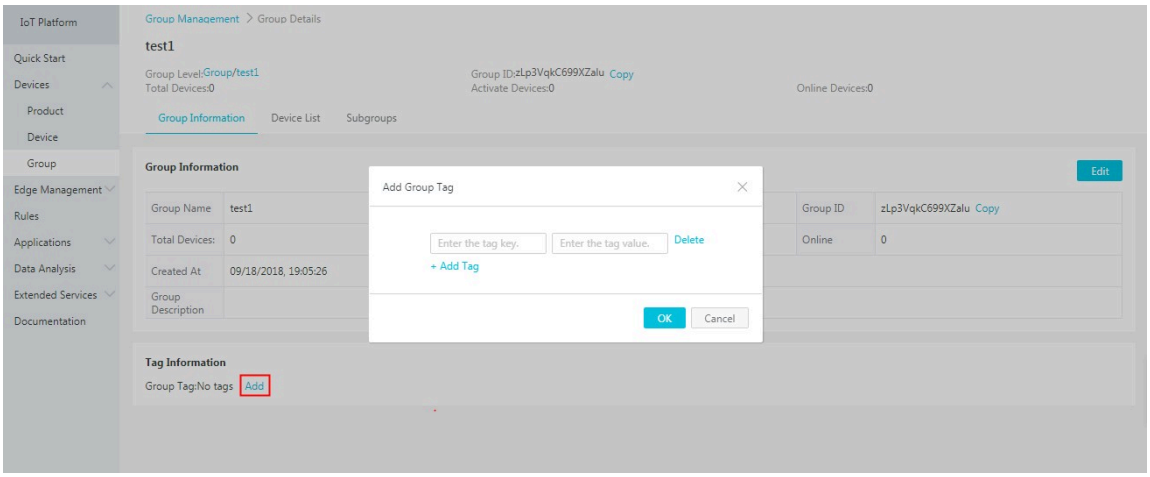


The parameters are as follows:

- **Parent Group: Select a group type.**
  - **Group: Indicates that the group to be created is a parent group.**
  - **Select an existing group: Specifies a group as the parent group and creates a subgroup for it.**
- **Group Name: Enter a name for the group. A group name can be 4 to 30 characters in length and can include Chinese characters, English letters, digits and underscores (_) . The group name must be unique among the groups for an account, and cannot be modified once the group has been created.**
- **Group Description: Describes the group. Can be left empty.**

4. On the Group Management page, click View to view the Group Details page of the corresponding group.
5. (Optional) Add tags for the group. Tags can be used as group identifiers when you manage your groups.
   a) Click Add under Tag Information, and then enter keys and values of tags.
   b) Click OK to create all the entered tags.

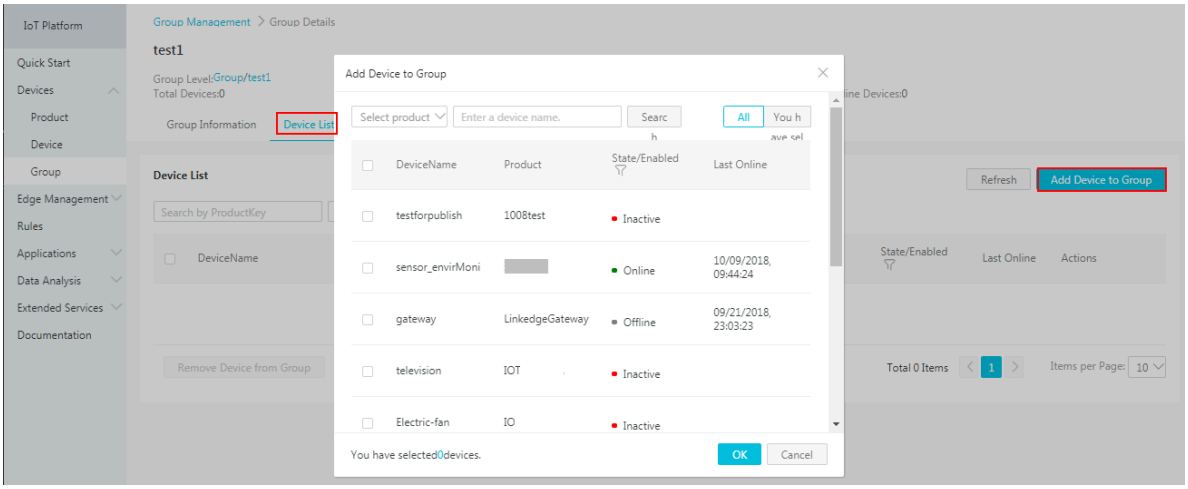
> **Note:**
> You can add up to 100 tags for a group.



6. Click Device List > Add Device to Group. Select the devices that you want to add to the group.

> **Note:**
> - You can add up to 1,000 devices at a time. You can add up to 20,000 devices for a group in total.

- A device can be included in a maximum of 10 groups.



There are two buttons at the upper-right corner of the Add Device to Group page:.

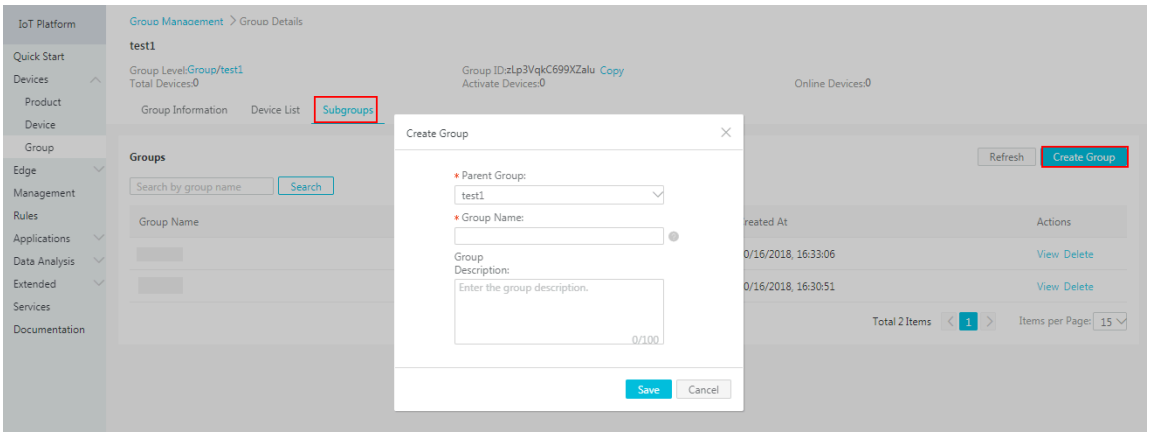
- Click All to display all the devices.
- Click You have selected to display the devices you have selected.

7. (Optional) Click Subgroups > Create Group to add a subgroup for the group.

   Subgroups are used to manage devices in a more specific manner. For example , you can create subgroups such as "SmartKitchen" and "SmartBedroom" for a

parent group "SmartHome", and then you can manage your kitchen devices and bedroom devices separately. The procedure is as follows:

a) Select the parent group, enter a group name and description, and click Save.



b) On the Subgroups page of the parent group , click View to view the corresponding Group Details page.

c) Click Device List > Add Device to Group, and then add devices for the subgroup.

After creating the subgroup and adding devices for it, you can then manage it. You can also create sub-subgroups within the subgroup.

Note:

- A group can include up to 100 subgroups.
- Only three layers of groups are supported: parent group>subgroup>sub-subgroup.
- A group can only be a subgroup of one parent group.
- You can not change the relationships between a parent group and its subgroups once they have been created. If you want to change the relationsh ips, delete the existing subgroups and create new ones.
- You cannot delete a group that has subgroups. You must delete all its subgroups before deleting the parent group.

6 Device shadows

6.1 Device Shadow overview

IoT Platform provides the Device Shadow function to cache property information for a device. If the device is online, the device can directly receive commands from IoT Platform. If the device is offline, the device can actively request for cached commands from IoT Platform after it comes online again.

A device shadow is a JSON file that is used to store the reported status and desired status information for a device.

Each device has only one shadow. A device can obtain and set the shadow over MQTT for status synchronization. The synchronization is bi-directional, either from the shadow to the device or from the device to the shadow.

Scenarios

- **Scenario 1: In an unstable network, a device frequently disconnects from and reconnects to IoT Platform.**

  The device frequently disconnects from and reconnects to IoT Platform due to network instability. When an application that interacts with the device requests the current device status, the device is offline, which leads to a request failure. When the device is reconnected, the application fails to initiate another device status request.

  The Device Shadow function can synchronize with the device to update and store the latest device status information in the device shadow. The application can obtain the current device status information from the device shadow despite of the connection status.

- **Scenario 2: Multiple applications simultaneously request the device status information.**

  In a stable network, a device must respond to each status request from multiple applications, even if the responses are the same. The device may be overloaded with the requests.

  By using the Device Shadow function, the device only needs to synchronize status information to the device shadow that is stored in IoT Platform. Applications can request the latest device status information from the device shadow instead of the target device. In this way, applications are decoupled from the device.
- **Scenario 3: Device disconnection**
  - **In an unstable network, a device frequently disconnects from and reconnects to IoT Platform. When an application sends a control command to the device, the device is offline and the command fails to be dispatched to the device.**
    - **Quality of Service 1 or 2 (QoS 1 or 2) may solve this issue. However, we recommend that you do not use this method. This method increases the workload of the server.**
    - **By using the Device Shadow function, IoT Platform stores the control commands from the application to the device shadow. Each command is stored with the timestamp when the command was received. After the device is reconnected to IoT Platform, the device obtains these commands and checks the timestamp of each command to determine whether to run the command.**
  - **A device goes offline and fails to receive commands from the application. When the device is reconnected, the device runs only the valid commands by checking the timestamp of each command that is pulled from the device shadow.**

View and update a device shadow

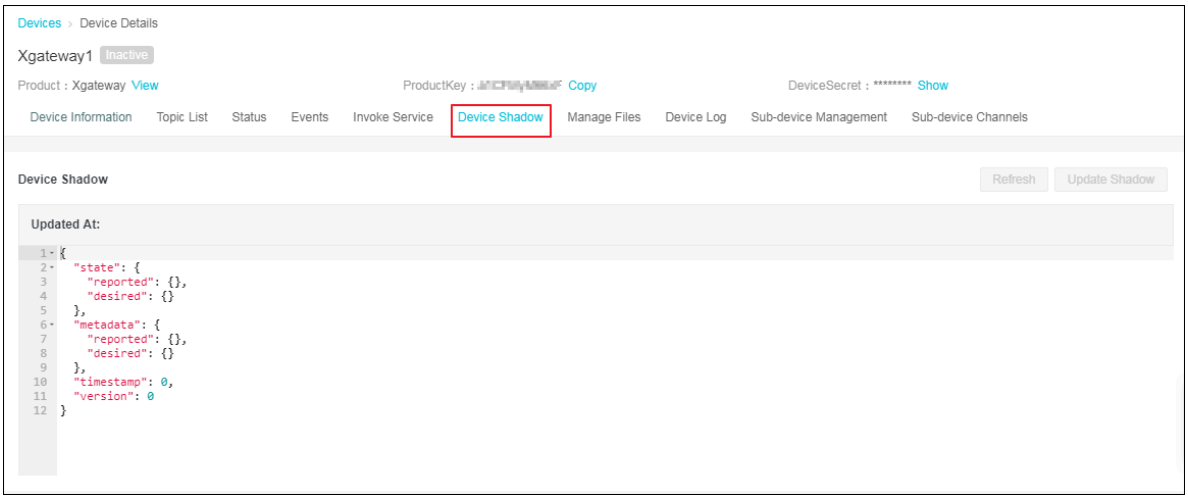
You can view and update the shadow of a device in the IoT Platform console.

Procedure:

1. **Log on to the** *IoT Platform console* **.**
2. **From the left-side navigation pane, choose Devices > Device.**
3. **Click View next to the corresponding device. The Device Details page appears.**

4. **Click the Device Shadow tab.**

   You can view the shadow that contains the latest information that is reported by the device.



5. **Click Update Shadow, and enter the desired status information in the "desired" section.**

   For more information about the shadow file format, see *Device shadow JSON format*.

   The device obtains the desired status information by subscribing to a specific topic. When the device is online, IoT Platform pushes the desired value to the device in real time.

   When the device is offline, the device's shadow caches the desired status information. After the device comes online again, it actively pulls the latest desired status information from IoT Platform.

Related API operations

Obtain a device shadow: *#unique_30*

Update a device shadow: *#unique_31*

6.2 Device shadow JSON format

Format of the device shadow JSON file

The format is as follows:

```
{
"state": {
"desired": {
"attribute1": integer2,
```

```
"attribute2": "string2",
...
"attributeN": boolean2
},
"reported": {
"attribute1": integer1,
"attribute2": "string1",
...
"attributeN": boolean1
}
},
"metadata": {
"desired": {
"attribute1": {
"timestamp": timestamp
},
"attribute2": {
"timestamp": timestamp
},
...
"attributeN": {
"timestamp": timestamp
}
},
"reported": {
"attribute1": {
"timestamp": timestamp
},
"attribute2": {
"timestamp": timestamp
},
...
"attributeN": {
"timestamp": timestamp
}
}
},
"timestamp": timestamp,
"version": version
}
```

The JSON properties are described in *Table 6-1: JSON property*.

Table 6-1: JSON property

Property	Description
desired	**The desired status of the device.** **The application writes the desired property of the device, without accessing the device.**
reported	**The status that the device has reported. The device writes data to the reported property to report its latest status.** **The application obtains the status of the device by reading this property.**

Property	Description
metadata	The device shadow service automatically updates metadata according to the updates in the device shadow JSON file. State metadata in the device shadow JSON file contains the timestamp of each property. The timestamp is represented as epoch time to obtain exact update time.
timestamp	The latest update time of the device shadow JSON file.
version	When you request updating the version of the device shadow, the device shadow checks whether the requested version is later than the current version. If the requested version is later than the current one, the device shadow updates to the requested version. If not, the device shadow rejects the request. The version number is increased according to the version update to ensure the latest device shadow JSON file version.

Example of the device shadow JSON file:

```
{
"state" : {
"desired" : {
"color" : "RED",
"sequence" : [ "RED", "GREEN", "BLUE" ]
},
"reported" : {
"color" : "GREEN"
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
},
"sequence" : {
"timestamp" : 1469564492
}
},
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
```

```
}
```

Empty properties

- **The device shadow JSON file contains the desired property only when you have specified the desired status. The following device shadow JSON file, which does not contain the desired property, is also effective:**

```
{
"state" : {
"reported" : {
"color" : "red",
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

- **The following device shadow JSON file, which does not contain the reported property, is also effective:**

```
{
"state" : {
"desired" : {
"color" : "red",
}
},
"metadata" : {
"desired" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492,
"version" : 1
}
```

Array

The device shadow JSON file can use an array, and must update this array as a whole when the update is required.

- **Initial status:**

```
{
"reported" : { "colors" : ["RED", "GREEN", "BLUE" ] }
```

```
}
```

- **Update:**

```
{
"reported" : { "colors" : ["RED"] }
}
```

- **Final status:**

```
{
"reported" : { "colors" : ["RED"] }
}
```

6.3 Device shadow data stream

IoT Platform predefines two topics for each device to enable data transmission. The predefined topics have fixed formats.

- **Topic:** */shadow/update/${YourProductKey}/${YourDeviceName}*

  Devices and applications publish messages to this topic. When IoT Platform receives messages from this topic, it will extract the status information in the messages and will update the status to the device shadow.

- **Topic:** */shadow/get/${YourProductKey}/${YourDeviceName}*

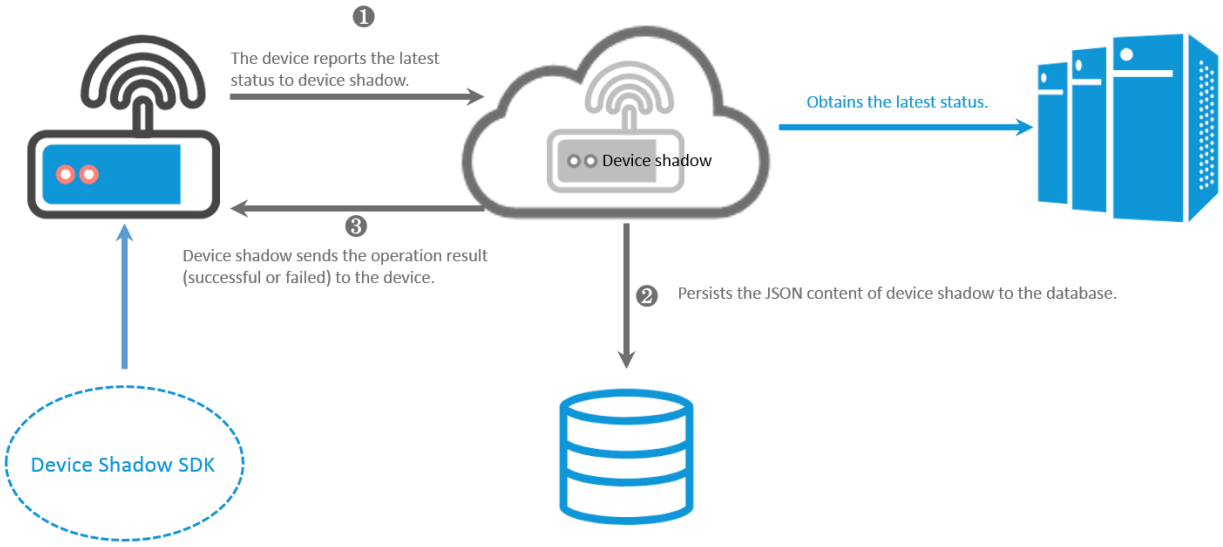
  The device shadow updates the status to this topic, and the device subscribes to the messages from this topic.

Take a lightbulb device of a product bulb_1 as an example to introduce the communication among devices, device shadows, and applications. In the following example, the ProductKey is aliDeEf** and the DeviceName is lightbulb. The device publishes messages to and subscribes to messages of the two custom topics using the method of QoS 1.**

Device reports status automatically

The flow chart is shown in *Figure 6-1: Device reports status automatically*.

Figure 6-1: Device reports status automatically



1. **When the lightbulb is online, the device uses topic** `/shadow/update/aliDeEf****/lightbulb` **to report the latest status to the device shadow.**

   Format of the JSON message:

```
{
"method": "update",
"state": {
"reported": {
"color": "red"
}
},
"version": 1
}
```

   The JSON parameters are described in *Table 6-2: Parameter description*.

   Table 6-2: Parameter description

Parameter	Description
`method`	**The operation type when a device or application requests the device shadow.** **When you update the status, This parameter** `method` **is required and must be set to** `update`**.**

Parameter	Description
state	The status information that the device sends to the device shadow. The reported field is required. The status information is synchronized to the reported field of the device shadow.
version	The version information contained in the request. The device shadow only accepts the request and updates to the specified version when the new version is later than the current version.

2. When the device shadow accepts the status reported by the device lightbulb, the JSON file of device shadow is successfully updated.

```
{
"state" : {
"reported" : {
"color" : "red"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564492
}
}
},
"timestamp" : 1469564492
"version" : 1
}
```

3. After the device shadow has been updated, it will return the result to the device (lightbulb) by sending a message to the topic `/shadow/get/aliDeEf****/lightbulb`.

   - If the update is successful, the message is as follows:

```
{
"method":"reply",
"payload": {
"status":"success",
"version": 1
},
"timestamp": 1469564576
}
```

   - If an error occurred during the update, the message is as follows:

```
{
"method":"reply",
"payload": {
"status":"error",
"content": {
"errorcode": "${errorcode}",
"errormessage": "${errormessage}"
}
},
"timestamp": 1469564576
}
```

Error codes are described in *Table 6-3: Error codes*.

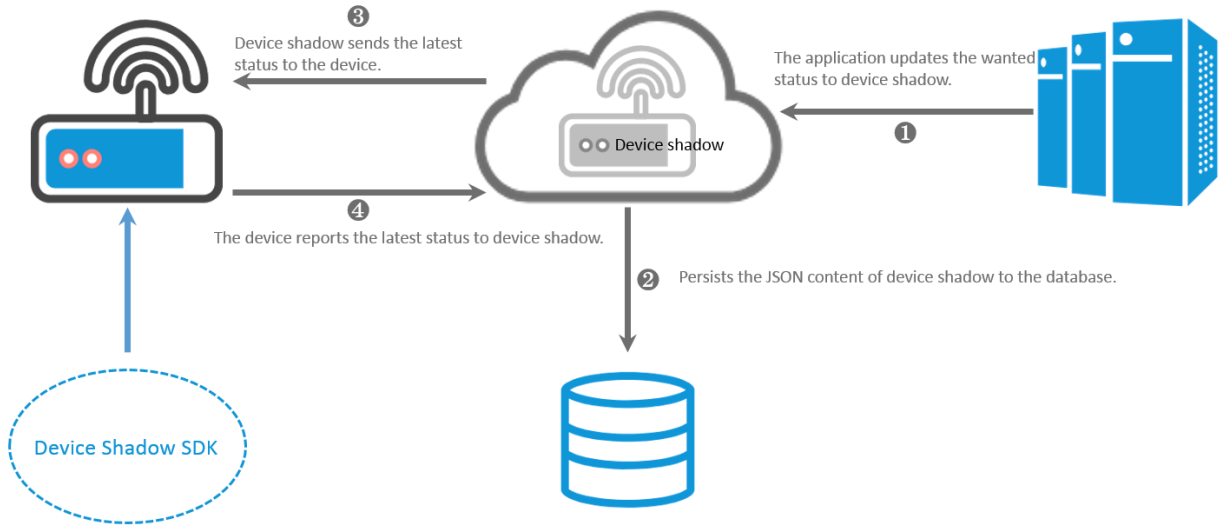
Table 6-3: Error codes

errorCode	errorMessage
400	Incorrect JSON file.
401	The method field is not found.
402	the state field is not found.
403	Invalid version field.
404	The reported field is not found.
405	The reported field is empty.
406	Invalid method field.
407	The JSON file is empty.
408	The reported field contains more than 128 attributes.
409	Version conflict.
500	Server exception.

Application changes device status

The flow chart is shown in *Figure 6-2: Application changes device status*.

Figure 6-2: Application changes device status



1. **The application calls the API** *#unique_31* **to send a command to the device shadow to change the status of the lightbulb.**

   The command is sent to topic `/shadow/update/aliDeEf****/lightbulb/`**. The message is as follows:**

```
{
"method": "update",
"state": {
"desired": {
"color": "green"
}
},
"version": 2
}
```

2. **The application sends an update request to update the device shadow JSON file. The device shadow JSON file is changed to:**

```
{
"state" : {
"reported" : {
"color" : "red"
},
"desired" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
```

```
"color" : {
"timestamp" : 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
}
}
},
"timestamp" : 1469564576,
"version" : 2
}
```

3. **After the update, the device shadow sends a message to the topic** `/shadow/get/aliDeEf****/lightbulb` **and returns the result of update to the device. The result message is created by the device shadow.**

```
{
"method":"control",
"payload": {
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
"timestamp": 1469564492
}
},
"desired" : {
"color" : {
"timestamp" : 1469564576
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

4. **When the device lightbulb is online and has subscribed to the topic** `/shadow/get/aliDeEf****/lightbulb`**, the device receives the message and changes its color to green according to the** `desired` **field in the request file. After the device has updated the status, it will report the latest status to the cloud.**

```
{
method": "update",
"state": {
"reported": {
"color": "green"
```

```
}
},
"version": 3
}
```

If the timestamp shows that the command has expired, you give up the update.

5. **After the latest status has been reported successfully, the device sends a message to the topic `/shadow/update/aliDeEf****/lightbulb` to empty the property of desired field. The message is as follows:**

```
{
"method": "update",
"state": {
"desired":"null"
},
"version": 4
}
```

6. **After the status has been reported, the device shadow is synchronously updated. The device shadow JSON file is as follows:**

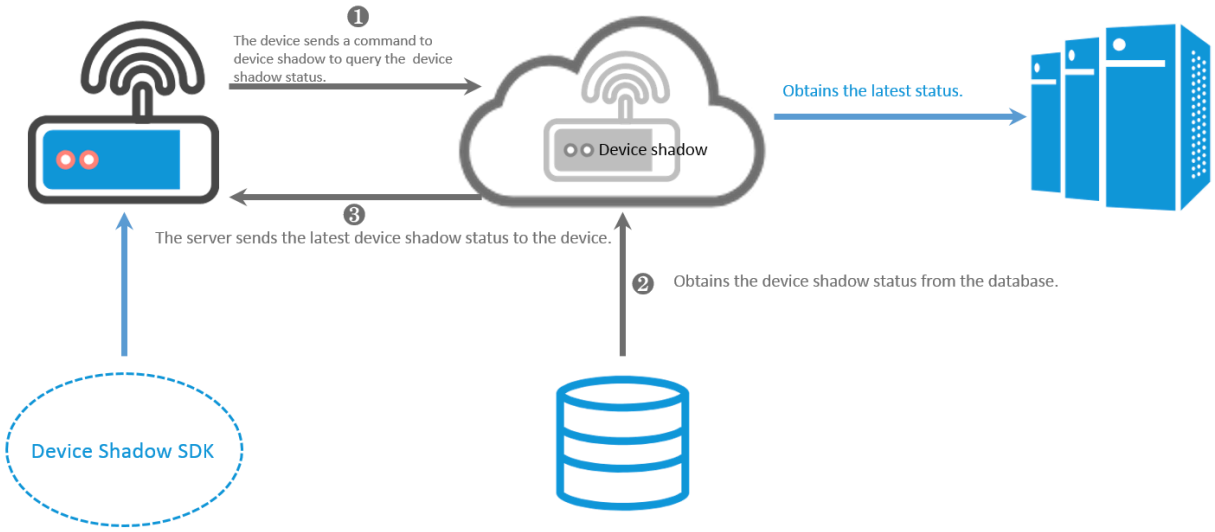
```
{
"state" : {
"reported" : {
"color" : "green"
}
},
"metadata" : {
"reported" : {
"color" : {
"timestamp" : 1469564577
}
},
"desired" : {
"timestamp" : 1469564576
}
},
"version" : 4
```

```
}
```

Devices request for device shadows

The flow chart is shown in *Figure 6-3: The device requests for device shadow*.

Figure 6-3: The device requests for device shadow



1. **The device lightbulb sends a message to the topic `/shadow/update/aliDeEf****/lightbulb` and obtains the latest status saved in the device shadow. The message is as follows:**

```
{
"method": "get"
}
```

2. **When the device shadow receives above message, the device shadow sends a message to the topic `/shadow/get/aliDeEf****/lightbulb`. The message is as follows:**

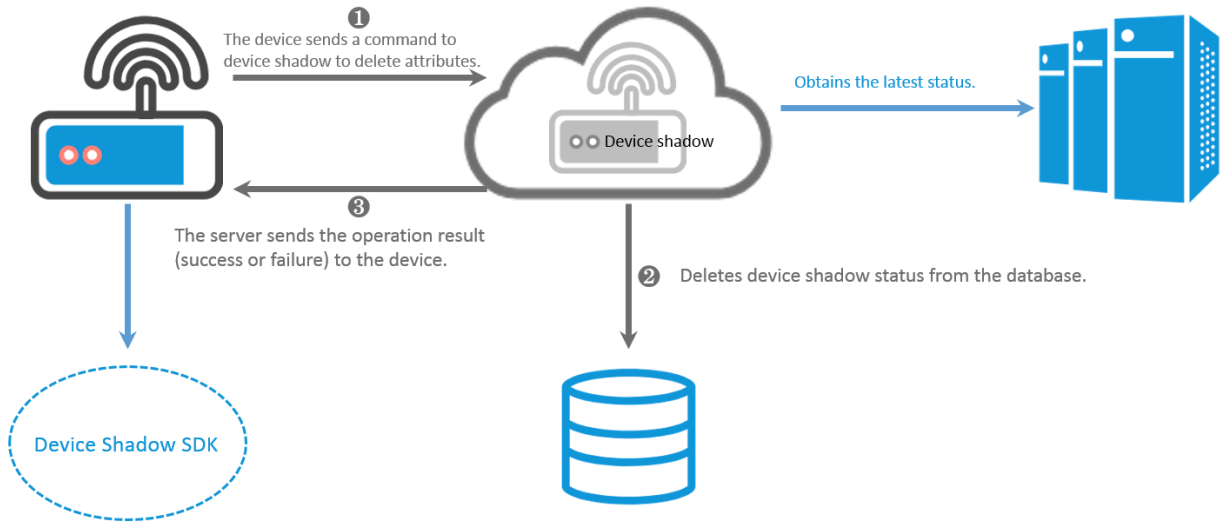
```
{
"method":"reply",
"payload": {
"status":"success",
"state": {
"reported": {
"color": "red"
},
"desired": {
"color": "green"
}
},
"metadata": {
"reported": {
"color": {
```

```
"timestamp": 1469564492
}
},
"desired": {
"color": {
"timestamp": 1469564492
}
}
}
},
"version": 2,
"timestamp": 1469564576
}
```

Devices delete device shadow attributes

The flow chart is shown in *Figure 6-4: Delete device shadow attributes*.

Figure 6-4: Delete device shadow attributes



The device lightbulb is to delete the specified attributes saved in the device shadow. The device sends a JSON message to the topic `/shadow/update/aliDeEf****/lightbulb`. See the message in the following example.

To delete attributes, set the value of `method` to `delete` and set the values of the attributes to `null`.

- **Delete one attribute:**

```
{
"method": "delete",
"state": {
"reported": {
"color": "null",
"temperature":"null"
}
```

```
},
"version": 1
}
```

- **Delete all the attributes:**

```
{
"method": "delete",
"state": {
"reported":"null"
},
"version": 1
}
```

7 Manage files

IoT Platform allows devices to upload files over HTTP/2 channels to the Alibaba Cloud IoT Platform server for storage. After a file is uploaded, you can download and delete the file in the IoT Platform console.

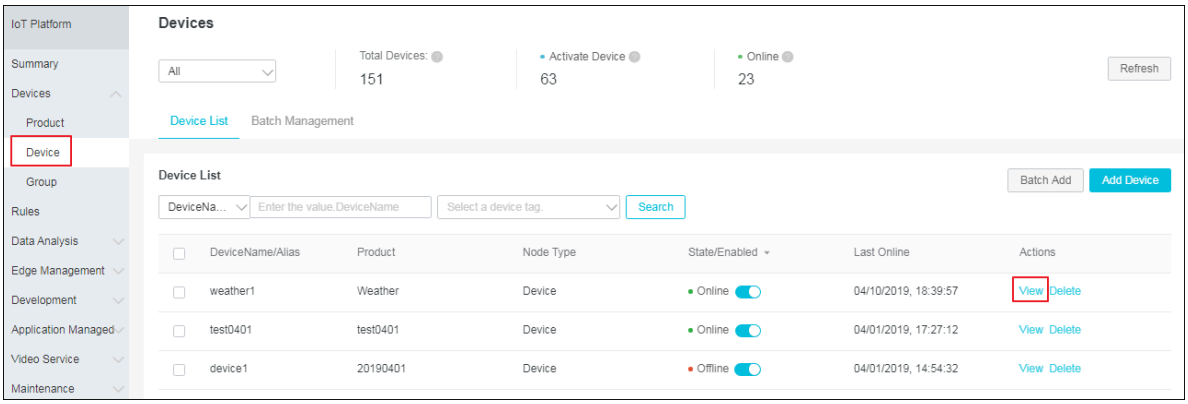
Prerequisites

- **The device is connected to IoT Platform.**

  For more information about device SDK development, see *Link Kit SDK documentation*.
- **The HTTP/2 file upload function is compiled and configured on the device.**

Procedure

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

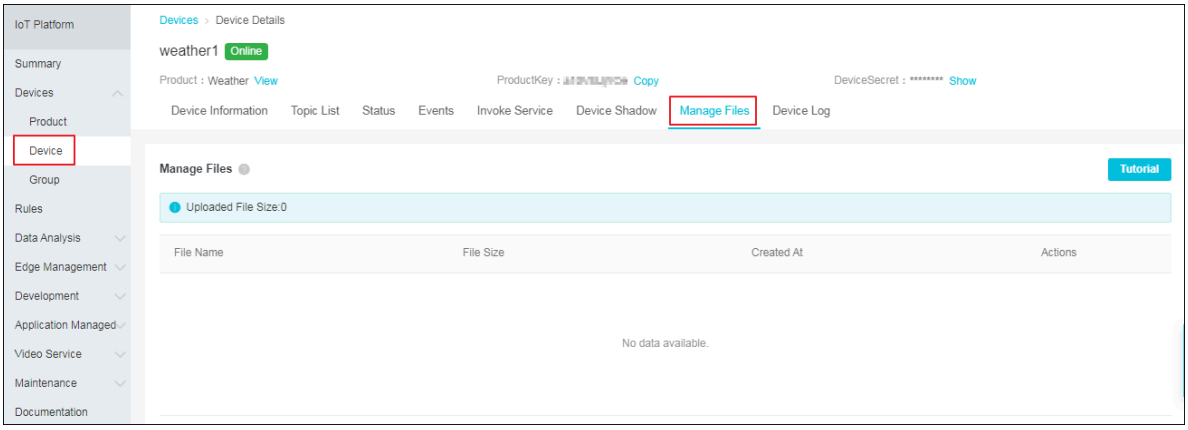


3. **On the Device Details page, click the Manage Files tab.**

   On the Manage Files tab page, you can view the files that were uploaded by the device through the HTTP/2 channel.

Note:

The maximum file size that can be stored on the IoT Platform server for each Alibaba Cloud account is 1 GB. The maximum number of files that can be stored for each device is 1,000.



You can perform the following operations on an uploaded file:

Operation	Description
Download	Download the file to your local device.
Delete	Delete the file.

In addition to file management in the console, you can also query or delete files by calling the following cloud API operations: *#unique_34*, *#unique_35*, and *#unique_36*.

8 Configure the NTP service

IoT Platform provides the NTP service to resolve the following issues on embedded devices: limited resources, no NTP service available in the system, and inaccurate timestamp.

How NTP works

Based on the NTP protocol, IoT Platform acts as the NTP server. A device sends a message of a specific topic to IoT Platform with the sending time in the message payload. IoT Platform adds the message receiving time and response sending time to the payload of the response packet. After the device receives the response, the device records its local time when it receives the response. All these four time will be used to calculate the time difference between the device and IoT Platform to obtain the exact current time on the device.

Note:

The NTP service can be used for time calibration only after the device is connected to IoT Platform.

An embedded device, which does not have an accurate time after it is powered, cannot pass the certificate verification during the TLS connection establishment process. If it does not connect to IoT Platform, this issue cannot be resolved by the NTP service of IoT Platform.

NTP service procedure

Request topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/request`

Response topic: `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`

Note:

ProductKey and DeviceName are part of the device certificate, which can be obtained from the IoT Platform console.

1. **The device subscribes to the topic:** `/ext/ntp/${YourProductKey}/${YourDeviceName}/response`.

2. **The device publishes a QoS 0 message with the current timestamp of the device in the payload to the topic `/ext/ntp/${YourProductKey}/${YourDeviceName}/request`. For example:**

```
{
    "deviceSendTime":"1571724098000"
}
```

> **Note:**
> **The data type of the timestamp, which supports Long and String.**
>
> **Only QoS 0 messages are supported for this feature.**

3. **The device receives a response from the NTP server. The payload includes the following information:**

```
{
    "deviceSendTime":"1571724098000",
    "serverRecvTime":"1571724098110",
    "serverSendTime":"1571724098115",
}
```

4. **The device calculates the current exact Unix time.**

   The time when the device receives the message from the server is recorded as $ {deviceрecvtime}, and the exact time on the device is: `($ {Serverrecvtime} + $ {serversendtime} + $ {devicerecvtime}-$ {devicesendtime})/2`

Example

In this example, the device time is 1571724098000, the server time is 1571724098100, the network delay is 10 ms, and the time spent before the server sends a response for a received request is 5 ms.

	Device time	Server time
deviceSend	1571724098000 (deviceSendTime)	1571724098100
serverReceive	1571724098010	1571724098110 (serverRecvTime)
serverSend	1571724098015	1571724098115 (serverSendTime)
deviceReceive	1571724098025 (deviceRecvTime)	1571724098125

The device calculates the current exact Unix time as (1571724098110 + 1571724098115 +15717240980255 - 1571724098000)/2 = 1571724098125.

If the device directly uses the timestamp returned from the server, the device will have a time error due to the network delay.

9 Gateways and sub-devices

9.1 Gateways and sub-devices

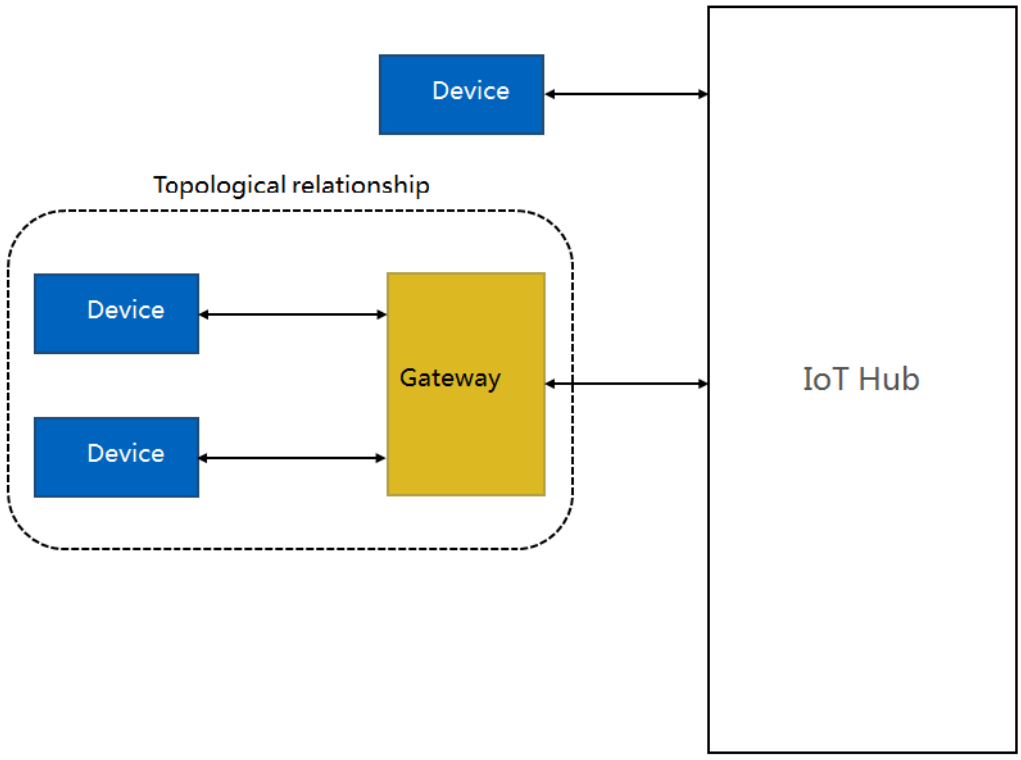
IoT Platform allows devices to connect to it directly, or be mounted as sub-devices to gateways that connect to IoT Platform.

Gateways and devices

When you create a product, you must select a node type for the devices of the product. Currently, IoT Platform supports two node types, `Device` and `Gateway`.

- **Device: Devices of this node type cannot be mounted with sub-devices, but can be connected directly to the IoT Platform or be mounted as sub-devices to gateways.**
- **Gateway: Devices of this node type can connect to IoT Platform directly and can be mounted with sub-devices. Gateways are then used to manage sub-devices , maintain topological relationships with sub-devices, and synchronize these topological relationships to IoT Platform.**

The topological relationship between a gateway and its sub-devices is shown in the following figure:



Connect gateways and sub-devices to IoT Platform

Once a gateway has been connected to IoT Platform, the gateway will synchroniz e its topological relationships with its sub-devices to IoT Platform. A gateway supports device authentication, message reporting, instruction receiving, and other communications with IoT Platform for all its sub-devices. That is, sub-devices are managed by their corresponding gateway.

1. **Develop the gateway and connect the gateway to Iot Platform.**

   For more information about how to connect gateways to IoT Platform, see *Link Kit SDK*.

2. **You can connect sub-devices to IoT Platform using either of the following two methods:**

   - **The *Unique-certificate-per-device authentication* method. This method requires you to install the device certificates (namely, the ProductKey, DeviceName, and DeviceSecret) in the physical sub-devices, and then connect the sub-devices to IoT Platform.**
   - **The *Unique-certificate-per-product authentication* method. This method requires you to enable Dynamic Registration on the product details page and register devices in the IoT Platform console. Then, when a physical sub-device is being connected, the gateway will initiate a connection request to IoT Platform for the sub-device. IoT Platform then verifies the sub-device information. If the verification passes, IoT Platform will assign the DeviceSecret to the sub-device. The sub-device then receives all the required information (namely, the ProductKey, DeviceName, and DeviceSecret) to successfully connect to IoT Platform.**

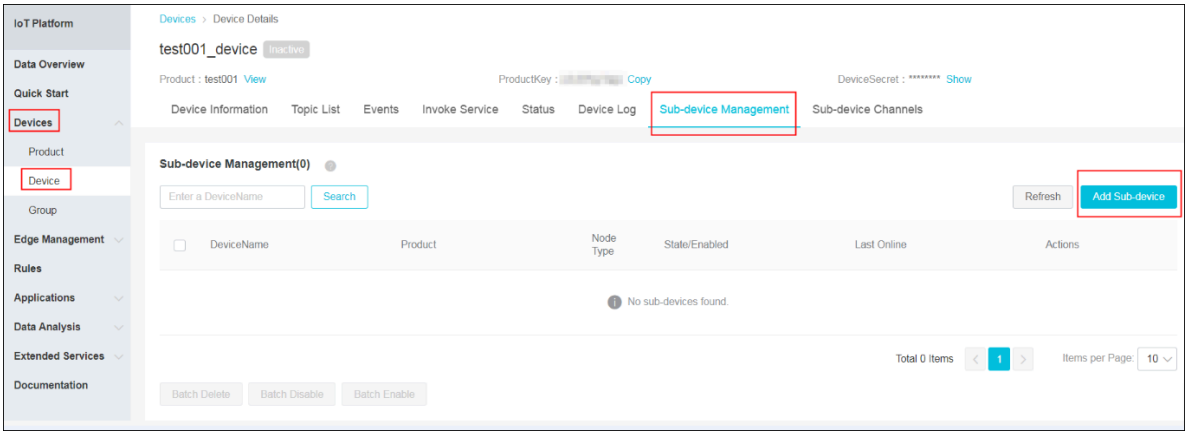
9.2 Sub-device management

You can add sub-devices to a gateway device, and send the TSL and the extended service information of the sub-devices to the gateway.

Procedure

1. **In the left-side navigation pane, click Devices > Device .**
2. **On the Devices page, find the gateway device for which you want to add sub-devices and click View corresponding to it. You are directed to the Device Details page.**

3. Click Sub-device Management > Add Sub-device.



4. Enter the information of the sub-device in the dialog box.

Parameter	Description
Product	Select the name of the product for which the sub-device belongs.
Device	Select the name of the device that you want to add as a sub-device.

What's next

The topologiacal relationship between the gateway and the sub-device has been built. On the details page of the sub-device, you can view the gateway device information.

9.3 Connect sub-devices to IoT Platform

Sub-devices do not directly connect to IoT Platform. Instead, they connect to IoT Platform through gateway devices by using the communication channel between the gateway and IoT Platform.

Context

When you develop a gateway device, you must implement the following capabilities in the gateway: manage its topological relationship with sub-devices, connect sub-devices to IoT Platform, and enable the communication between sub-devices and IoT Platform.

You can use the device SDKs provided by Alibaba Cloud to develop gateways. For more information, see *Link Kit SDK*.

If you develop your own SDK for gateway devices, you must encapsulate the Alink protocol data of the sub-devices in the gateway device. For more information, see the chapter: Alink protocol data related to sub-devices.

Procedure

You can use a gateway device to connect a sub-device to IoT Platform by using the following process.

1. **Connect the gateway device to IoT Platform.**
2. **Connect the sub-device to the gateway device.**

   A sub-device does not directly connect to IoT Platform. Therefore, you do not need to install a device SDK of IoT Platform on the sub-device. The sub-device supplier develops the sub-device.

   The gateway device supplier provides the following capabilities to the gateway device: the gateway device detects the sub-device and obtain the device certificat e of the sub-device that is issued by IoT Platform, detects the sub-device going online and offline, and sends messages from IoT Platform to the sub-device. All these capabilities are achieved by the gateway device supplier or by using the protocol that is defined by the suppliers for the gateway device and sub-device.
3. **The gateway device checks whether the topological relationship with the sub-device is available.**

   If the topological relationship already exists between the gateway and the sub-device, skip step 5.
4. **Optional step. The gateway device reports the ProductKey and DeviceName of the sub-device to IoT Platform to register the sub-device.**

   This step is only applicable when the gateway device has not obtain the DeviceSecret of the sub-device and you have enabled dynamic registration for the sub-device in the IoT Platform console.
5. **Optional step. Add the topological relationship between the gateway and the sub-device.**

   If no topological relationship exists between the gateway and the sub-device, create a topological relationship.
6. **The gateway device sends a connection request to IoT Platform on behalf of the sub-device.**

For more information about sub-device management, see *Link Kit SDK*.

Example

For more information about how to connect a gateway device to IoT Platform, see *Connect a sub-device to IoT Platform*.

10 Develop devices based on Alink Protocol

10.1 Communications over Alink protocol

IoT Platform provides device SDKs for you to configure devices. These device SDKs already encapsulate protocols for data exchange between devices and IoT Platform. You can use these SDKs to develop your devices. If these SDKs do not meet your business requirements, you can develop your own SDK with an Alink communication channel by yourself.

For SDKs provided by IoT Platform, see *Device SDKs*.

The Alink protocol is a data exchange standard for IoT development that allows communication between devices and IoT Platform. The protocol exchanges data that is formatted in Alink JSON.

The following sections describe the device connection procedures and data communication processes (upstream and downstream) when using the Alink protocol.

Connect devices to IoT Platform

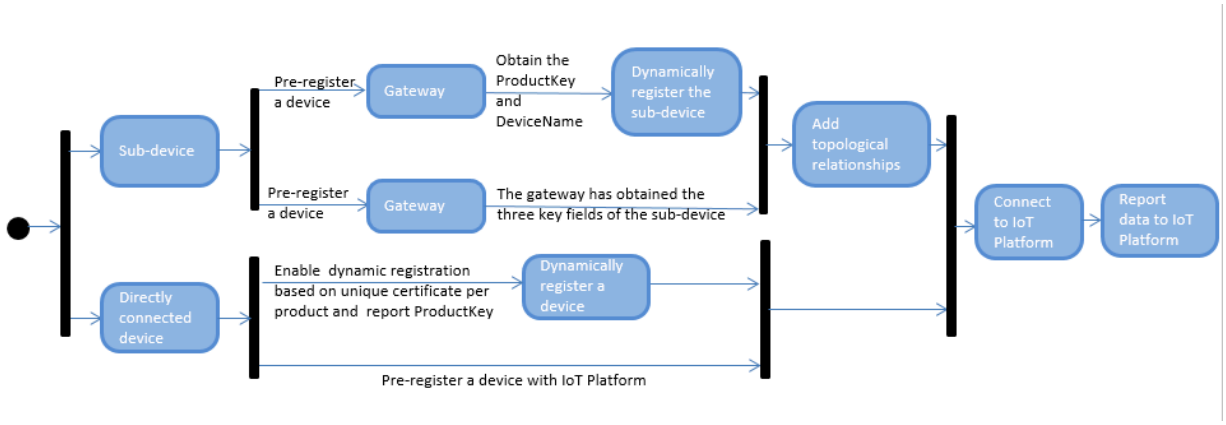
As shown in the following figure, devices can be connected to IoT Platform as directly connected devices or sub-devices. The connection process involves the following key steps: authenticate the device, establish a connection, and the device reports data to IoT Platform.

Directly connected devices can be connected to IoT Platform by using the following methods:

- **If** *Unique-certificate-per-device authentication* **is enabled, install the device certificate (ProductKey, DeviceName, and DeviceSecret) to the physical device for authentication, connect the device to IoT Platform, and then report data to IoT Platform.**
- **If dynamic registration based on** *Unique-certificate-per-product authentication* **is enabled, install the product certificate (ProductKey and ProductSecret) to the physical device for authentication, connect the device to IoT Platform, and then report data to IoT Platform.**

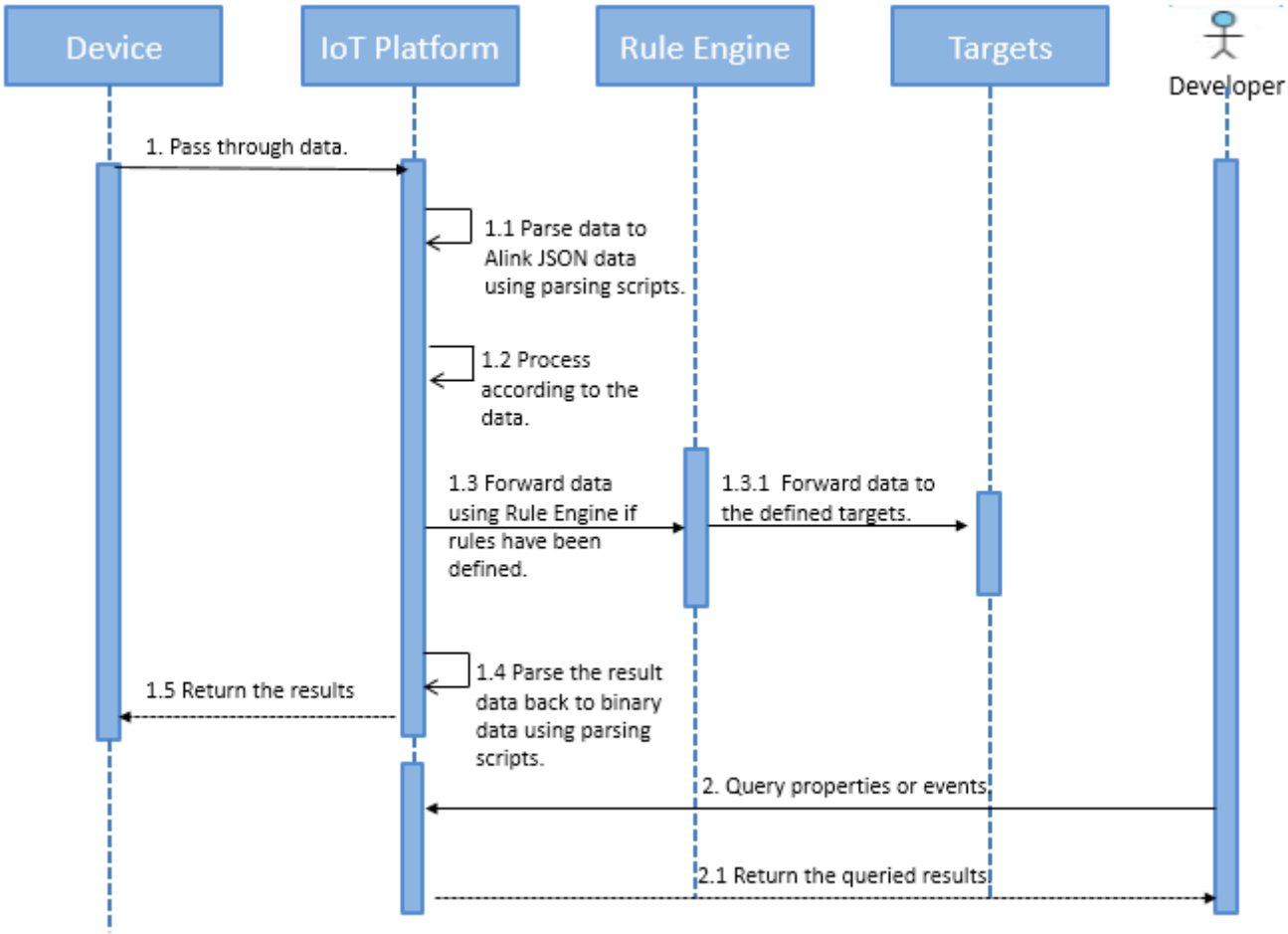
Sub-devices connect to IoT Platform through their gateways. Sub-devices can be connected to IoT Platform by using the following methods:

- **If *Unique-certificate-per-device authentication* is enabled, install the ProductKey, DeviceName, and DeviceSecret to the physical sub-device for authentication. The sub-device then sends its certificate information to the gateway, and then the gateway builds the topological relationship.The sub-device data are sent to IoT Platform through the gateway communication channel.**
- **If dynamic registration is enabled, install the ProductKey to the physical sub-device for authentication in advance. The sub-device sends the ProductKey and DeviceName to the gateway, and then the gateway forwards the ProductKey and DeviceName to IoT Platform. IoT Platform then verifies the received DeviceName and sends the DeviceSecret to the sub-device. The sub-device sends its certificate (ProductKey, DeviceName, and DeviceSecret) to the gateway for building topological relationship. The sub-device data are sent to IoT Platform through the gateway communication channel.**



Devices report properties or events

- **Pass-through (Do not parse/Custom) data**



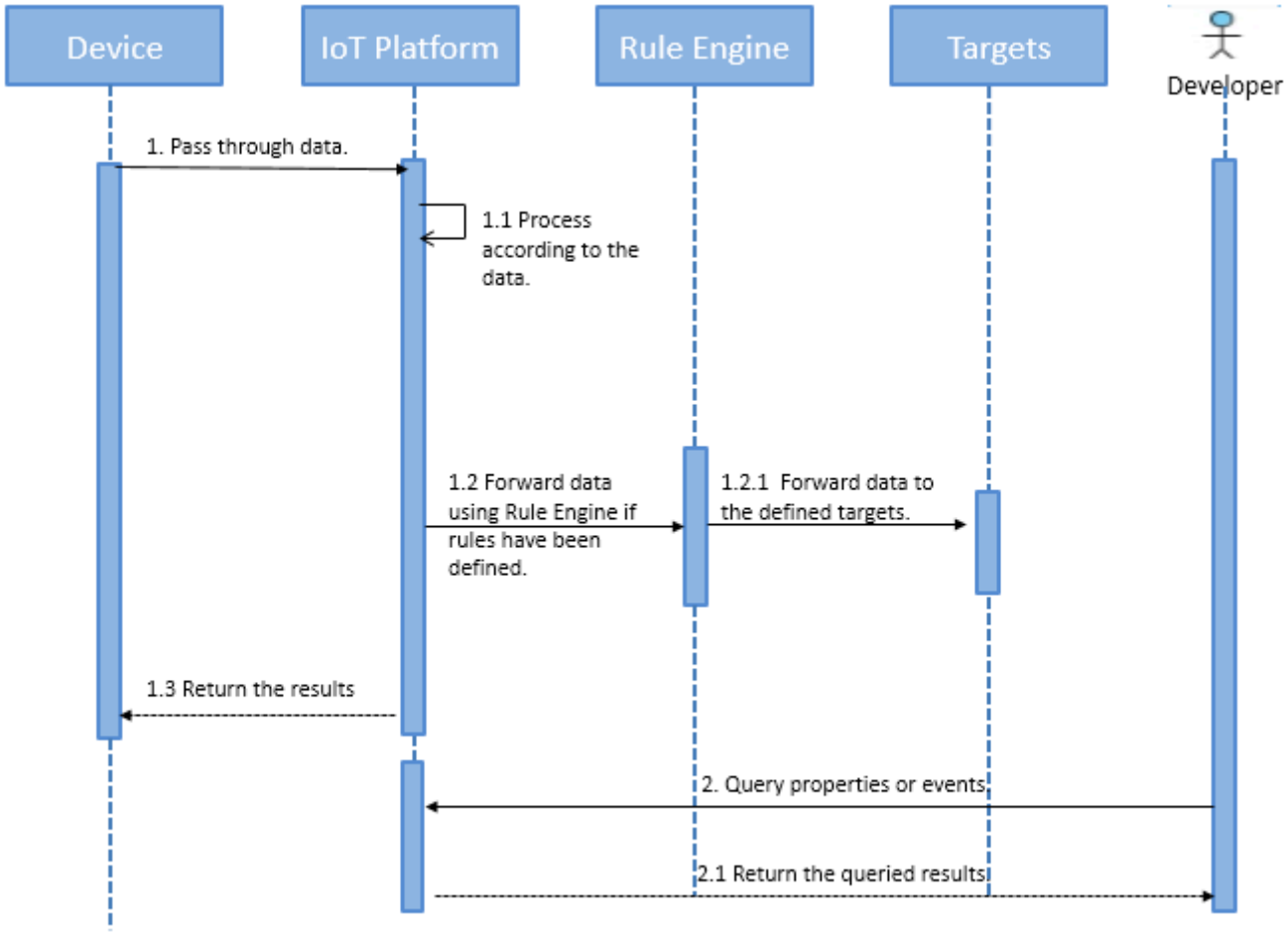
1. **The device reports raw data to IoT Platform using the topic for passing through data.**
2. **IoT Platform parses the received data using the data parsing script that you have submitted in the IoT Platform console. The `rawDataToProtocol` method in the script is called to convert the raw data reported by the device to Alink JSON data.**
3. **IoT Platform uses the Alink JSON data for further processes.**

> **Note:**
> **If you have configured rules for data forwarding, the Alink JSON data will be forwarded to the targets according to the rules.**
>
> - **The data forwarded by the rules engine are the data that have been parsed by the data parsing script.**

- **When you configure SQL statements for rules, to obtain the device properties, specify the data topic to be `/sys/{productKey}/{deviceName}/thing/event/property/post`.**
- **When you configure SQL statements for rules, to obtain the device events, specify the data topic to be `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`.**

4. **IoT Platform calls the `protocolToRawData` method in the data parsing script to convert the result data to the data format of the device.**
5. **IoT Platform pushes the converted data to the device.**
6. **You can query the device property data using the API *#unique_45* and query the device event data using the API *#unique_46*.**

- **Non-pass through (Alink JSON) data**



1. **The device reports Alink JSON data to IoT Platform using the topic for non-pass through data.**
2. **IoT Platform handles the received data.**

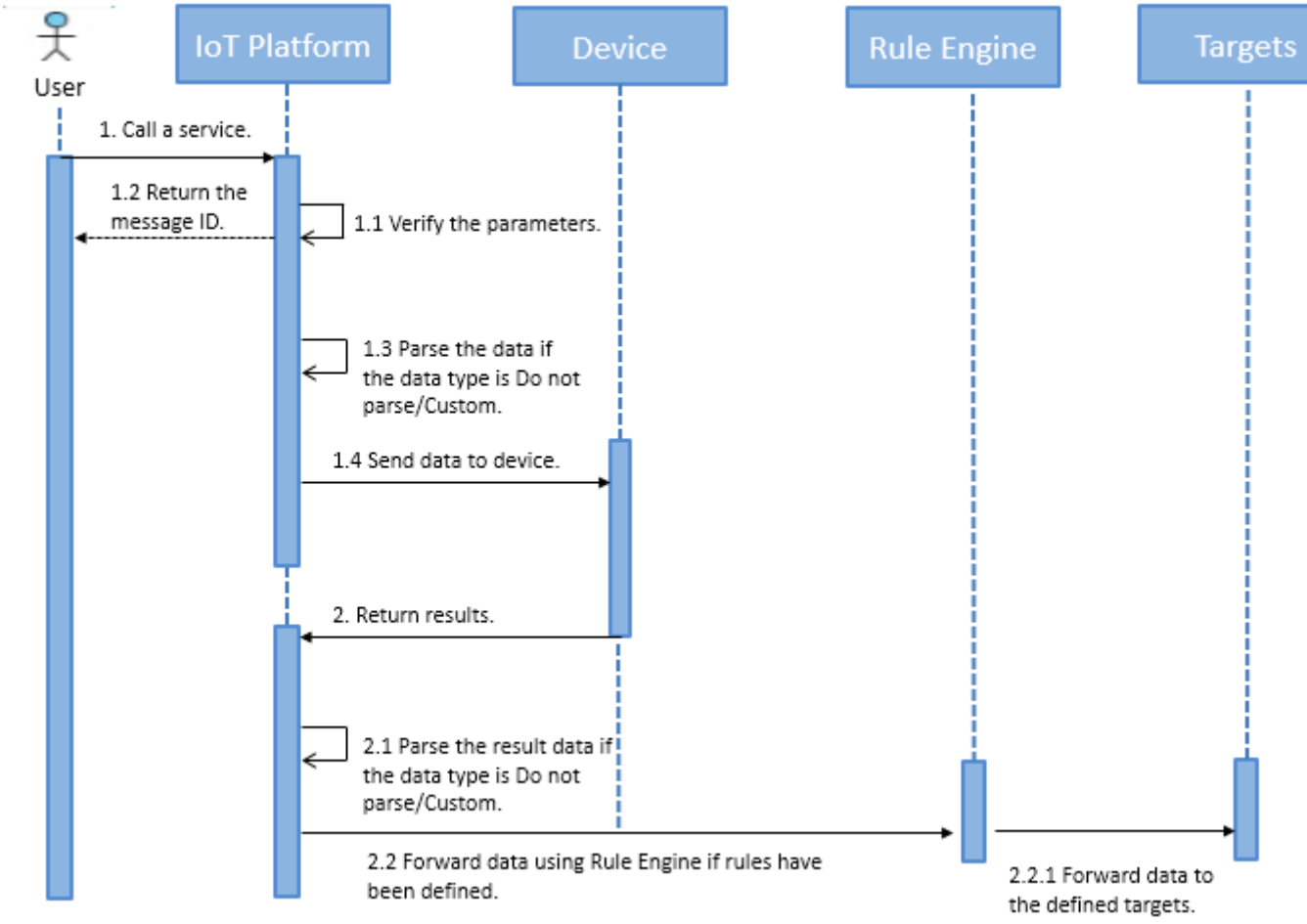
> **Note:**
> **If you have configured rules for data forwarding, the data will be forwarded to the targets according to the rules.**
>
> - **When you configure SQL statements for rules, to obtain the device properties, specify the data topic to be `/sys/{productKey}/{deviceName}/thing/event/property/post`.**

- **When you configure SQL statements for rules, to obtain the device events, specify the data topic to be `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`.**

3. **IoT Platform returns the results to the device.**
4. **You can query the device property data using the API *#unique_45* and query the device event data using the API *#unique_46*.**

Call device services or set device properties

- **Call device services or set device properties asynchronously**



1. **Set a device property or call a device service using the asynchronous method.**

Note:

- **Call the API *#unique_47* to set a property asynchronously.**

- Call the API *#unique_48* to call a service asynchronously (if you select Asynchronous as the method when you define the service, this service is called in the asynchronous method).

2. IoT Platform verifies the parameters.
3. IoT Platform uses the asynchronous method to handle the request and return the results. If the call is successful, the message ID is included in the response.

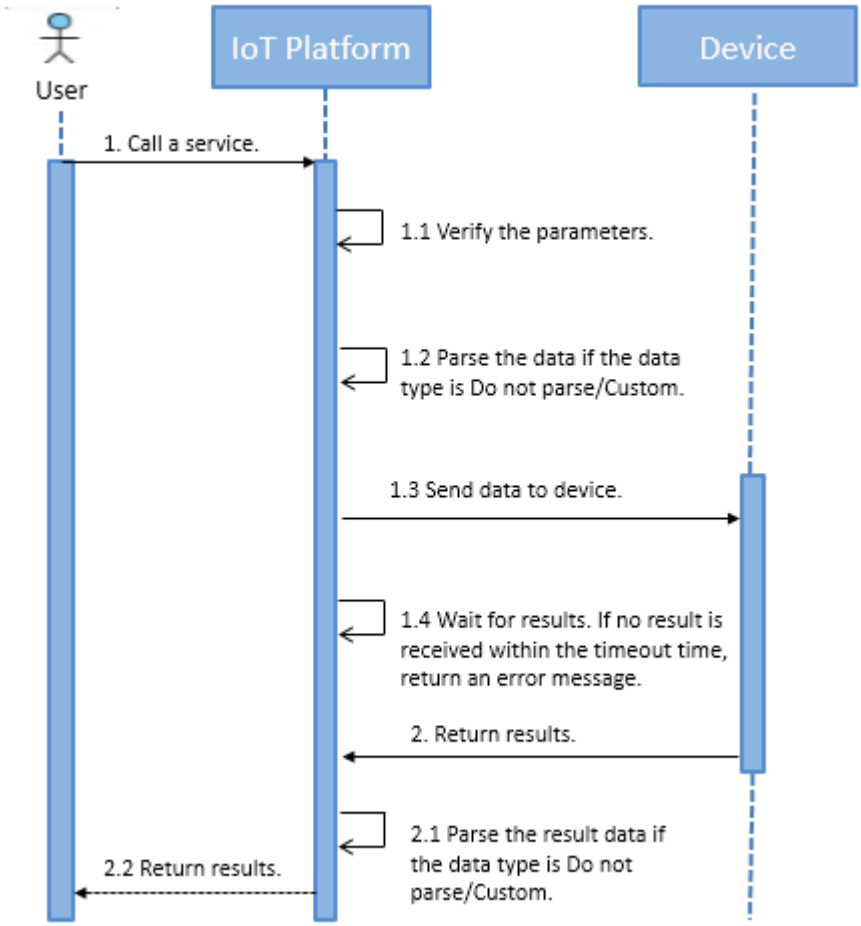
> **Note:**
> If the data type is pass-through (Do not parse/Custom), IoT Platform will call the `protocolToRawData` method in the data parsing script to convert the data before sending the data to the device.

4. IoT Platform sends the data to the device, and then the device handles the request asynchronously.

> **Note:**
> - If the data is pass-through (Do not parse/Custom) data, the topic for pass-through data is used.
> - If the data is non-pass through (Alink JSON) data, the topic for non-pass through data is used.

5. After the device has completed the requested operation, it returns the results to IoT Platform.
6. IoT Platform receives the results, and
   - If the data type is pass-through (Do not parse/Custom), IoT Platform will call the `rawDataToProtocol` method in the data parsing script to convert the data returned by the device.
   - If you have configured rules for data forwarding, IoT Platform rules engine will forward the data to the targets according to the rules.
     - When you configure SQL statements for rules, to obtain the results of service processing, specify the data topic as `/sys/{productKey}/{deviceName}/thing/downlink/reply/message`.
     - If the data type is pass-through (Do not parse/Custom), the data forwarded by the rules engine is the data that has been parsed by the data parsing script.

- **Call services using the synchronous method.**



1. **Call the API *#unique_48* to call a service synchronously (if you select Synchronous as the method when you define the service, this service is called in the synchronous method).**
2. **IoT Platform verifies the parameters.**
3. **The synchronous call method is where IoT Platform calls the RRPC topic to send the request data to the device, and waits for the device to return a result.**

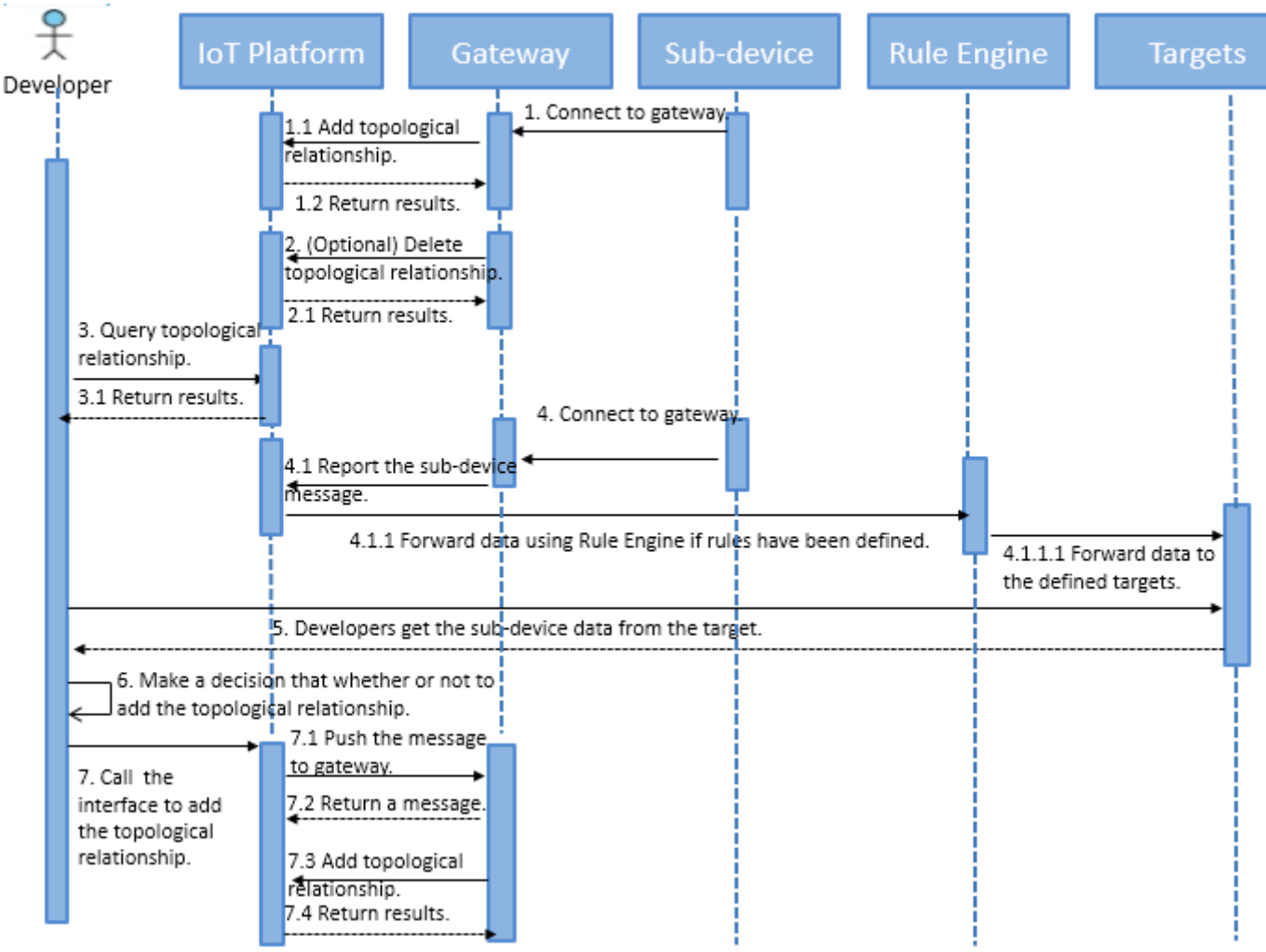
Note:

If the data type of the device is Do not parse/Custom, IoT Platform will call the `protocolToRawData` method in the data parsing script to convert the data before sending the data to the device.

4. **After the device has completed the requested operation, it returns the results to IoT Platform. If IoT Platform does not receive a result within the timeout period, it will send a timeout error to you.**
5. **IoT Platform returns the results to you.**

> **Note:**
> **If the data type of the device is Do not parse/Custom, IoT Platform will call the `rawDataToProtocol` method in the data parsing script to convert the data returned by the device, and then will send the results to you.**

Build topological relationships between gateways and sub-devices.



1. **After a sub-device has been connected to a gateway, the gateway sends a message using the topic for adding topological relationship messages to notify IoT**

Platform to build topological relationship between the gateway and the sub-device. IoT Platform handles the request and then returns a result.

2. Also, a gateway can send a message using the topic for deleting topological relationship messages to notify IoT Platform to remove a sub-device from the gateway.
3. Call the API *#unique_49* to query topological relationships of devices.
4. If you use the rules engine to forward device messages to another Alibaba Cloud service, and you receive device messages from that service, the process of building a topological relationship is as the following.
   a. The gateway device reports the information of the sub-device that has been detected to IoT Platform.
   b. IoT Platform receives the message and then forwards the message to the data forwarding target that you have specified when you were configuring the rule.
   c. You obtain the sub-device information from the data forwarding target service and then determine whether or not to build the topological relationship. Call the API *#unique_50* to send a request for building topological relationship to IoT Platform.
   d. IoT Platform receives the request from *#unique_50*, and then pushes the request to the gateway.
   e. The gateway receives the request and builds the topological relationship with the sub-device.

Note:

- Gateways use the topic `/sys/{productKey}/{deviceName}/thing/topo/add` to build topological relationships with sub-devices.
- Gateways use the topic `/sys/{productKey}/{deviceName}/thing/topo/delete` to delete topological relationships with sub-devices.
- Gateways use the topic `/sys/{productKey}/{deviceName}/thing/topo/get` to query the topological relationships with sub-devices.
- Gateways use the topic `/sys/{productKey}/{deviceName}/thing/list/found` to report information of sub-devices.
- Gateways use the topic `/sys/{productKey}/{deviceName}/thing/topo/add/notify` to initiate requests for building topological relationships.

10.2 Device identity registration

Before you connect a device to IoT Platform, you need to register the device identity to identify it on IoT Platform.

The following methods are available for identity registration:

- **Unique certificate per device: Obtain the ProductKey, DeviceName, and DeviceSecret of a device on IoT Platform and use them as the unique identifier . Install these three key fields on the firmware of the device. After the device is connected to IoT Platform, the device starts to communicate with IoT Platform.**
- **Dynamic registration: You can perform dynamic registration based on unique-certificate-per-product authentication for directly connected devices and perform dynamic registration for sub-devices.**
  - **To dynamically register a directly connected device based on unique-certificate-per-product authentication, follow these steps:**
    1. **In the IoT Platform console, pre-register the device and obtain the ProductKey and ProductSecret. When you pre-register the device, use device information that can be directly read from the device as the DeviceName, such as the MAC address or the serial number of the device.**
    2. **Enable dynamic registration in the console.**
    3. **Install the product certificate on the device firmware.**
    4. **The device authenticates to IoT Platform. If the device passes authentica tion, IoT Platform assigns a DeviceSecret to the device.**
    5. **The device uses the ProductKey, DeviceName, and DeviceSecret to establish a connection to IoT Platform.**
  - **To dynamically register a sub-device, follow these steps:**
    1. **In the IoT Platform console, pre-register a sub-device and obtain the ProductKey. When you pre-register the sub-device, use device information that can be read directly from the sub-device as the DeviceName, such as the MAC address and SN.**
    2. **Enable dynamic registration in the console.**
    3. **Install the ProductKey on the firmware of the sub-device or on the gateway.**
    4. **The gateway authenticates to IoT Platform on behalf of the sub-device.**

Dynamically register a sub-device

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/sub/register`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/sub/register_reply`

Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "iotId": "12344",
      "productKey": "1234556554",
      "deviceName": "deviceName1234",
      "deviceSecret": "xxxxxx"
    }
  ]
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	List	Parameters used for dynamic registrati on.
deviceName	String	Name of the sub-device.
productKey	String	ID of the product to which the sub-device belongs.

Parameter	Type	Description
iotId	String	Unique identifier of the sub-device.
deviceSecret	String	DeviceSecret key.
code	Integer	Result code.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6402	topo relation cannot add by self	A device cannot be added to itself as a sub-device.
401	request auth error	Signature verification has failed.

Dynamically register a directly connected device based on unique-certificate-per-product authentication

Directly connected devices send HTTP requests to perform dynamic register. Make sure that you have enabled dynamic registration based on unique certificate per product in the console.

- **URL template:** `https://iot-auth.cn-shanghai.aliyuncs.com/auth/register/device`
- **HTTP method： POST**

Request message

```
POST /auth/register/device  HTTP/1.1
Host: iot-auth.cn-shanghai.aliyuncs.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 123
productKey=1234556554&deviceName=deviceName1234&random=567345&sign=
adfv123hdfdh&signMethod=HmacMD5
```

Response message

```
{
  "code": 200,
  "data": {
    "productKey": "1234556554",
    "deviceName": "deviceName1234",
    "deviceSecret": "adsfweafdsf"
  },
  "message": "success"
```

```
}
```

Parameter description

Parameter	Type	Description
productKey	String	ID of the product to which the device belongs.
deviceName	String	Name of the device
random	String	Random number.
sign	String	Signature.
signMethod	String	Signing method. The supported methods are hmacmd5, hmacsha1, and hmacsha256.
code	Integer	Result code.
deviceSecret	String	DeviceSecret key.

Sign the parameters

All parameters reported to IoT Platform will be signed except `sign` and `signMethod`. Sort the signing parameters in alphabetical order, and splice the parameters and values without any splicing symbols.

Then, sign the parameters by using the algorithm specified by `signMethod`.

Example:

```
sign = hmac_sha1(productSecret, deviceNamedeviceName1234produc
tKey1234556554random123)
```

10.3 Add a topological relationship

After a sub-device has registered with IoT Platform, the gateway reports the topological relationship of gateways and sub-devices to IoT Platform before the sub-device connects to IoT Platform.

IoT Platform verifies the identity and the topological relationship during connection. If the verification is successful, IoT Platform establishes a logical connection with the sub-device and associates the logical connection with the physical connection of the gateway. The sub-device uses the same protocols as a directly connected device for data upload and download. Gateway information is not required to be included in the protocols.

After you delete the topological relationship of the sub-device from IoT Platform , the sub-device can no longer connect to IoT Platform through the gateway. IoT Platform will fail the authentication because the topological relationship does not exist.

Add topological relationships of sub-devices

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/topo/add`
- **Reply topic:** `sys/{productKey}/{deviceName}/thing/topo/add_reply`

Request data format when using the Alink protocol

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554",
      "sign": "xxxxxx",
      "signmethod": "hmacSha1",
      "timestamp": "1524448722000",
      "clientId": "xxxxxx"
    }
  ]
}
```

Response data format when using the Alink protocol

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	List	Input parameters of the request.
deviceName	String	Device name. The value is the name of the sub-device.

Parameter	Type	Description
productKey	String	Product ID. The value is the ID of the product to which the sub-device belongs .
sign	String	Signature. Signature algorithm: Sort all the parameters (except for `sign` and `signMethod`) that will be submitted to the server in lexicographical order, and then connect the parameters and values in turn (no connect symbols). Sign the signing parameters by using the algorithm specified by the signing method. For example, in the following request, sort the parameters in `params` in alphabetic order and then sign the parameters. `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp1524448722000)`
signmethod	String	Signing method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.
clientId	String	Identifier of a sub-device. This parameter is optional and may have the same value as ProductKey or DeviceName.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6402	topo relation cannot add by self	A device cannot be added to itself as a sub-device.
401	request auth error	Signature verification has failed .

Delete topological relationships of sub-devices

A gateway can publish a message to this topic to request IoT Platform to delete the topological relationship between the gateway and a sub-device.

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/topo/delete`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/topo/delete_reply`

Request data format when using the Alink protocol

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

Response data format when using the Alink protocol

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.

Parameter	Type	Description
version	String	Protocol version. Currently, the value can only be 1.0.
params	List	Request parameters.
deviceName	String	Device name. The value is the name of the sub-device.
productKey	String	Product ID. The value is the ID of the product to which the sub-device belongs.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6100	device not found	The device does not exist.

Obtain topological relationships of sub-devices

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/topo/get`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/topo/get_reply`

A gateway can publish a message to this topic to obtain the topological relationsh ips between the gateway and its connected sub-devices.

Request data format when using the Alink protocol

```
{
  "id": "123",
  "version": "1.0",
  "params": {}
}
```

Response data format when using the Alink protocol

```
{
  "id": "123",
  "code": 200,
  "data": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
```

```
    }
  ]
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters. This can be left empty.
deviceName	String	Name of the sub-device.
productKey	String	Product ID of the sub-device.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.

Report new sub-devices

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/list/found`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/list/found_reply`

In some scenarios, the gateway can discover new sub-devices. The gateway reports information of a new sub-device to IoT Platform. IoT Platform forwards the sub-device information to third-party applications, and the third-party applications choose the sub-devices to connect to the gateway.

Request data format when using the Alink protocol

```
{
  "id": "123",
  "version": "1.0",
  "params": [
```

```
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ]
}
```

Response data format when using the Alink protocol

```
{
  "id": "123",
  "code": 200,
  "data":{}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters. This parameter can be left empty.
deviceName	String	Name of the sub-device.
productKey	String	Product ID of the sub-device.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6250	product not found	The specified product to which the sub-device belongs does not exist.

Error code	Error message	Description
6280	devicename not meet specs	The name of the sub-device is invalid. The device name must be 4 to 32 characters in length and can contain letters, digits, hyphens (-), underscores (_), at signs (@), periods (.), and colons (:).

Notify the gateway to add topological relationships of the connected sub-devices

Downstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/topo/add/notify`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/topo/add/notify_reply`

IoT Platform publishes a message to this topic to notify a gateway to add topological relationships of the connected sub-devices. You can use this topic together with the topic that reports new sub-devices to IoT Platform. IoT Platform can subscribe to a data exchange topic to receive the response from the gateway. The data exchange topic is `/{productKey}/{deviceName}/thing/downlink/reply/message`.

Request data format when using the Alink protocol

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "deviceName": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.add.notify"
}
```

Response data format when using the Alink protocol

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters. This parameter can be left empty.
method	String	Request method. The value is `thing.topo.add.notify`.
deviceName	String	Name of the sub-device.
productKey	String	Product ID of the sub-device.
code	Integer	Result code. A value of 200 indicates the request is successful.

Notify the gateway about topological relationship change

When you add, delete, disable, or enable sub-devices on IoT Platform, IoT Platform will change the topological relationships accordingly and will send notifications to the gateway.

The gateway subscribes to the topic: `/sys/{productKey}/{deviceName}/thing/topo/change` for topological relationship change notifications.

Message format when using the Alink protocol

```
{
    "id":"123",
    "version":"1.0",
    "params":{
        "status":0,  //0:create  1:delete 2-enable  8-disable
        "subList":[{
            "productKey":"a1hRrzD****",
            "deviceName":"abcd"
        }]
    },
  "method":"thing.topo.change"
}
```

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value can only be 1.0.

Parameter	Type	Description
method	String	Request method. The value is `thing.topo.change`.
params	Object	Message content parameters, including `status` and `sublist`.
status	Integer	The operation result of topological relationship change. · 0: Create · 1: Delete · 2: Disable · 8: Enable
deviceName	String	Name of the sub-device.
productKey	String	Product ID of the sub-device.

Response data format when using the Alink protocol

```
{
    "id":"123",
    "code":200,
    "message":"success",
    "data":{}
}
```

10.4 Connect and disconnect sub-devices

Register devices with IoT Platform, assign the devices to a gateway device as sub-devices, and then connect these sub-devices to IoT Platform using the communication channel of the gateway device. When a sub-device is connecting to IoT Platform, IoT Platform verifies the identity of the sub-device according to the topological relationship between the gateway and the sub-device to identify whether the sub-device can use the channel of the gateway.

Note:
For messages about sub-device connection and disconnection, the QoS is 0.

Connect a sub-device to IoT Platform

Note:

A gateway device can have up to 1500 sub-devices connected to IoT Platform. When the maximum number is reached, IoT Platform will deny new connection requests from sub-devices of the gateway.

Upstream

- **Request topic:** `/ext/session/${productKey}/${deviceName}/combine/login`
- **Response topic:** `/ext/session/${productKey}/${deviceName}/combine/login_reply`

Note:

Because sub-devices use channels of gateways to communicate with IoT Platform, these topics are topics of gateway devices. Replace the variables *${productKey}* and *${deviceName}* in the topics with the corresponding information of the gateway device.

Request message

```
{
  "id": "123",
  "params": {
    "productKey": "123",
    "deviceName": "test",
    "clientId": "123",
    "timestamp": "123",
    "signMethod": "hmacmd5",
    "sign": "xxxxxx",
    "cleanSession": "true"
  }
}
```

Note:

In the request message, the values of parameters `productKey` and `deviceName` are the corresponding information of the sub-device.

Response message:

```
{
  "id":"123",
  "code":200,
  "message":"success"
  "data":""
}
```

Request Parameters

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	The unique identifier of the product to which the device belongs.
sign	String	Signature of the sub-device. Sub-devices use the same signature rules as gateways. Sign algorithm: 1. Sort all the parameters (except `sign` and `signMethod` and `cleanSession`) to be submitted to the server in alphabetical order, and then concatenate the parameters and values in turn (without any delimiters). 2. Then, sign the parameters by using the algorithm specified by signMethod and the DeviceSecret of the sub-device. Example: `sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp123 )`
signMethod	String	Sign method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.
clientId	String	The device identifier. The value of this parameter can be the value of ProductKey and DeviceName.
cleanSession	String	· A value of true indicates that when the sub-device is offline, messages sent based on QoS= 1 method will be cleared. · A value of false indicates that when the sub-device is offline, messages sent based on QoS= 1 method will not be cleared.

Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	Result code. A value of 200 indicates that the request is successful.
message	String	Result message.
data	Object	Additional information in the response, in JSON format.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
429	rate limit, too many subDeviceOnline msg in one minute	The authentication requests from the device are limited because the device requested authentication to IoT Platform too frequently.
428	too many subdevices under gateway	The number of sub-devices connected to IoT Platform has reached the upper limit.
6401	topo relation not exist	The topological relationship between the gateway and the sub-device does not exist.
6100	device not found	The sub-device does not exist.
521	device deleted	The sub-device has been deleted.
522	device forbidden	The sub-device has been disabled.
6287	invalid sign	The password or signature of the sub-device is incorrect.

Disconnect a sub-device from IoT Platform

Upstream

- **Request topic:** `/ext/session/{productKey}/{deviceName}/combine/logout`
- **Response topic:** `/ext/session/{productKey}/{deviceName}/combine/logout_reply`

Note:
Because sub-devices use channels of gateways to communicate with IoT Platform, these topics are topics of gateway devices. Replace the variables `${productKey}` and `${deviceName}` in the topics with the corresponding information of the gateway device.

Request message:

```
{
  "id": 123,
  "params": {
    "productKey": "xxxxx",
    "deviceName": "xxxxx"
  }
}
```

Note:
In the request message, the values of parameters `productKey` and `deviceName` are the corresponding information of the sub-device.

Response message:

```
{
  "id": "123",
  "code": 200,
  "message": "success",
  "data": ""
}
```

Request Parameters

Parameter	Type	Description
id	String	The message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	The unique identifier of the product to which the device belongs.

Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	Result code. A value of 200 indicates that the request is successful.
message	String	Result message.
data	Object	Additional information in the response, in JSON format.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
520	device no session	The sub-device session does not exist.

For more information about sub-device connections, see *Device identity registration*. For more information about error codes, see *Error codes for device SDKs*.

10.5 Device properties, events, and services

If you have defined the *TSL* for a product, the devices of this product can separately report data regarding the properties, events, and services that you have defined. For information about the data format of TSL, see *Data format* . This topic describes how data is reported based on the TSL.

When you create a product, you must select a data type for devices of the product. IoT Platform supports two data types: ICA Standard Data Format (Alink JSON) and Do not parse/Custom. We recommend that you select Alink JSON, because it is the standard data format of IoT Platform.

- ICA Standard Data Format (Alink JSON): Devices generate data in the standard format defined by IoT Platform, and then report the data to IoT Platform. The following sections provide examples of Alink JSON data format.
- Do not parse/Custom: Devices report raw data, such as binary data, to IoT Platform, and then IoT Platform parses the raw data to be standard data using the *parsing script* that you have submitted in the console. Data generated by IoT

Platform is in Alink JSON format, and before sending the data to devices, IoT Platform will parse the data to the format that the devices support.

Devices report properties

Report data (Do not parse/Custom)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

The raw data of a request message:

Note:
In raw data, the request `methodthing.event.property.post` **must be included.**

```
0x020000007b00
```

Response message from IoT Platform:

```
{
    "id":"123",
    "code":200,
    "method":"thing.event.property.post"
    "data":{}
}
```

Report Data (Alink JSON)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/event/property/post`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/event/property/post_reply`

Request message:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": {
      "value": "on",
      "time": 1524448722000
```

```
    },
    "WF": {
      "value": 23.6,
      "time": 1524448722000
    }
  }
}
```

Table 10-1: Request Parameters

Parameter	Type	Description
id	String	The message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	The protocol version. Currently, the value is 1.0.
params	Object	The request parameters. In the preceding request example, the device reports two properties: Power and WF. Property information includes time (the time when the property is reported) and value (the value of the property).
time	Long	The time when the property is reported.
value	Object	The value of the property.

Response message:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Table 10-2: Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	The result code. See *Common codes on devices*.
data	String	The data that is returned when the request is successful.

Table 10-3: Error codes

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6106	map size must less than 200	The number of reported properties exceeds the maximum limit. Up to 200 properties can be reported at a time.
6313	tsl service not available	The TSL verification service is not available. IoT Platform verifies all the received properties according to the TSLs of products. If the TSL verification service is available, but some reported properties do not match with any properties defined in the TSL, IoT Platform ignores the invalid properties. If all the reported properties do not match with any properties defined in the TSL, IoT Platform ignores them all. In this case, the response will still indicate that the verification is successful. This error is reported when a system exception occurs.

You can use the *Rules engine* to forward property information reported by devices to other supported Alibaba Cloud services. For more information about topics and data formats, see *Messages about device properties reported by devices*.

Set device properties

Push data to devices (Do not parse/Custom)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/model/down_raw`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/model/down_raw_reply`

Push data to devices (Alink JSON)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/service/property/set`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/service/property/set_reply`

Request message:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "temperature": "30.5"
  },
  "method": "thing.service.property.set"
}
```

Table 10-4: Request Parameters

Parameter	Type	Description
id	String	The message ID. IoT Platform generates IDs for downstream messages.
version	String	The protocol version. Currently, the value is 1.0.
params	Object	The property parameters. In the preceding request example, the property to be set is `{ "temperature": "30.5" }`
method	String	Request method. The value is `thing.service.property.set`.

Response message:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Table 10-5: Response parameters

Parameter	Type	Description
id	String	The message ID.

Parameter	Type	Description
code	Integer	The result code. See *Common codes on devices*.
data	String	The data that is returned when the request is successful.

You can use the *Rules engine* to forward the property setting results from devices to other supported Alibaba Cloud services. For message topics and data formats, see *Devices return result data to the cloud*.

Devices report events

Report data (Do not parse/Custom)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

The raw data of a request message:

> **Note:**
> **In raw data, the request `methodthing.event.{tsl.event.identifier}.post` must be included. `tsl.event.identifier` indicates the event identifier in the TSL.**

```
0xff0000007b00
```

Response message from IoT Platform:

```
{
    "id":"123",
    "code":200,
    "method":"thing.event.{tsl.event.identifier}.post"
    "data":{}
}
```

Report Data (Alink JSON)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post_reply`

Request message:

```
{
  "id": "123",
```

```
  "version": "1.0",
  "params": {
    "value": {
      "Power": "on",
      "WF": "2"
    },
    "time": 1524448722000
  }
}
```

Table 10-6: Request Parameters

Parameter	Type	Description
id	String	The message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
version	String	The protocol version. Currently, the value is 1.0.
params	List	The parameters of the reported events.
value	Object	The event information. In the preceding request example, the events are: `{ "Power": "on", "WF": "2" }`
time	Long	The UTC timestamp when the event occurs.

Response message:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Table 10-7: Response parameters

Parameter	Type	Description
id	String	The message ID.

Parameter	Type	Description
code	Integer	The result code. See *Common codes on devices*. **Note:** IoT Platform verifies all the events reported by devices according to the TSLs of products. If the reported event does not match with any events defined in the TSL, an error code is returned.
data	String	The data that is returned when the request is successful.

For example, an event alarm has been defined in the TSL of a product:

```
{
  "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.json
",
  "link": "/sys/${productKey}/airCondition/thing/",
  "profile": {
    "productKey": "al123456789",
    "deviceName": "airCondition"
  },
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",
      "desc": "Fan alarm",
      "type": "alert",
      "required": true,
      "outputData": [
        {
          "identifier": "errorCode",
          "name": "ErrorCode",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ],
      "method": "thing.event.alarm.post"
    }
  ]
}
```

Request message of reporting an event:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
```

```
      "errorCode": "error"
    },
    "time": 1524448722000
  }
}
```

You can use the *Rules engine* to forward event information reported by devices to other supported Alibaba Cloud services. For more information about topics and data formats, see *Messages about events reported by devices*

Call device services

Push data to devices (Do not parse/Custom)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/model/down_raw`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/model/down_raw_reply`

Push data to devices (Alink JSON)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}_reply`

Service calling methods

Supports synchronous calls and asynchronous calls. When you *define a service*, you are required to select a method for the service.

- **Synchronous method: IoT Platform uses the RRPC method to push requests to devices. For information about the RRPC method, see *What is RRPC*.**
- **Asynchronous method: IoT Platform pushes requests to devices in an asynchronous manner, and the devices return operation results in an asynchronous manner.**

  Only when asynchronous method is selected for a service does IoT Platform subscribe to the response topic. You can use the *Rules engine* to forward the results of asynchronous calls returned by devices to other supported Alibaba Cloud services. For more information about topics and data formats, see *#unique_59/unique_59_Connect_42_section_mgr_2tl_b2b*.

Request message:

```
{
```

```
  "id": "123",
  "version": "1.0",
  "params": {
    "Power": "on",
    "WF": "2"
  },
  "method": "thing.service.{tsl.service.identifier}"
}
```

Table 10-8: Request Parameters

Parameter	Type	Description
id	String	The message ID. IoT Platform generates IDs for downstream messages.
version	String	The protocol version. Currently, the value is 1.0.
params	Map	The parameters used to call a service, including the identifier and value of the service. Example: `{ "Power": "on", "WF": "2" }`
method	String	Request method. **Note:** `tsl.service.identifier` indicates the identifier of the service in TSL. For information about how to define a TSL, see *Overview*.

Response message:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Table 10-9: Response parameters

Parameter	Type	Description
id	String	The message ID.

Parameter	Type	Description
code	Integer	The result code. See *Common codes on devices*.
data	String	The data that is returned when the request is successful. The value of data is determined by the TSL of the product. If the device does not return any information about the service, the value of data is empty. If the device returns service information, the returned data value will strictly comply with the definition of the service in the TSL.

For example, the service SetWeight has been defined in the TSL of the product as follows:

```
{
  "schema": "https://iotx-tsl.oss-ap-southeast-1.aliyuncs.com/schema.
json",
  "profile": {
    "productKey": "testProduct01"
  },
  "services": [
    {
      "outputData": [
        {
          "identifier": "OldWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "max": "200",
              "step": "1"
            },
            "type": "double"
          },
          "name": "OldWeight"
        },
        {
          "identifier": "CollectTime",
          "dataType": {
            "specs": {
              "length": "2048"
            },
            "type": "text"
          },
          "name": "CollectTime"
        }
      ],
```

```
      "identifier": "SetWeight",
      "inputData": [
        {
          "identifier": "NewWeight",
          "dataType": {
            "specs": {
              "unit": "kg",
              "min": "0",
              "max": "200",
              "step": "1"
            },
            "type": "double"
          },
          "name": "NewWeight"
        }
      ],
      "method": "thing.service.SetWeight",
      "name": "SetWeight",
      "required": false,
      "callType": "async"
    }
  ]
}
```

Request message of a service call:

```
{
  "method": "thing.service.SetWeight",
  "id": "105917531",
  "params": {
    "NewWeight": 100.8
  },
  "version": "1.0.0"
}
```

Response message:

```
{
  "id": "105917531",
  "code": 200,
  "data": {
    "CollectTime": "1536228947682",
    "OldWeight": 100.101
  }
}
```

Gateway devices report data

A gateway device can report properties and events of itself and properties and events of its sub-devices to IoT Platform.

Note:

- **A gateway can report up to 200 properties and 20 events at one time.**
- **A gateway can report up to 20 properties and events of sub-devices.**

Report data (Do not parse/Custom)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/model/up_raw_reply`

The raw data of a request message:

> **Note:**
> **In raw data, the request** `methodthing.event.property.pack.post` **must be included.**

```
0xff0000007b00
```

Response message from IoT Platform:

```
{
  "id": "123",
  "code": 200,
  "method": "thing.event.property.pack.post",
  "data": {}
}
```

Report data (Alink JSON)

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post_reply`

Request message:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "properties": {
      "Power": {
        "value": "on",
        "time": 1524448722000
      },
      "WF": {
        "value": { },
        "time": 1524448722000
      }
    },
    "events": {
      "alarmEvent1": {
        "value": {
          "param1": "on",
          "param2": "2"
        },
        "time": 1524448722000
      },
      "alertEvent2": {
```

```
        "value": {
          "param1": "on",
          "param2": "2"
        },
        "time": 1524448722000
      }
    },
    "subDevices": [
      {
        "identity": {
          "productKey": "",
          "deviceName": ""
        },
        "properties": {
          "Power": {
            "value": "on",
            "time": 1524448722000
          },
          "WF": {
            "value": { },
            "time": 1524448722000
          }
        },
        "events": {
          "alarmEvent1": {
            "value": {
              "param1": "on",
              "param2": "2"
            },
            "time": 1524448722000
          },
          "alertEvent2": {
            "value": {
              "param1": "on",
              "param2": "2"
            },
            "time": 1524448722000
          }
        }
      }
    ]
  }
}
```

Table 10-10: Request Parameters

Parameter	Type	Description
id	**String**	**The message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.**
version	**String**	**The protocol version. Currently, the value is 1.0.**
params	**Object**	**The request parameters.**

Parameter	Type	Description
properties	Object	The information about a property, including property identifier, value and time when the property was generated.
events	Object	The information about an event, including event identifier, value and time when the event was generated.
subDevices	Object	The sub-device information.
productKey	String	The ProductKey of a sub-device.
deviceName	String	The name of a sub-device.

Response message:

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Table 10-11: Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	Result code. A value of 200 indicates that the request is successful. **Note:** IoT Platform then verifies the devices, topological relationships, and property and event definitions in the TSL. If any one of the verifications fails, the data report also fails.
data	Object	The data that is returned when the request is successful.

Devices report TSL historical data

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/event/property/pack/post_reply`

Request message:

```
{
  "id": 123,
  "version": "1.0",
  "method": "thing.event.property.history.post",
  "params": [
    {
      "identity": {
        "productKey": "",
        "deviceName": ""
      },
      "properties": [
        {
          "Power": {
            "value": "on",
            "time": 123456
          },
          "WF": {
            "value": "3",
            "time": 123456
          }
        },
        {
          "Power": {
            "value": "on",
            "time": 123456
          },
          "WF": {
            "value": "3",
            "time": 123456
          }
        }
      ],
      "events": [
        {
          "alarmEvent": {
            "value": {
              "Power": "on",
              "WF": "2"
            },
            "time": 123456
          },
          "alertEvent": {
            "value": {
              "Power": "off",
              "WF": "3"
            },
            "time": 123456
          }
        }
      ]
    },
    {
      "identity": {
        "productKey": "",
        "deviceName": ""
      },
      "properties": [
        {
          "Power": {
            "value": "on",
            "time": 123456
```

```
            },
            "WF": {
              "value": "3",
              "time": 123456
            }
          }
        ],
        "events": [
          {
            "alarmEvent": {
              "value": {
                "Power": "on",
                "WF": "2"
              },
              "time": 123456
            },
            "alertEvent": {
              "value": {
                "Power": "off",
                "WF": "3"
              },
              "time": 123456
            }
          }
        ]
      }
    ]
}
```

Response message from IoT Platform:

```
{
    "id":"123",
    "code":200,
    "data":{}
}
```

10.6 Desired device property values

After you set a desired property value for a device in IoT Platform, the property value is updated in real time if the device is online. If the device is offline, the desired value is cached in IoT Platform. When the device comes online again, it will obtain the desired value and update the property value. This topic describes the message formats related to desired property values.

Obtain desired property values

Upstream data in Alink JSON format

A device requests the desired property values from IoT Platform.

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/property/desired/get`

- **Response topic:** `/sys/{productKey}/{deviceName}/thing/property/desired/get_reply`

Request format

```
{
    "id" : "123",
    "version": "1.0",
    "params" : [
        "power",
        "temperature"
    ]
}
```

Response format

```
{
    "id":"123",
    "code": 200,
    "data":{
  "power": {
   "value": "on",
   "version": 2
  }
 }
}
```

Table 10-12: Request parameters

Parameter	Type	Description
id	**String**	**The message ID. Define the message ID to be a string of numbers, and be unique in the device.**
version	**String**	**The protocol version. Currently, the value can only be `1.0`.**

Parameter	Type	Description
params	List	The identifier list of properties of which you want to obtain the desired values. In this example, the following property identifiers are listed: `[ "power", "temperature" ]`

Table 10-13: Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	The result code. For more information, see the *common codes on the device*.
data	Object	The desired value information that is returned. In this example, the desired value information about property "power" is returned. The information includes the `value` and `version` of the property. `{ "power": { "value": "on", "version": 2 } }` **Note:** If no desired value is set for a property in IoT Platform or the desired value has been cleared, the returned data will not contain the identifier of this property. In this example, the property "temperature" does not have a desired value, therefore, the returned data does not contain this property identifier. For more information about the parameters in data, see the following table *Parameters in data*.

Table 10-14: Parameters in data

Parameter	Type	Description
key	String	The identifier of the property, such as "power" in this example.
value	Object	The desired value.
version	Integer	The current version of the desired value. **Note:** When you set the desired property value for the first time, this value is 0. After the first desired value is set, the version automatically changes to 1. Then, the version increases by 1 every time you set the desired value.

Clear desired property values

Upstream data in Alink JSON format

Requests to clear the desired property values that are cached in IoT Platform.

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/property/desired/delete`
- **Response topic:** `/sys/{productKey}/{deviceName}/thing/property/desired/delete_reply`

Request format

```
{
    "id" : "123",
    "version": "1.0",
    "params" : [{
  "power": {
   "version": 1
  },
  "temperature": {
  }
 }
}
```

Response format

```
{
    "id":"123",
    "code":200,
    "data":{
 }
```

```
}
```

Table 10-15: Request parameters

Parameter	Type	Description
id	String	The message ID. Define the message ID to be a string of numbers, and be unique in the device.
version	String	The protocol version. Currently, the value can only be `1.0`.
params	List	The list of the properties of which you want to clear the desired values. A property is identified by the identifier and `version`. For example: `{ "power": { "version": 1 }, "temperature": { } }` For more information about params, see the following table *Parameters in params*.

Table 10-16: Parameters in params

Parameter	Type	Description
key	String	The identifier of the property. In this example , the following property identifiers are listed: power and temperature.

Parameter	Type	Description
version	Integer	The current version of the desired value. **Note:** · You can obtain the value of the `version` parameter from topic `/sys/{productKey}/{deviceName}/thing/property/desired/get`. · If you set `version` to 2, IoT Platform clears the desired value only if the current version is 2. If the current version of the desired value is 3 in IoT Platform, this clear request will be ignored. · If you are not sure about the current version, do not specify this parameter in the request. When there is no `version` in the request, IoT Platform does not verify the version, but clears the desired value directly.

Table 10-17: Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	The result code. For more information, see the *common codes on the device*.
data	String	The returned data.

10.7 Disable and delete devices

Gateways can disable and delete their sub-devices.

Disable devices

Downstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/disable`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/disable_reply`

This topic disables a device connection. IoT Platform publishes messages to this topic asynchronously, and the devices subscribe to this topic. Gateways can subscribe to this topic to disable the corresponding sub-devices.

Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.disable"
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Request parameters. Leave empty.
method	String	Request method.
code	Integer	Results information. For more information, see*Common codes on devices*

Enable devices

Downstream

- **Request Topic:** `/sys/{productKey}/{deviceName}/thing/enable`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/enable_reply`

This topic enables a device connection. IoT Platform publishes messages to this topic asynchronously, and the devices subscribe to this topic. Gateways can subscribe to this topic to enable the corresponding sub-devices.

Request message

```
{
  "id": "123",
  "version": "1.0",
```

```
  "params": {},
  "method": "thing.enable"
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Request parameters. Leave empty.
method	String	Request method.
code	Integer	Result code. For more information, see the common codes.

Delete devices

Downstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/delete`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/delete_reply`

This topic deletes a device connection. IoT Platform publishes messages to this topic asynchronously, and the devices subscribe to this topic. Gateways can subscribe to this topic to delete the corresponding sub-devices.

Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": {},
  "method": "thing.delete"
```

```
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Request parameters. Leave empty.
method	String	Request method.
code	String	Result code. For more information, see the common codes.

10.8 Device tags

Some static extended device information, such as vendor model and device model, can be saved as device tags.

Report tags

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/deviceinfo/update`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/deviceinfo/update_reply`

Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature",
      "attrValue": "36.8"
```

```
    }
  ]
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters. This parameter can contain a maximum of 200 items.
attrKey	String	Tag name. · Length: Up to 100 bytes. · Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_). · The tag name must start with an English letter or underscore (_).
attrValue	String	Tag value.

Parameter	Type	Description
code	Integer	Result code. A value of 200 indicates the request is successful.

Error codes

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6100	device not found	The device does not exist.

Delete tags

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/deviceinfo/delete`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/deviceinfo/delete_reply`

Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": [
    {
      "attrKey": "Temperature"
    }
  ]
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters.
attrKey	String	Tag name. · Length: Up to 100 bytes. · Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_). · The tag name must start with an English letter or underscore (_).
attrValue	String	Tag value.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6100	device not found	The device does not exist.

10.9 TSL model

A device can publish requests to the request topic to obtain the *Device TSL model* from IoT Platform.

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/dsltemplate/get`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/dsltemplate/get_reply`

The Allink data format of a request

```
{
  "id": "123",
  "version": "1.0",
  "params": {}
}
```

The Allink data format of a response

```
{
  "id": "123",
  "code": 200,
  "data": {
    "schema": "https://iot-tsl.oss-cn-shanghai.aliyuncs.com/schema.
json",
    "link": "/sys/1234556554/airCondition/thing/",
    "profile": {
      "productKey": "1234556554",
      "deviceName": "airCondition"
    },
    "properties": [
      {
        "identifier": "fan_array_property",
        "name": "Fan array property",
        "accessMode": "r",
        "required": true,
        "dataType": {
          "type": "array",
          "specs": {
            "size": "128",
            "item": {
              "type": "int"
            }
          }
        }
      }
    ],
    "events": [
      {
        "identifier": "alarm",
        "name": "alarm",
        "desc": "Fan alert",
        "type": "alert",
        "required": true,
        "outputData": [
          {
            "identifier": "errorCode",
            "name": "Error code",
            "dataType": {
```

```
            "type": "text",
            "specs": {
              "length": "255"
            }
          }
        }
      ],
      "method": "thing.event.alarm.post"
    }
  ],
  "services": [
    {
      "identifier": "timeReset",
      "name": "timeReset",
      "desc": "Time calibration",
      "inputData": [
        {
          "identifier": "timeZone",
          "name": "Time zone",
          "dataType": {
            "type": "text",
            "specs": {
              "length": "512"
            }
          }
        }
      ],
      "outputData": [
        {
          "identifier": "curTime",
          "name": "Current time",
          "dataType": {
            "type": "date",
            "specs": {}
          }
        }
      ],
      "method": "thing.service.timeReset"
    },
    {
      "identifier": "set",
      "name": "set",
      "required": true,
      "desc": "Set properties",
      "method": "thing.service.property.set",
      "inputData": [
        {
          "identifier": "fan_int_property",
          "name": "Integer property of the fan",
          "accessMode": "rw",
          "required": true,
          "dataType": {
            "type": "int",
            "specs": {
              "min": "0",
              "max": "100",
              "unit": "g/ml",
              "unitName": "Millilitter"
            }
          }
        }
      ],
      "outputData": []
    },
```

```
      {
        "identifier": "get",
        "name": "get",
        "required": true,
        "desc": "Get properties",
        "method": "thing.service.property.get",
        "inputData": [
          "array_property",
          "fan_int_property",
          "batch_enum_attr_id",
          "fan_float_property",
           "fan_double_property",
          "fan_text_property",
          "Maid ",
          "batch_boolean_attr_id",
          "fan_struct_property"
        ],
        "outputData": [
          {
            "identifier": "fan_array_property",
            "name": "Fan array property",
            "accessMode": "r",
            "required": true,
            "dataType": {
              "type": "array",
              "specs": {
                "size": "128",
                "item": {
                  "type": "int"
                }
              }
            }
          }
        ]
      }
    ]
  }
}
```

Parameter descriptions:

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Leave this parameter empty.

Parameter	Type	Description
productKey	String	ProductKey. In the example, the ProductKey is 1234556554.
deviceName	String	Device name. In the example, the device name is airCondition.
data	Object	TSL model of the device. For more information, see*Overview*

Error codes

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6321	tsl: device not exist in product	The device does not exist.

10.10 Firmware update

For information about the firmware update, see *#unique_68* and *#unique_69*.

Report the firmware version

Upstream

- **Request topic:** `/ota/device/inform/{productKey}/{deviceName}`

  The device publishes a message to this topic to report the current firmware version to IoT Platform.

Request message

```
{
  "id": "123",
  "params": {
    "version": "1.0.1"
  }
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Version information of the firmware.

Push firmware information

Downstream

- **Request topic:** `/ota/device/upgrade/{productKey}/{deviceName}`

  IoT Platform publishes messages to this topic to push firmware information. The devices subscribe to this topic to obtain the firmware information.

Request message

```
{
  "code": "1000",
  "data": {
    "size": 432945,
    "version": "2.0.0",
    "url": "https://iotx-ota-pre.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e",
    "sign": "93230c3bde425a9d7984a594ac55ea1e",
    "signMethod": "Md5"
  },
  "id": "1507707025",
  "message": "success"
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
message	String	Result information.
version	String	Version information of the firmware.
size	Long	Firmware size in bytes.
url	String	OSS address of the firmware.
sign	String	Firmware signature.
signMethod	String	Signing method. Currently, the supported methods are MD5 and sha256.
md5	String	This parameter is reserved. This parameter is used to be compatible with old device information. When the signing method is MD5, IoT Platform will assign values to both the sign and md5 parameters.

Report update progress

Upstream

- **Request topic:** `/ota/device/progress/{productKey}/{deviceName}`

  A device subscribes to this topic to report the firmware update progress.

Request message

```
{
  "id": "1213",
  "params": {
    "step": "-1",
    "desc": "Firmware update has failed. No firmware information is
available."
  }
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
step	String	Firmware update progress information. Value range: · A value from 1 to 100 indicates the progress percentage. · A value of -1 indicates the firmware update has failed. · A value of -2 indicates that the firmware download has failed. · A value of -3 indicates that firmware verification has failed. · A value of -4 indicates that the firmware installation has failed.
desc	String	Description of the current step. If an exception occurs, this parameter displays an error message.

Request firmware information from IoT Platform

- **Request topic:** `/ota/device/request/{productKey}/{deviceName}`

Request message

```
{
  "id": "123",
  "params": {
    "version": "1.0.1"
  }
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Version information of the firmware.

Response message:

- **Message with firmware information:**

```
{
  "code": "1000",
  "data": {
    "size": 93796291,
    "sign": "f8d85b250d4d787a9f483d89a9747348",
    "version": "1.0.1.9.20171112.1432",
    "url": "https://the_firmware_url",
    "signMethod": "Md5",
    "md5": "f8d85b250d4d787a9f483d89a9747348"
  },
  "id": "8758548588458",
  "message": "success"
}
```

- **No firmware file for update**

```
{
  "code": 500,
  "message": "none upgrade operation of the device."
}
```

10.11 Remote configuration

This article introduces Topics and Alink JSON format requests and responses for remote conficuration. For how to use remote configuration, see *#unique_71* in User Guide.

Device requests configuration information from IoT Platform

Upstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/config/get`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/config/get_reply`

Request message

```
{
  "id": 123,
  "version": "1.0",
  "params": {
    "configScope": "product",
    "getType": "file"
  }
}
```

Response message

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  }
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently, the value is 1.0.

Parameter	Type	Description
configScope	String	Configuration scope. Currently, IoT Platform supports only product dimension configuration. Value: product.
getType	String	Desired file type of the configuration. Currently, the supported type is file. Set the value to file.
configId	String	ID of the configuration.
configSize	Long	Size of the configuration file, in bytes.
sign	String	Signature value.
signMethod	String	Signing method. The supported signing method is Sha256.
url	String	The OSS address where the configuration file is stored.
code	Integer	Result code. A value of 200 indicates that the operation is successful, and other status codes indicate that the operation has failed.

Error codes

Error code	Error message	Description
6713	thing config function is not available	Remote configuration feature of the product has been disabled. On the Remote Configuration page of the IoT Platform console, enable remote configuration for the product .
6710	no data	Not found any configured data.

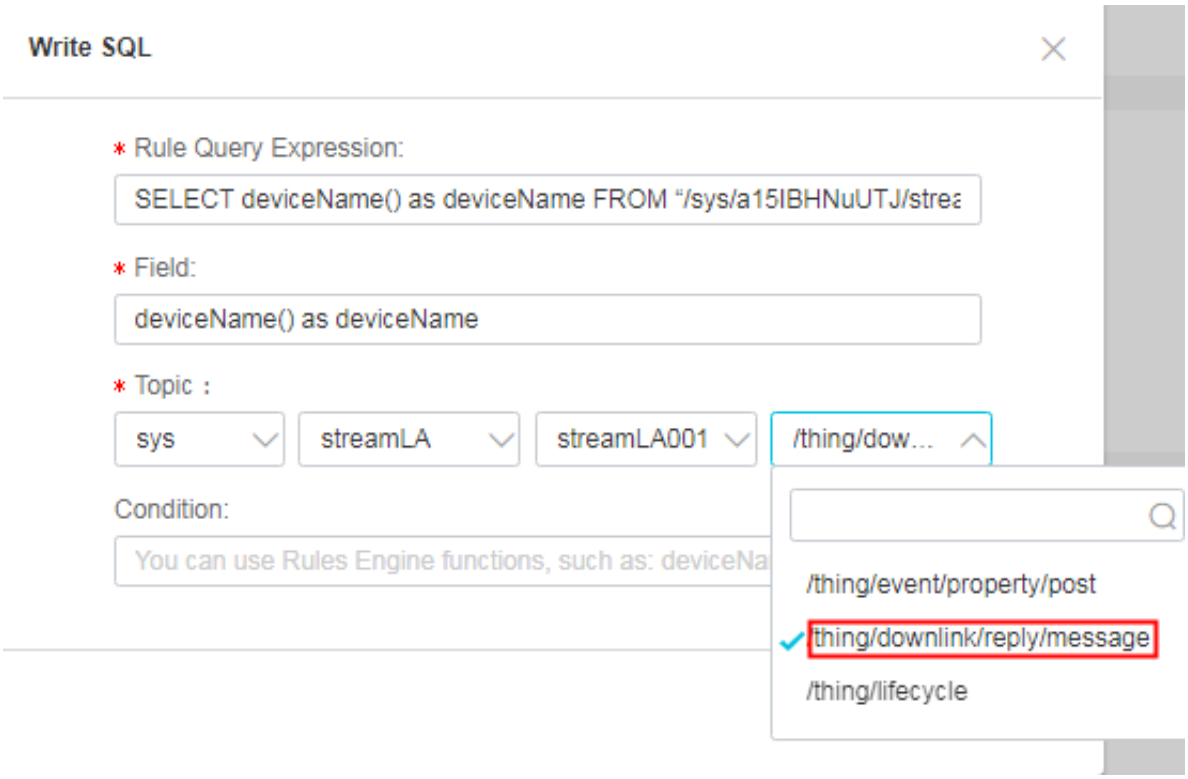
Push configurations in the IoT Platform console to devices.

Downstream

- **Request topic:** `/sys/{productKey}/{deviceName}/thing/config/push`
- **Reply topic:** `/sys/{productKey}/{deviceName}/thing/config/push_reply`

Devices subscribe to this configuration push topic for configurations that is pushed by IoT Platform. After you have edited and submitted a configuration file in the IoT Platform console, IoT Platform pushes the configuration to the devices in an asynchronous method. IoT Platform subscribes to a data exchange topic for the result of asynchronous calls. The data exchange topic is `/{productKey}/{deviceName}/thing/downlink/reply/message`.

You can use *Rules Engine* to forward the results returned by the devices to another Alibaba Cloud product. The following figure shows an example of rule action configuration.



Request message:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
```

```
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0
.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXX
&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJ1q6
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP%
2BpKWSKuGfLC1dysQcO1wEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTknxR7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fV4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK
8CfNOkfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "getType": "file"
  },
  "method": "thing.config.push"
}
```

Response message

```
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

Parameter description

Parameter	Type	Description
id	String	Message ID. IoT Platform generates IDs for downstream messages.
version	String	Protocol version. Currently, the value is 1.0.
configScope	String	Configuration scope. Currently, IoT Platform supports only product dimension configuration. Value: product.
getType	String	Desired file type of the configuration. Currently, the supported type is file. Set the value to file.
configId	String	ID of the configuration.
configSize	Long	Size of the configuration file, in bytes.
sign	String	Signature value.

Parameter	Type	Description
signMethod	String	Signing method. The supported signing method is Sha256.
url	String	The OSS address where the configuration file is stored.
method	String	Request method. The value is `thing.config.push`.
code	Integer	Result code. For more information, see Common codes on devices.

10.12 Device network status

Devices that are connected to Wi-Fi can report the network status to the cloud through specified device topics. This article describes device topics, data formats, and network errors that are related to the reporting of device network status.

> **Note:**
> **If your devices use AliOS Things of version 3.0 or later, the system automatically monitors and reports network status data.**

Device automatically reports status

Data is reported through the following topics.

Request topic: `/sys/{productKey}/{deviceName}/_thing/diag/post`

Response topic: `/sys/{productKey}/{deviceName}/_thing/diag/post_reply`

The following is the request format for the Alink protocol.

- *Current data*: **the data that is immediately reported after being collected by the device.**

  The device immediately reports the network status data for the following two scenarios.

  - **When a network error occurs, the device immediately reports the error to IoT Platform.**
  - **If you have set scheduled collection, the device collects data at the specified time and immediately reports the data.**

  For example, the device detects a network error at 08:10:29 of August 22, 2019, and immediately reports the data. Then, the data format for the network error is as follows:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "p": {
      "wifi": {
        "rssi": 75,
        "snr": 20,
        "per": 10,
        "err_stats":"10,02,01;10,05,01"
      },
      "_time": 1566432629000
    },
    "model": "quantity=single|format=simple|time=now"
  }
}
```

Note:
If no error occurs, the `err_stats` parameter is empty.

- *Historical data*: **the data that is not reported immediately. In most cases, the device may delay the reporting of the collected network metrics. A device can report historical data in batches.**

  Data format:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "p": [
      {
        "wifi": {
          "rssi": 75,
          "snr": 20,
          "per": 10,
```

```
            "err_stats":"10,02,01;10,05,01"
          },
          "_time": 1566432629000
        }
      ],
      "model": "format=simple|quantity=batch|time=history"
    }
}
```

Note:

If no error occurs, the `err_stats` parameter is empty.

Table 10-18: Request parameters

Field	Type	Description
`id`	**String**	**The message ID. The message ID must be a number string and it must be unique among all messages for a device.**
`version`	**String**	**The protocol version. Set the value to `1.0`.**
`params`	**Object**	**Input parameters of the request.**
`wifi`	**Object**	**The network connection mode of the device is WiFi. This parameter value includes four metrics of the network status.**
`rssi`	**Integer**	**The received signal strength.**
`snr`	**Integer**	**The signal-to-noise ratio of the wireless signal.**
`per`	**Integer**	**The data packet loss rate.**
`err_stats`	**String**	**The error message. This parameter is only included in the reported data only when the device detects a network error.** **Format: `"type,code,count;type,code,count"`.** **Example: `"10,02,01;10,05,01"`.** **Parameter description:** • `type`: **the error type** • `code`: **the error code** • `count`: **the number of errors** **For specific errors, you can refer to the err_stats table below.**

Field	Type	Description
_time	Long	The timestamp of the network status. Note: The timestamp can be empty.
model	String	The model of the message body. Valid values: · format: the data format. Only simple format is supported. It indicates that the data is in a simplified format. · quantity: the number of data records to report. - single: indicates that a single data record is reported. - batch: indicates that multiple data records are reported. This option is only used to report historical data. · time: reports the data based on the collection time. - now: reports the current data. - history: reports the historical data.

Table 10-19: err_stats

Error type	Description	Cause
0x00	Wireless environment parameters.	· The signal strength (RSSI): 0x01 · Signal-to-noise ratio (SNR): 0x02 · Packet loss rate (drop ratio): 0x03
0x10	The device failed to connect to IoT Platform.	· Router connection failure (Wi-Fi fail): 0x01 · The device failed to acquire the IP address (DHCP fail): 0x02 · DNS failed to resolve the domain name DNS (DNS fail): 0x03 · TCP handshake failed (TCP fail): 0x04 · TLS handshake failed (TLS fail): 0x05

Error type	Description	Cause
0x20	Network exceptions between devices and IoT Platform.	· IoT Platform rejected the connection from the device (CLOUD_REJECT): 0x01 · Errors occurred during device upload or download (RW_EXCEPTION): 0x02 · Errors occurred during the pinging between the device and IoT Platform (PING_EXCEPTION): 0x03
0x30	Device runtime exceptions.	· The watchdog timer reset (WD_RST): 0x01 · Unexpected restart of the device storage (PANIC_ERR): 0x02 · Reboot errors after the device powers off (RE-POWER): 0x03 · Reboot errors (FATAL_ERR): 0x04
0x40	Dynamic memory monitoring.	· Total memory (type of total size): 0x01 · Total idle memory (type of free size): 0x02
0x50	BLE exceptions.	N/A

Response format:

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {}
}
```

Table 10-20: Response parameters

Field	Type	Description
id	String	The message ID. It is a number string.
code	Integer	The response code. A value of 200 indicates that the request is successful.

Field	Type	Description
version	String	The protocol version. The current protocol version is `1.0`.
data	Object	If the request is successful, the data object is empty.

10.13 Common codes on devices

Common codes on devices indicate the results that are returned to IoT Platform in response to requests from IoT Platform.

Result code	Message	Description
200	success	The request is successful.
400	request error	Internal service error.
460	request parameter error	The request parameters are invalid. The device has failed input parameter verification.
429	too many requests	The system is busy. This code can be used when the device is too busy to process the request.
100000-110000	Device-specific error messages	Devices use numbers from 100000 to 110000 to indicate device-specific error messages.

11 Error codes for device SDKs

This topic describes error codes for device SDKs.

Common error codes

Table 11-1: Common error codes and descriptions

Error code	Cause	Solution
400	An error occurred while processing the request.	Submit a ticket.
429	Traffic throttling is triggered due to frequent requests.	Submit a ticket.
460	The data reported by the device is empty, the format of the parameters is invalid, or the number of parameters has reached the upper limit.	Follow the data formats described in *Communications over Alink protocol*.
500	An unknown error occurred in the system.	Submit a ticket.
5005	An error occurred while querying the product information.	Check the product information in the console and make sure that the ProductKey is correct.
5244	An error occurred while querying the metadata of LoRaWAN-based products.	Submit a ticket.
6100	An error occurred while querying the information about the specified device.	Log on to the console and check whether the device information on the Devices page is correct.
6203	An error occurred while parsing the topic.	Submit a ticket.
6250	An error occurred while querying the product information.	Check the product information in the console and make sure that the ProductKey is correct.
6204	The specified device is disabled. You cannot perform any operation on this device.	Check the status of the device on the Devices page in the console.

Error code	Cause	Solution
6450	The method parameter is missing after the pass-through (custom) data is parsed to the standard Alink format.	On the Device Log page in the console, or in the local log file of the device, check whether the data reported by the device contains the method parameter.
6760	An error occurs in the system.	Submit a ticket.

Table 11-2: Common error codes about parsing scripts

Error code	Cause	Solution
26001	The system does not find any parsing script.	Navigate to the Data Parsing tab in the console and make sure that the script has been submitted. **Note:** You cannot run scripts that are not submitted.
26002	The script runs correctly, but it contains errors.	Use the same data to test the script. Check the error message and revise the script. We recommend that you test the script on a local device before you submit the script to IoT Platform.
26006	The script runs correctly but it contains errors. The script must contain the protocolToRawData and rawDataToProtocol methods. An error occurs if these methods are missing.	Navigate to the Data Parsing tab in the console and check whether the protocolToRawData and rawDataToProtocol methods exist.

Error code	Cause	Solution
26007	The script runs correctly, but the format of the response is invalid. The script must contain the protocolToRawData and rawDataToProtocol methods . The protocolToRawData method must return a byte [] array, and the rawDataToP rotocol method must return a JSON object. An error occurs if the response is not in the required format.	Test the script in the console or on a local device, and check whether the format of the responses returned by these methods is valid.
26010	Traffic throttling is triggered due to frequent requests.	Submit a ticket.

Table 11-3: Common error codes about TSL models

Error code	Cause	Solution
5159	When the system verifies parameters based on the TSL model, an error occurred while querying the property.	Submit a ticket.
5160	When the system verifies parameters based on the TSL model, an error occurred while querying the event.	Submit a ticket.
5161	When the system verifies parameters based on the TSL model, an error occurred while querying the service.	Submit a ticket.
6207	The Alink data reported by the device or the data returned after the system parses the custom data is not in the JSON format.	Follow the data formats described in *Device properties, events, and services* when you report data.

Error code	Cause	Solution
6300	The specified method does not exist. When the system verifies parameters based on the TSL model, the `method` parameter does exist in the Alink data reported by the device, or after the pass-through (custom) data is parsed to Alink format.	On the Device Log page in the console, or in the local log file of the device, check whether the data reported by the device contains the `method` parameter.
6301	When the system verifies parameters based on the TSL model, the data type is specified as an array. However, the type of data reported by the device is not an array.	Navigate to the Define Feature tab in the console, and check the data type specified in the TSL model. Report data with the required data type.
6302	Some required input parameters of the service are not set.	Log on to the console and check the TSL model. Make sure that the required input parameters are correctly set.
6306	When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameters are different from those specified in the TSL model. · The values of the input parameter are not within the value range specified in the TSL model.	Log on to the console and check the TSL model. Make sure that the data types of the input parameters are the same as those defined in the TSL model , and the parameter values are within the value range specified in the TSL model.

Error code	Cause	Solution
6307	The input parameter does not comply with the 32-bit float data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameter are different from those specified in the TSL model. · The values of the input parameters are not within the value range specified in the TSL model.	
6308	The input parameters do not comply with the Boolean data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameters are different from those specified in the TSL model. · The values of the input parameters are not within the value range specified in the TSL model.	

Error code	Cause	Solution
6310	The input parameters do not comply with the text data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the parameters are different from those specified in the TSL model. · The length of the parameters exceeds the upper limit specified in the TSL model.	
6322	The input parameters do not comply with the 64-bit float data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameters are different from those specified in the TSL model. · The values of the input parameters are not within the value range specified in the TSL model.	
6304	The input parameters cannot be found in the struct specified in the TSL model.	Log on to the console and check the TSL model. Make sure that the data types of the input parameters are correct.
6309	The input parameters do not comply with the enum data specifications specified in the TSL model.	

Error code	Cause	Solution
6311	The input parameters do not comply with the date data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameters are different from those specified in the TSL model. · The input data is not a UTC timestamp.	
6312	The input parameters do not comply with the struct data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The data types of the input parameters are different from those specified in the TSL model. · The number of the parameters contained in the struct is different from that specified in the TSL model.	
6320	The specified property cannot be found in the TSL model of the device.	Log on to the console and check whether the specified property exists in the TSL model. If the property does not exist, add the property.
6321	The Identifier parameter of the property, event, or service is not set.	Submit a ticket.
6317	Parameters required in the TSL model are not set, such as the `type` and `specs` parameters.	Submit a ticket.

Error code	Cause	Solution
6324	The input parameters do not comply with the array data specifications specified in the TSL model. When the system verifies parameters based on the TSL model, the following errors may be found: · The elements in the passed-in array do not match the array definition in the TSL model. · The number of elements in the array exceeds the upper limit specified in the TSL model.	· Log on to the console and check the array definition in the TSL model. · View the log reported by the device and check the number of elements in the array reported by the device.
6325	The type of elements in the array is not supported by IoT Platform. Currently, the following element types are supported: int32, float, double, text, and struct.	Check whether the element type is supported by IoT Platform.
6326	The format of the time field reported by the device is invalid.	Follow the formats described in *Device properties, events, and services*. Report data in the required formats.
6328	The value of the input parameter is not an array.	Log on to the console and check the TSL model. Make sure that the data type of the input parameter is array.
System error codes		
6318	An error occurred in the system while parsing the TSL model.	Submit a ticket.
6313		
6329		
6323		
6316		
6314		
6301		

Device registration error codes

Request topic: /sys/{productKey}/{deviceName}/thing/sub/register.

Error codes: 460, 5005, 5244, 500, 6288, 6100, 6619, 6292, and 6203.

The following table lists the causes and solutions of errors that may occur when you register a device. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6288	Dynamic registration is disabled for the device.	Log on to the console and enable dynamic registration on the Product Details page.
6619	The device has been bound to another gateway.	Navigate to the Device Information tab in the console and check whether the sub-device has already been bound to a gateway.

Unique-certificate-per-product authentication

Error codes: 460, 6250, 6288, 6600, 6289, 500, and 6292.

The following table lists the causes and solutions of errors that may occur when you dynamically register a directly connected device based on unique-certificate-per-product authentication. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6288	Dynamic registration is disabled for the device.	Log on to the console and enable dynamic registration on the Product Details page.
6292	The algorithm for calculating the signature is not supported by IoT Platform.	Use algorithms that are supported by the `signMethod` parameter, as described in *Device identity registration*.
6600	An error occurred while verifying the signature.	Use the supported algorithms to calculate and verify the signature, as described in *Device identity registration*.

Error code	Cause	Solution
6289	The device has already been activated.	Log on to the console and check the status of the device.

Topology error codes

Add topological relationships

Request topic: /sys/{productKey}/{deviceName}/thing/topo/add.

Error codes: 460, 429, 6402, 6100, 401, 6204, 6400, and 6203.

The following table lists the causes and solutions of error that may occur when you add a topological relationship between the gateway and a sub-device. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
401	The system failed to verify the signature while adding the topological relationship.	Use the supported algorithms to calculate and verify the signature, as described in *Add a topological relationship*.
6402	The gateway and sub-device are the same device. When you add a topological relationship , you must not add the current gateway to itself as a sub-device .	View the information of all existing sub-devices, and check whether the gateway and a sub-device have the same information.
6400	The number of sub-devices that you have added to the gateway has reached the upper limit.	Log on to the console and check the number of existing sub-devices on the Sub-device Management tab page. For more information about the limits, see *#unique_74*.

Delete topological relationships

Request topic: /sys/{productKey}/{deviceName}/thing/topo/delete.

Error codes: 460, 429, 6100, 6401, and 6203.

The following table lists the cause and solution of the error that may occur when you delete a topological relationship between the gateway and a sub-device. For

more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6401	The topological relationship does not exist when the system verifies the topological relationship.	Log on to the console, click Devices in the left-side navigation pane, and then click the Sub-device Management tab on the Device Details page. You can then view the information about the sub-device.

Obtain topological relationships

Request topic: /sys/{productKey}/{deviceName}/thing/topo/get

Error codes: 460, 429, 500, and 6203. For more information about these error codes, see the Common error codes section in this topic.

The gateway reports a detected sub-device

Request topic: /sys/{productKey}/{deviceName}/thing/list/found.

Error codes: 460, 500, 6250, 6280, and 6203.

The following table lists the cause and solution of error that may occur when a gateway reports a detected sub-device. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6280	The name of the sub-device reported by the gateway is invalid. The device name can contain Chinese characters, letters, digits, and underscores (_). It must be from 4 to 32 characters in length. Each Chinese character accounts for two character spaces.	Check whether the name of the sub-device reported by the gateway is valid.

Sub-device connection and disconnection error codes

A sub-device connects to IoT Platform

Request topic: /ext/session/${productKey}/${deviceName}/combine/login.

Error codes: 460, 429, 6100, 6204, 6287, 6401, and 500.

A sub-device automatically disconnects from IoT Platform

Error messages are sent to this topic: /ext/session/{productKey}/{deviceName}/combine/logout_reply.

Error codes: 460, 520, and 500.

A sub-device is disconnected from IoT Platform by force

Error messages are sent to this topic: /ext/error/{productKey}/{deviceName}.

Error codes: 427, 521, 522, and 6401.

A sub-devices fails to send a message

Error messages are sent to this topic: /ext/error/{productKey}/{deviceName}.

Error code: 520.

The following table lists the causes and solutions of errors that may occur when a sub-device connects to or disconnects from IoT Platform. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
427	The device frequently reconnects to IoT Platform. The same device certificate information is used to connect another device to IoT Platform . This disconnects the current device from IoT Platform.	Navigate to the Device Details page in the console and check when the device was most recently connected to IoT Platform. You can then determine whether the same device certificate information is used to connect another device to IoT Platform.
428	The number of sub-devices that you have added to the specified gateway has reached the upper limit. Currently, you can add up to 1,500 sub-devices to each gateway.	Check the number of sub-devices that you have added to the gateway.
521	The device has been deleted.	Navigate to the Devices page in the console and check whether the device has been deleted.

Error code	Cause	Solution
522	The device has been disabled.	Navigate to the Devices page in the console and check the status of the device.
520	An error occurred with the session between the sub-device and IoT Platform. · The specified session does not exist because the sub-device is not connected to IoT Platform, or the sub-device is already disconnected from IoT Platform. · The session exists, but the session is not established through the current gateway.	
6287	An error occurred while verifying the signature based on the ProductSecret or DeviceSecret.	Use the supported algorithms to calculate and verify the signature, as described in *Connect and disconnect sub-devices*.

Property, event, and service error codes

A device reports a property

Request topic for pass-through data: /sys/{productKey}/{deviceName}/thing/model/up_raw.

Request topic for Alink data: /sys/{productKey}/{deviceName}/thing/event/property/post.

Error codes: 460, 500, 6250, 6203, 6207, 6313, 6300, 6320, 6321, 6326, 6301, 6302, 6317, 6323, 6316, 6306, 6307, 6322, 6308, 6309, 6310, 6311, 6312, 6324, 6328, 6325, 6200, 6201, 26001, 26002, 26006, and 26007.

The following table lists the cause and solution of error 6106 that may occur when the device reports a property. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6106	The number of properties that are reported by the device has reached the upper limit. A device can report up to 200 properties at the same time.	Log on to the console, choose Maintenance > Device Log, and check the number of properties reported by the device. You can also check this information in the local log file of the device.

The device reports an event

Request topic for pass-through data: /sys/{productKey}/{deviceName}/thing/model/up_raw.

Request topic for Alink Data: /sys/{productKey}/{deviceName}/thing/event/{tsl.identifier}/post.

Error codes: 460, 500, 6250, 6203, 6207, 6313, 6300, 6320, 6321, 6326, 6301, 6302, 6317, 6323, 6316, 6306, 6307, 6322, 6308, 6309, 6310, 6311, 6312, 6324, 6328, 6325, 6200, 6201, 26001, 26002, 26006, and 26007.

For more information about these error codes, see the Common error codes section in this topic.

The gateway reports data of multiple sub-devices at the same time

Request topic for pass-through data: /sys/{productKey}/{deviceName}/thing/model/up_raw.

Request topic for Alink data: /sys/{productKey}/{deviceName}/thing/event/property/pack/post.

Error codes: 460, 6401, 6106, 6357, 6356, 6100, 6207, 6313, 6300, 6320, 6321, 6326, 6301, 6302, 6317, 6323, 6316, 6306, 6307, 6322, 6308, 6309, 6310, 6311, 6312, 6324, 6328, 6325, 6200, 6201, 26001, 26002, 26006, and 26007.

The following table lists the causes and solutions of errors that may occur when the gateway reports data of multiple sub-devices at the same time. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6401	The topological relationship does not exist.	Navigate to the Sub-device Management tab in the console and check the information about the sub-device.
6106	The number of properties reported by the device has reached the upper limit. A device can report up to 200 properties at the same time.	Log on to the console, choose Maintenance > Device Log, and check the number of properties reported by the device. You can also check this information in the local log file of the device.
6357	The amount of data reported by the gateway has reached the upper limit. When the gateway reports data for sub-devices, the gateway can report data of up to 20 devices at the same time.	Check the report records in the local log file of the device.
6356	The number of events reported by the gateway has reached the upper limit. When the gateway reports data for sub-devices, the gateway can report up to 200 events at the same time.	Check the report records in the local log file of the device.

Error codes about desired device property values

Obtain desired property values

Request topic: /sys/{productKey}/{deviceName}/thing/property/desired/get.

Error codes: 460, 6104, 6661, and 500.

The following table lists the causes and solutions of errors that may occur when you perform operations on desired device property values. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6104	The number of properties contained in the request has reached the upper limit. A request can contain up to 200 properties.	Log on to the console, choose Maintenance > Device Log, and check the number of properties in the reported data. You can also check this information in the local log file of the device.
6661	An error occurred while querying the desired property.	Submit a ticket.

A device clears the desired property values

Request topic: /sys/{productKey}/{deviceName}/thing/property/desired/delete.

Error codes: 460, 6104, 6661, 500, 6207, 6313, 6300, 6320, 6321, 6326, 6301, 6302, 6317, 6323, 6316, 6306, 6307, 6322, 6308, 6309, 6310, 6311, 6312, 6324, 6328, and 6325.

Device tag error codes

A device reports tag information

Request topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/update.

Error codes: 460 and 6100.

A device deletes tag information

Request topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/delete.

Error codes: 460 and 500.

Error codes about obtaining TSL models

Request topic: /sys/{productKey}/{deviceName}/thing/dsltemplate/get.

Error codes: 460, 5159, 5160, and 5161.

Error codes about querying firmware information

Request Topic: /ota/device/request/${YourProductKey}/${YourDeviceName}.

Note:

- **In the AbstractAlinkJsonMessageConsumer class that provides methods for upgrading firmware, only the `DeviceOtaUpgradeReqConsumer` method returns error codes. Other methods do not return error codes or data.**

- **The topic used to query firmware information is the same as that used to return responses.**

Error codes: 429, 9112, and 500.

The following table lists the cause and solution of the error that may occur when a device queries firmware information. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
9112	The system failed to query information about the specified device.	Check whether the device information specified in the console is correct.

Error codes about obtaining the configuration information

Request topic: /sys/{productKey}/{deviceName}/thing/config/get.

Error codes: 460, 500, 6713, and 6710.

The following lists the causes and solution of errors that may occur when a device attempts to obtain the configuration information. For more information about the other error codes, see the Common error codes section in this topic.

Error code	Cause	Solution
6713	Remote configuration services are unavailable. The remote configuration feature of the specified product is disabled.	Log on to the console, choose Maintenance > Remote Config, and enable the remote configuration feature for the specified product.
6710	The system failed to query the remote configuration information.	Log on to the console, choose Maintenance > Remote Config, and check whether you have edited the configuration file for the specified product.