ALIBABA CLOUD

Alibaba Cloud

物联网平台

Device Management

Document Version: 20220705

Alibaba Cloud

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 because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should 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 this document based on the "status quo", "being defective", and "existing functions" of its products and services. 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 take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, 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 directly contact Alibaba Cloud for any errors of this document.

Document conventions

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

Table of 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 Create a product.
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 Create multiple devices at a time and Create a device.

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 Download device SDKs.

   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-certificate-per-device verification: 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-certificate-per-product verification: 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 disconnection 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. 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 enable. Turn

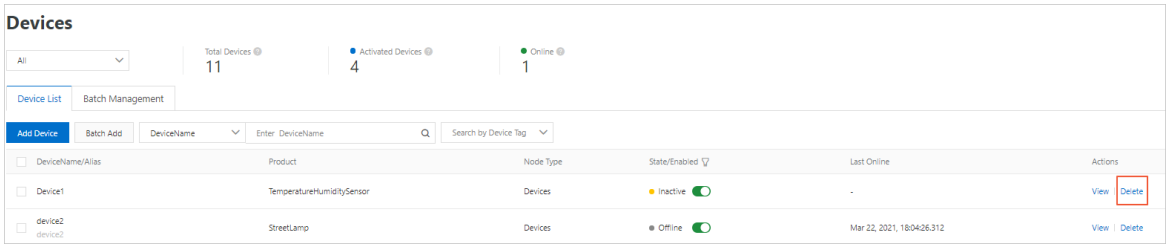
on the **Enabled** switch.

1.4. Delete a device

You can delete a device, regardless of its status, from IoT Platform. After you delete a device from IoT Platform, the certificate of the device becomes invalid. Except for the operational logs in the cloud, other data that is associated with the device is also deleted. You can still query the operational logs of the device in the cloud, but you are not able to perform operations on the device in the IoT Platform console.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Devices**.
4. On the **Device List** tab, find the device that you want to delete.
5. Click **Delete** in the Actions column. In the message that appears, click OK.



Result

After you delete a device, the certificate of the device becomes invalid and cannot be restored. You can still query the operational logs of the device in the cloud, but you are not able to perform operations on the device in the IoT Platform console.

2.Data parsing

2.1. What is data parsing?

The standard data format defined in IoT Platform is Alink JSON. For low-end devices with limited resources or devices that require high network throughput, using JSON data to communicate with IoT Platform is inappropriate. You can transmit raw data to IoT Platform. IoT Platform provides the data parsing feature that converts data between device-specific formats and the JSON format based on the scripts you have submitted.

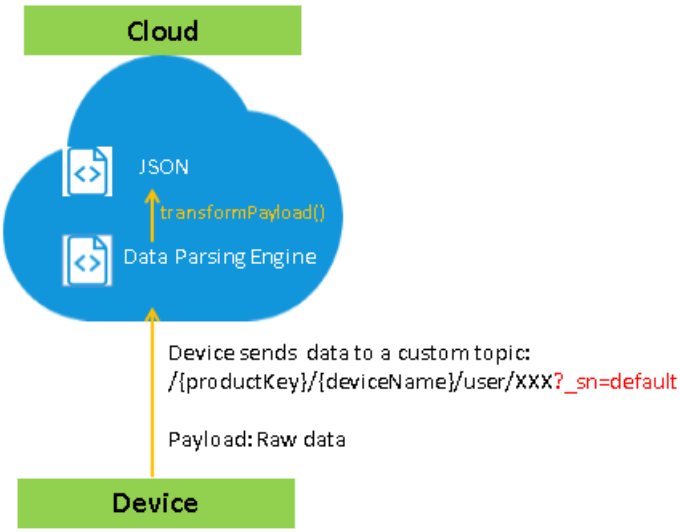
The following two types of data can be parsed:

- Upstream data from custom topics. When devices submit data to the cloud by using custom topics, the payloads are parsed into JSON data.
- Upstream and downstream TSL data. IoT Platform parses the custom TSL data submitted by devices into Alink JSON data and parses Alink JSON data sent from the cloud into custom format data.

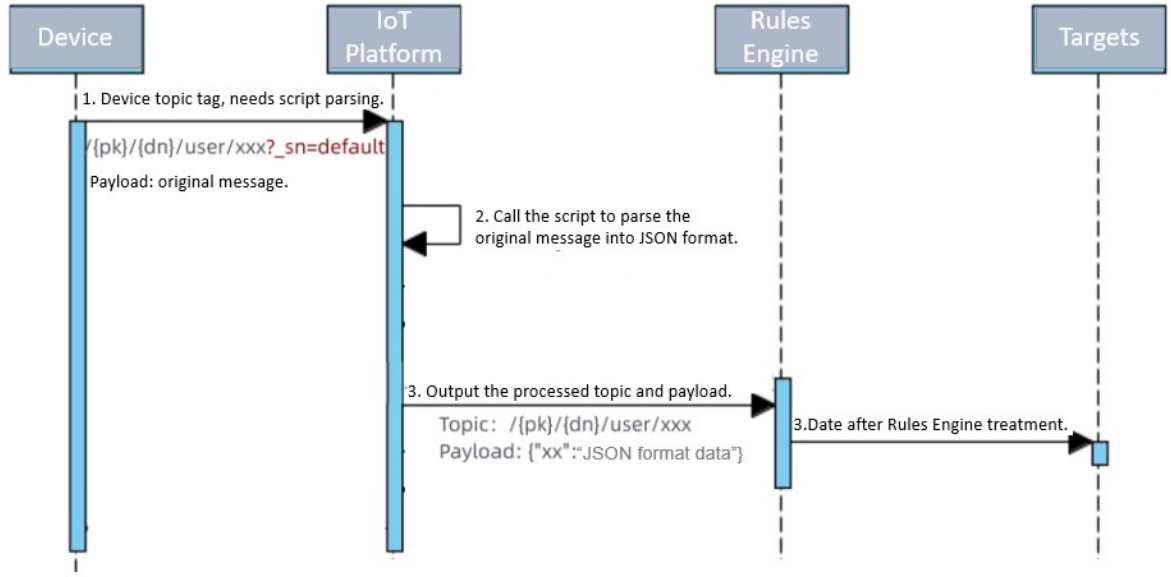
Parse data from custom topics

After a device submits data by using a custom topic that is attached with the `?_sn=default` tag, IoT Platform calls the data parsing script that you submitted to the console to convert the custom format payloads into JSON data for subsequent processing.

The following figure shows the procedure of data parsing.



The following figure shows the procedure of using custom topics to submit data.



For information about how to write a script for parsing custom topic data, see the following topics:

Submit a data parsing script

JavaScript sample script

Python sample script

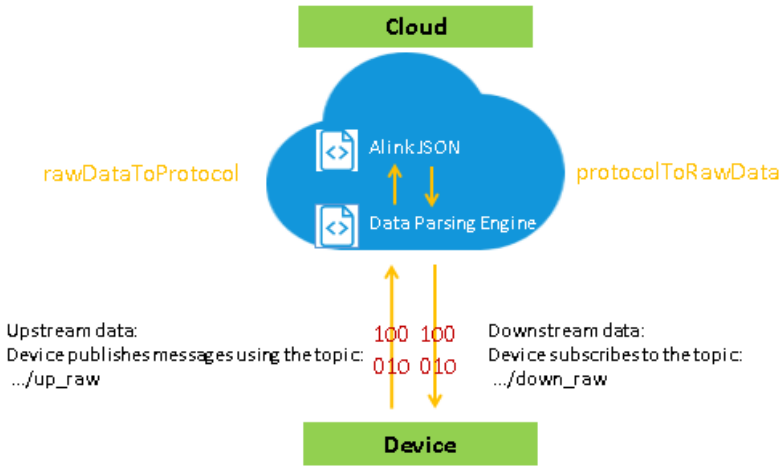
PHP sample script

Parse Thing Specification Language (TSL) data

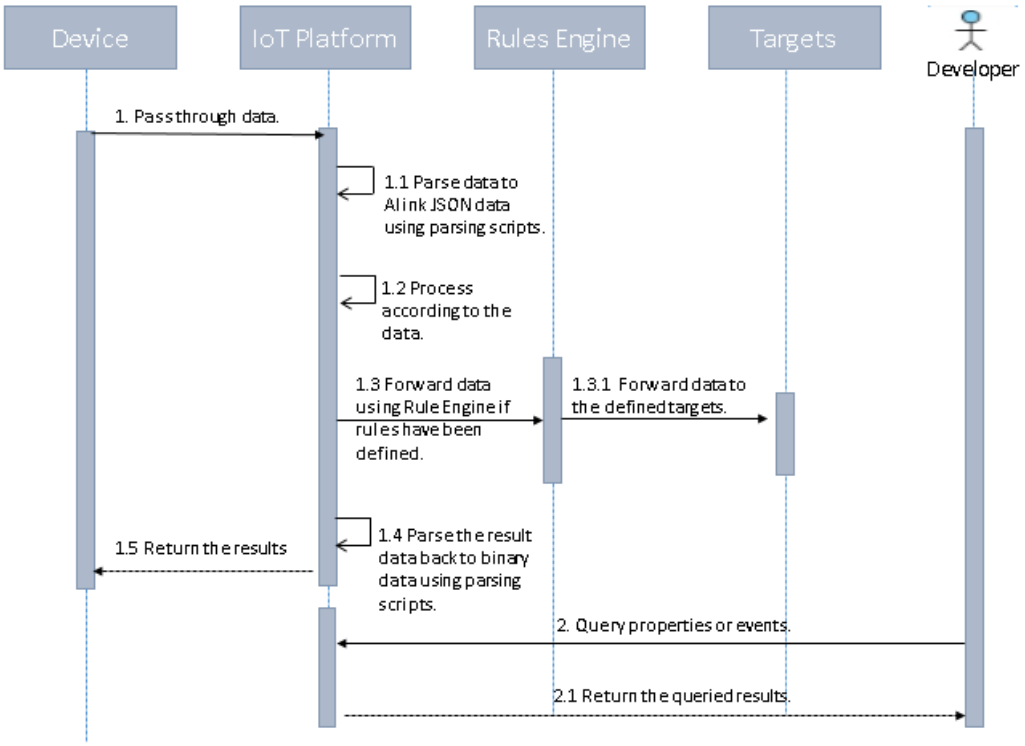
When a device of a product whose **Data Format** parameter is set to **Custom** communicates with the cloud, IoT Platform calls the data parsing script that you submitted to parse upstream data into standard Alink JSON data and downstream data into custom format data.

After receiving data from the device, IoT Platform runs the script to convert the pass-through data into Alink JSON data for subsequent processing. Before sending data to the device, IoT Platform runs the script to convert the data into custom format data.

The following figure shows the procedure of data parsing.



The following figure shows the procedure of submitting pass-through property or event data (upstream data).



The following figure shows the procedure of calling device services or configuring device properties (downstream data).

References

For information about sample scripts, see Example for parsing TSL data, Sample JavaScript script, Python sample script, and PHP sample script.

2.2. Parse custom topic data

2.2.1. Submit a data parsing script

You need to submit a data parsing script in the IoT Platform console. When a device reports data, the device can send the data to a custom topic that has the parsing tag `?_sn=default`. After IoT Platform receives the data, IoT Platform calls the data parsing script, converts the data in the custom format into a JSON structure, and then forwards the JSON-formatted data to the subsequent business system.

Usage notes

- The parsing of data that is sent to custom topics is supported only in the China (Shanghai) and Japan (Tokyo) region.
- The parsing of data that is sent to custom topics is supported only for devices that are connected by using Message Queuing Telemetry Transport (MQTT).
- IoT Platform parses only data that is reported by devices to the cloud, and does not parse downstream data that is sent from the cloud.

- IoT Platform parses the payloads of data that is reported to the cloud and returns the payloads of the parsed data.
- After IoT Platform parses data, the data still resides in the same topic as that before the parsing. For example, a device sends data to the `/${productKey}/${deviceName}/user/update` topic, and the parsed data still resides in this topic.

Parsing tag

When you configure a device to publish messages to a custom topic, append the parsing tag `?_sn=default` to the custom topic. IoT Platform parses data only when the custom topic to which the data is sent has the parsing tag.

For example, to parse the data that is sent by a device to the `/${productKey}/${deviceName}/user/update` topic into JSON-formatted data, you must specify the following topic when you configure the device: `/${productKey}/${deviceName}/user/update?_sn=default`.

> **Note** When you create a custom topic in the IoT Platform console, do not append the parsing tag to the topic.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Products**.
4. On the **Products** page, find the product that you want to manage and click **View** in the Actions column.
5. On the **Product Details** page, click the **Data Parsing** tab.
6. Select a programming language and write a script in the **Edit Script** field.

   Only JavaScript (ECMAScript 5), Python 2.7, and PHP 7.2 are supported.

   In the script, define one of the following functions based on your selected programming language:

   - JavaScript (ECMAScript 5): transformPayload()
   - Python 2.7: transform_payload()
   - PHP 7.2: transformPayload()

   For information about the complete sample code, see JavaScript sample script, Python sample script, and PHP sample script.

   > **Note** If the **Data Type** parameter of the product is set to **Custom**, you also need to write a script for parsing Thing Specification Language (TSL) data. For information about how to write a script for parsing TSL data, see Example for parsing TSL data.

7. Test the script.
   i. On the **Input Simulation** tab, set the **Simulation Type** parameter to **Custom** and select a device and a topic.
   ii. Enter simulated data that a device reports and click **Run**.

8. After you confirm that the script is valid, click **Submit** to submit the script to IoT Platform.

2.2.2. JavaScript sample script

This article provides a template and a sample script in JavaScript for parsing data that is sent to custom topics.

Script template

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' // The custom topic /user/update.
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' // The custom topic /user/update/er
ror.
/**
 * Call a function to convert the data that is sent by a device to a custom topic into JSON
-formatted data when the device reports data to IoT Platform.
 * topic: an input parameter that specifies the topic to which the device sends messages. T
he value is a string.
 * rawData: an input parameter. The value is a byte array that cannot be empty.
 * jsonObj: an output parameter. The value is a JSON object that cannot be empty.
 */
function transformPayload(topic, rawData) {
var jsonObj = {};
return jsonObj;
}
```

Sample script

Note The following sample script can be used only to parse data that is sent to custom topics. If the **Data Type** parameter of the product is set to **Custom**, you also need to write a script to parse Thing Specification Language (TSL) data. For information about how to write a script to parse TSL data, see Example for parsing TSL data.

For information about the **custom data format**, see Create a product.

```
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' // The custom topic /user/update.
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' // The custom topic /user/update/er
ror.
/*
  Sample data
  Custom topic:
     /user/update, to which data is sent.
  Input parameters:
     topic: /{productKey}/{deviceName}/user/update
     bytes: 0x000000000100320100000000
  Output parameters:
  {
     "prop_float": 0,
     "prop_int16": 50,
     "prop_bool": 1,
     "topic": "/{productKey}/{deviceName}/user/update"
   }
 */
function transformPayload(topic, 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 = {};
    if(topic.includes(SELF_DEFINE_TOPIC_ERROR_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['errorCode'] = dataView.getInt8(0)
    } else if (topic.includes(SELF_DEFINE_TOPIC_UPDATE_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['prop_int16'] = dataView.getInt16(5);
        jsonMap['prop_bool'] = uint8Array[7];
        jsonMap['prop_float'] = dataView.getFloat32(8);
    }
    return jsonMap;
}
```

2.2.3. Python sample script

This article provides a template and a sample script in Python to parse data that is sent to custom topics.

Script template

```
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update' # The custom topic /user/update.
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' # The custom topic /user/update/error.
# Call a function to convert the data in a custom topic into JSON-formatted data when the d
evice submits data to IoT Platform.
# topic: an input parameter that specifies the topic to which the device sends messages. Th
e value is a string.
# rawData: an input parameter. The value is a list of integers that cannot be empty.
# jsonObj: an output parameter. The value is a JSON object that indicates a dictionary.

def transform_payload(topic, rawData):
   jsonObj = {}
   return jsonObj
```

Sample script

Note The following sample script can be used only to parse data that is sent to custom topics. If the **Data Type** parameter of the product is set to **Custom**, you also need to write a script to parse Thing Specification Language (TSL) data. For information about how to write a script to parse TSL data, see Example for parsing TSL data.

For information about the **custom data format**, see Create a product.

```
# coding=UTF-8
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'# Custom topic: /user/update.
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error'# Custom topic: /user/update/error.
# Sample data:
# Custom topic: /user/update. The topic is used to submit data.
# Input parameters:
# topic:  /{productKey}/{deviceName}/user/update
#     bytes:  0x000000001003201000000000
# Output parameters:
# {
# "prop_float": 0,
#     "prop_int16":  50,
#     "prop_bool": 1,
#     "topic": "/{productKey}/{deviceName}/user/update"
#   }
def transform_payload(topic, bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)
    jsonMap = {}
    if SELF_DEFINE_TOPIC_ERROR_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['errorCode'] = bytes_to_int(uint8Array[0:1])
    elif SELF_DEFINE_TOPIC_UPDATE_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['prop_int16'] = bytes_to_int(uint8Array[5:7])
        jsonMap['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        jsonMap['prop_float'] = bytes_to_int(uint8Array[8:])
    return jsonMap
# Convert a byte array to integers.
```

```
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16) 50,
#     "prop_bool": 1,
#     "topic": "/{productKey}/{deviceName}/user/update"
#   }
def transform_payload(topic, bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)
    jsonMap = {}
    if SELF_DEFINE_TOPIC_ERROR_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['errorCode'] = bytes_to_int(uint8Array[0:1])
    elif SELF_DEFINE_TOPIC_UPDATE_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['prop_int16'] = bytes_to_int(uint8Array[5:7])
        jsonMap['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        jsonMap['prop_float'] = bytes_to_int(uint8Array[8:])
    return jsonMap
# Convert a byte array to integers.
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16)
```

2.2.4. PHP sample script

This article provides a template and a sample script in PHP for parsing data that is sent to custom topics.

Script template

You can write a data parsing script in PHP based on the following script template:

```
<? php
/**
 * Call a function to convert the data that is sent by a device to a custom topic into JSON
-formatted data when the device reports data to IoT Platform.
 * $topic: an input parameter that specifies the topic to which the device sends messages.
The value is a string.
 * $rawData: an input parameter. The value is an array of integers.
 * $jsonObj: an output parameter. The value is a JSON object that indicates an association
array. The value of each key must be a string that is not a numeric string such as "10". Ke
y values cannot be empty.
 */
function transformPayload($topic, $rawData)
{
    $jsonObj = array();
    return $jsonObj;
}
```

Notes for script writing

- Do not use global variables or static variables. Otherwise, results may be inconsistent.
- In the script, process data by using the two's complement operation. The complement range for values in the range of [-128,127] is [0,255]. For example, the complement of -1 is 255 in decimal.
- The input parameter of the transformPayload function is an array of integers. Use `0xFF` for the bitwise AND operation to obtain the complement. The return result is an association array. Each key value must contain non-array characters. For example, if a key value is "10", the PHP array retrieves the integer 10.
- The PHP runtime environment is significantly strict with exception handling. If an error occurs, an exception is thrown and subsequent code is not run. To ensure code robustness, you need to capture and process exceptions.

Sample script

Note The following sample script can be used only to parse data that is sent to custom topics. If the **Data Type** parameter of the product is set to **Custom**, you also need to write a script to parse Thing Specification Language (TSL) data. For information about how to write a script to parse TSL data, see Example for parsing TSL data.

For information about the **custom data format**, see Create a product.

```
<? php
/*
  Sample data
  Custom topic:
     /user/update, to which data is sent.
  Input parameters:
     topic: /{productKey}/{deviceName}/user/update
     bytes: 0x000000000100320100000000
  Output parameters:
  {
     "prop_float": 0,
     "prop_int16": 50,
     "prop_bool": 1,
     "topic": "/{productKey}/{deviceName}/user/update"
   }
 */
function transformPayload($topic, $bytes)
{
    $data = array();
    $length = count($bytes);
    for ($i = 0; $i < $length; $i++) {
        $data[$i] = $bytes[$i] & 0xff;
    }
    $jsonMap = array();
    if (strpos($topic, '/user/update/error') ! == false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['errorCode'] = getInt8($data, 0);
    } else if (strpos($topic, '/user/update') ! == false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['prop_int16'] = getInt16($data, 5);
        $jsonMap['prop_bool'] = $data[7];
    }
```

```
    return $jsonMap;
}
function getInt32($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1], $bytes[$index + 2], $bytes[$index +
3]);
    return hexdec(byteArrayToHexString($array));
}
function getInt16($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1]);
    return hexdec(byteArrayToHexString($array));
}
function getInt8($bytes, $index)
{
    $array = array($bytes[$index]);
    return hexdec(byteArrayToHexString($array));
}
function byteArrayToHexString($data)
{
    $hexStr = '';
    for ($i = 0; $i < count($data); $i++) {
        $hexValue = dechex($data[$i]);
        $tempHexStr = strval($hexValue);
        if (strlen($tempHexStr) === 1) {
            $hexStr = $hexStr . '0' . $tempHexStr;
        } else {
            $hexStr = $hexStr . $tempHexStr;
        }
    }
    return $hexStr;
}
function hexStringToByteArray($hex)
{
    $result = array();
    $index = 0;
    for ($i = 0; $i < strlen($hex) - 1; $i += 2) {
        $result[$index++] = hexdec($hex[$i] . $hex[$i + 1]);
    }
    return $result;
}
function concat($array, $data)
{
    return array_merge($array, $data);
}
function toHex($data)
{
    $var = dechex($data);
    $length = strlen($var);
    if ($length % 2 == 1) {
        $var = '0' . $var;
    }
    return $var;
}
```

2.3. TSL Data Parsing

2.3.1. Example for parsing TSL data

This article describes how to write code to parse Thing Specification Language (TSL) data for products whose **Data Type** parameter is set to **Custom**. Sample scripts are provided for parsing upstream and downstream property data.

> **Note** The properties in this example are default module properties. If you use a custom module, each identifier must be in the following format: `Module identifier:Property identifier`. Example: `model1:prop_int16`.

Step 1: Write a script

1. Log on to the IoT Platform console and create a product.

   For more information, see Create a product.

   > **Note** Set the **Data Type** parameter to **Custom**.

2. Define a TSL model for the product. For information about how to set custom properties, see Add a TSL feature.

   In this example, define the three properties that are described in the following table.

Identifier	Data type	Valid values	Read/write type
prop_float	FLOAT	-100 to 100	Read and write
prop_int16	INT32	-100 to 100	Read and write
prop_bool	BOOLEAN	0: enabled, 1: disabled	Read and write

   The TSL model that is defined in this example is used to write the script for parsing upstream and downstream TSL data.

3. Specify lengths for the parameters that are used for data communication. In this example, define the parameter lengths as described in the following tables.

   Requests that devices send to report data

Field	Number of bytes
Frame type	1
Request ID	4
prop_int16	2
prop_bool	1

Field	Number of bytes
prop_float	4

Responses to requests for reporting data

Field	Number of bytes
Frame type	1
Request ID	4
Result code	1

Requests for setting properties

Field	Number of bytes
Frame type	1
Request ID	4
prop_int16	2
prop_bool	1
prop_float	4

Responses to requests for setting properties

Field	Number of bytes
Frame type	1
Request ID	4
Result code	1

4. Write a script.
   i. On the **Products** page, find the product that you want to manage and click **View** in the Actions column.
   ii. On the **Product Details** page, click the **Data Parsing** tab.

iii. Select a programming language and write a script in the **Edit Script** field.

Only JavaScript (ECMAScript 5), Python 2.7, and PHP 7.2 are supported.

In the script, define the two functions that can be called to parse upstream and downstream TSL data. Select the functions based on your selected programming language.

- The function that converts Alink JSON-formatted data into data in a custom format:
  - JavaScript (ECMAScript 5): protocolToRawData
  - Python 2.7: protocol_to_raw_data
  - PHP 7.2: protocolToRawData
- The function that converts data in a custom format into Alink JSON-formatted data:
  - JavaScript (ECMAScript 5): rawDataToProtocol
  - Python 2.7: raw_data_to_protocol
  - PHP 7.2: rawDataToProtocol

For information about the complete sample script, see Sample JavaScript script, Python sample script, and PHP sample script.

> **Note** You also need to write a script for parsing data that is sent to custom topics. For information about how to write the script, see Submit a data parsing script.
>
> For information about the complete sample scripts for parsing data that is sent to custom topics and for parsing TSL data in different programming languages, see Complete sample script in JavaScript, Complete sample script in Python, and Complete sample script in PHP.

Step 2: Test the script online

After you write the script, set the **Simulation Type** parameter and write simulated data to test the script on the **Input Simulation** tab.

> **Note** In this example, the hexadecimal string and JSON-formatted data that are specified and returned are used only for a test. Do not use them for debugging in actual scenarios.

- Test the script by using simulated property data that a device reports to IoT Platform.

  Set the Simulation Type parameter to Upstreamed Device Data, enter the following simulated data, and then click **Run**.

  > **Note** In this example, simulated JavaScript data is used. For more information about the JavaScript sample script, see Sample JavaScript script.
  >
  > For information about Python and PHP sample scripts, see Python sample script and PHP sample script.

  You can use a tool that converts strings to hexadecimal data to convert the JSON-formatted data into hexadecimal data. For example, if the hexadecimal data after the conversion is `00002233441232013fa00000`, enter the following data:

  ```
  0x00002233441232013fa00000
  ```

The data parsing engine converts the pass-through data into JSON-formatted data based on the rules that are defined in the script.

Click the **Parsing Results** tab and view the parsed data.

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

- Test the script by using simulated result data that IoT Platform returns to a device.

  Set the Simulation Type parameter to **Received Device Data**, enter the following JSON-formatted data, and then click **Run**.

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

  The data parsing engine converts the JSON-formatted data into the following data:

  ```
  0x0200003039c8
  ```

- Test the script by using simulated data that IoT Platform sends to a device to request the device to modify properties.

  Set the Simulation Type parameter to Received Device Data, enter the following JSON-formatted 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 the JSON-formatted data into the following data:

  ```
  0x0100003039014d0142f6e76d
  ```

- Test the script by using simulated result data that a device returns to IoT Platform after the device modifies its properties.

Set the Simulation Type parameter to Upstreamed Device Data, enter the following data, and then click **Run**.

```
0x0300223344c8
```

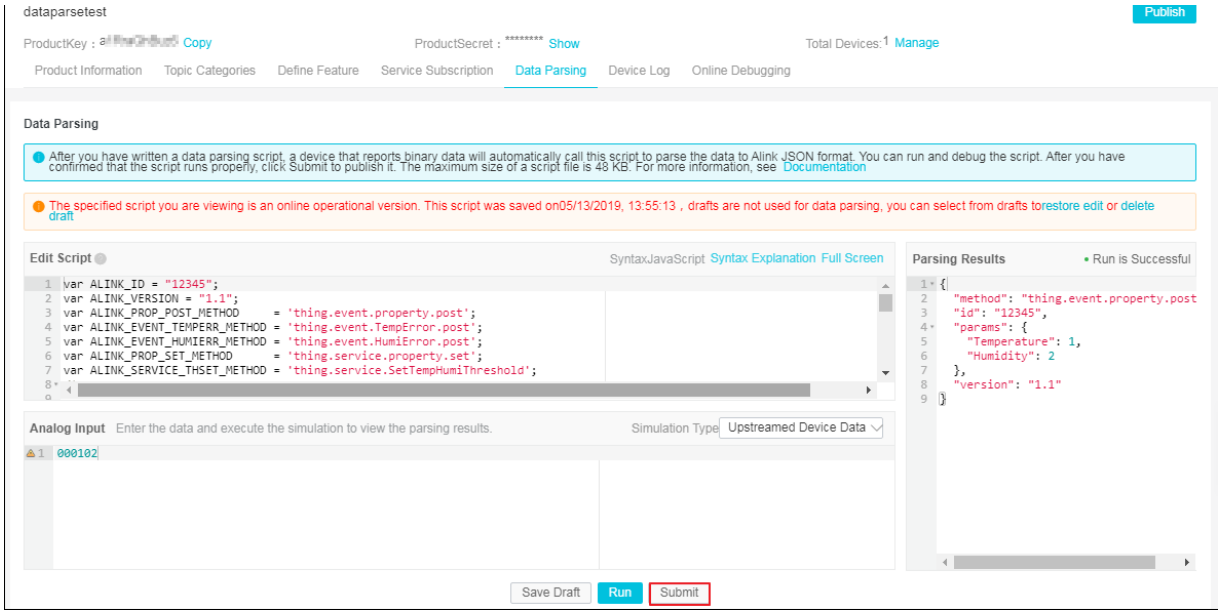
The data parsing engine converts the pass-through data into the following JSON-formatted data:

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

Step 3: Submit the script

After you confirm that the script can properly parse data, click **Submit** to submit the script to IoT Platform. Then, IoT Platform can call this script to parse upstream and downstream data.

> **Note** Scripts can be called only after they are submitted to IoT Platform. Scripts that are in the draft state cannot be called.



Step 4: Use a physical device for debugging

Before you use the script in your business, use a physical device to communicate with IoT Platform. This verifies that IoT Platform can call the script and parse upstream and downstream data.

- Verify that IoT Platform can call the script to parse upstream property data.
  i. Use the device to report property data to IoT Platform, such as `0x000022334412320 13fa00000`.
  ii. In the IoT Platform console, choose **Devices > Devices** in the left-side navigation pane.
  iii. On the Devices page, find the device and click **View** in the Actions column. On the **Device Details** page, click the **TSL Data** tab and then the **Status** tab. Check whether the reported property data exists.

- Verify that IoT Platform can call the script to parse downstream property data.
  i. In the IoT Platform console, choose **Maintenance > Online Debug** in the left-side navigation pane.
  ii. On the Online Debug page, select the product and device that you want to debug. Select **Default Module**. Select the identifier of the property that you want to debug, such as prop_int16. Set the method to **Set**. Enter the following data and click **Send Command**.

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

  iii. Check whether the device receives the command for setting properties.
  iv. On the **Device Details** page, click the **TSL Data** tab and then the **Status** tab. Check whether the device has reported its current property data.

References

- For information about JavaScript (ECMAScript 5) script templates and examples, see Sample JavaScript script.
- For information about Python 2.7 script templates and examples, see Python sample script.
- For information about PHP 7.2 script templates and examples, see PHP sample script.
- For information about the process of data parsing, see What is data parsing?.
- For information about the parsing of data that is sent to custom topics, see Submit a data parsing script.
- For information about the API operations that are used to send messages to devices, see Use of TSL models and Messaging.

2.3.2. Sample JavaScript script

This topic provides a script template and a sample JavaScript script that you can use to parse Thing Specification Language (TSL) data.

Script template

You can write a script in JavaScript to parse TSL data based on the following script template:

> **Note** This template applies only to products whose **Data Format** parameter is set to **Custom**.

```
/**
 * Convert Alink data to data that can be recognized by a device when IoT Platform sends da
ta to the device.
 * jsonObj: the input parameter. The value is a JSON object that cannot be empty.
 * rawData: the output parameter. The value is a byte array that cannot be empty.
 *
 */
function protocolToRawData(jsonObj) {
    return rawdata;
}
/**
 * Convert the custom data of a device to Alink data when the device submits the custom dat
a to IoT Platform.
 * rawData: the input parameter. The value is a byte array that cannot be empty.
 * jsonObj: the output parameter. The value is a JSON object that cannot be empty.
 */
function rawDataToProtocol(rawData) {
    return jsonObj;
}
```

Notes for script writing

- Do not use global variables. Otherwise, results may be inconsistent.
- The script processes data by using the two's complement operation. The complement range for the values in the range of [-128,127] is [0,255]. For example, the complement of -1 is 255 in decimal.
- The rawDataToProtocol function parses the data that is submitted by devices. The input parameter of the function is an integer array. Use `0xFF` for the bitwise AND operation to obtain the complement.
- The protocolToRawData function parses the data that is sent by IoT Platform and returns the result in an array. Each element in the array must be an integer within a value range of [0,255].

Sample script

The following sample script is written based on the properties and protocol that are defined in the Example for parsing TSL data topic.

For more information about the data types supported by TSL models, see Supported data types. After a device submits TSL property data or event data, IoT Platform generates a response whose data is in the Alink JSON format. Before IoT Platform sends the response to the device, IoT Platform uses the script to convert the format of the data to a format that can be recognized by the device. For more information about the Alink JSON format, see Device properties, events, and services.

```
var COMMAND_REPORT = 0x00 // Property data reported by devices.
var COMMAND_SET = 0x01 // Property setting command data from the cloud.
var COMMAND_REPORT_REPLY = 0x02  // Response data for devices reporting properties.
var COMMAND_SET_REPLY = 0x03  // Response data for setting device properties.
var Command_unkown = 0xff  // Unknown commands.
var ALINK_PROP_REPORT_METHOD = 'thing.event.property.post' // The topic for devices to uplo
ad property data to the cloud.
var ALINK_PROP_SET_METHOD = 'thing.service.property.set'  // The topic for IoT Platfor
m to send a property control command to devices.
var ALINK_PROP_SET_REPLY_METHOD = 'thing.service.property.set'  // The topic for devic
```

```
es to report the property setting results to IoT Platform.
var SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  // Custom topic: /user/update.
var SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' // Custom topic: /user/update/error
.
/*
Sample data:
The device reports property data.
Input parameters:
    0x00000000010032010000000
Output result:
    {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":50
,"prop_bool":1},"version":"1.0"}
The device responds after setting properties.
Input parameters:
    0x0300223344c8
Output result:
    {"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]; // The command prefix.
    if (fHead == COMMAND_REPORT) {
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD; // The topic for reporting properties
in the ALink JSON.
        jsonMap['version'] = '1.0'; // The fixed protocol version in the ALink JSON.
        jsonMap['id'] = ''+ dataView.getInt32(1); // The request ID in the ALink JSON.
        var params = {};
        params['prop_int16'] = dataView.getInt16(5); // The value of the prop_int16 propert
y.
        params['prop_bool'] = uint8Array[7]; // The value of the prop_bool property.
        params['prop_float'] = dataView.getFloat32(8); // The value of the prop_float prope
rty.
        jsonMap['params'] = params; // The params field in the ALink JSON.
    } else if(fHead == COMMAND_SET_REPLY) {
        jsonMap['version'] = '1.0'; // The fixed protocol version in the ALink JSON.
        jsonMap['id'] = ''+ dataView.getInt32(1); // The request ID in the ALink JSON.
        jsonMap['code'] = ''+ dataView.getUint8(5);
        jsonMap['data'] = {};
    }
    return jsonMap;
}
/*
Sample data:
IoT Platform pushes a command for setting properties to the device.
Input parameters:
    {"method":"thing.service.property.set","id":"12345","version":"1.0","params":{"prop_flo
at":123.452, "prop_int16":333, "prop_bool":1}}
Output result:
    0x0100003039014d0142f6e76d
```

```
IoT Platform responds after the device reports properties.
Input data:
    {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{}
}
Output result:
    0x0200003039c8
*/
function protocolToRawData(json) {
    var method = json['method'];
    var id = json['id'];
    var version = json['version'];
    var payloadArray = [];
    if (method == ALINK_PROP_SET_METHOD) // Sets properties
    {
        var params = json['params'];
        var prop_float = params['prop_float'];
        var prop_int16 = params['prop_int16'];
        var prop_bool = params['prop_bool'];
        // Raw data is concatenated based on the custom protocol format.
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET)); // The command field
.
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the AL
ink JSON.
        payloadArray = payloadArray.concat(buffer_int16(prop_int16)); // The value of the p
rop_int16 property.
        payloadArray = payloadArray.concat(buffer_uint8(prop_bool)); // The value of the pr
op_bool property.
        payloadArray = payloadArray.concat(buffer_float32(prop_float)); // The value of the
prop_float property.
    } else if (method == ALINK_PROP_REPORT_METHOD) {// The response of the reported data.
        var code = json['code'];
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_SET)); // The command field
.
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the AL
ink JSON.
        payloadArray = payloadArray.concat(buffer_uint8(code));
    } Else {// Unknown commands. Some commands are not processed.
        var code = json['code'];
        payloadArray = payloadArray.concat(buffer_uint8(COMMAND_UNKOWN)); // The command fi
eld.
        payloadArray = payloadArray.concat(buffer_int32(parseInt(id))); // The id in the AL
ink JSON.
        payloadArray = payloadArray.concat(buffer_uint8(code));
    }
    return payloadArray;
}
/*
  Sample data
  Custom topic for reporting data: /user/update
  Input parameters:
     topic: /{productKey}/{deviceName}/user/update
     bytes: 0x000000000100320100000000
  Output parameters:
  {
```

```
      "prop_float": 0,
      "prop_int16": 50,
      "prop_bool": 1,
      "topic": "/{productKey}/{deviceName}/user/update"
    }
 */
function transformPayload(topic, bytes) {
    var uint8Array = new Uint8Array(bytes.length);
    for (int i = 0; i < edges.length; i++) {
        uint8Array[i] = bytes[i] & 0xff;
    }
    var dataView = new DataView(uint8Array.buffer, 0);
    var jsonMap = {};
    if(topic.includes(SELF_DEFINE_TOPIC_ERROR_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['errorCode'] = dataView.getInt8(0)
    } else if (topic.includes(SELF_DEFINE_TOPIC_UPDATE_FLAG)) {
        jsonMap['topic'] = topic;
        jsonMap['prop_int16'] = dataView.getInt16(5);
        jsonMap['prop_bool'] = uint8Array[7];
        jsonMap['prop_float'] = dataView.getFloat32(8);
    }
    return jsonMap;
}
// The following are some helper 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);
}
```

2.3.3. Python sample script

This article provides a script template and a sample script in Python to parse Thing Specification Language (TSL) data.

Script template

You can write a script in Python based on the following script template to parse TSL data:

> **Note** This template applies only to products whose **Data Format** parameter is set to **Custom**.

```
#Convert custom data of a device to Alink data when the device submits data to IoT Platform
.
#rawData: the input parameter. The value is a list of integers that cannot be empty.
#jsonObj: the output parameter. The value is a JSON object that indicates a dictionary and
cannot be empty.
def raw_data_to_protocol(rawData):
    jsonObj = {}
    return jsonObj
#Convert Alink data to data that can be recognized by a device when IoT Platform sends data
to the device.
#jsonData: the input parameter. The value is a JSON object that indicates a dictionary and
cannot be empty.
#rawData: the output parameter. The value is a list of integers within a value range of [0,
255] and cannot be empty.
def protocol_to_raw_data(jsonData):
    rawData = []
    return rawData
```

Notes for script writing

- Do not use global variables. Otherwise, results may be inconsistent.
- The script is used to process data by using the two's complement operation. The complement range for values in the range of [-128,127] is [0,255]. For example, the complement of -1 is 255 in decimal.
- The raw_data_to_protocol function parses the data that is submitted by devices. The input parameter of the function is an integer array. Use `0xFF` for the bitwise AND operation to obtain the complement.
- The protocol_to_raw_data function parses the data that is sent by IoT Platform and returns the result in an array. Each element in the array must be an integer within a value range of [0,255].

Sample script

The following sample script is written based on the properties and protocol that are defined in the Example for parsing TSL data topic.

For more information about the data types supported by TSL models, see Supported data types. After a device submits TSL property data or event data, IoT Platform generates a response whose data is in the Alink JSON format. Before IoT Platform sends the response to the device, IoT Platform uses the script to convert the format of the data to a format that can be recognized by the device. For more information about the Alink JSON format, see Device properties, events, and services.

```
# coding=UTF-8
import struct
import common_util
```

```
import common_util
COMMAND_REPORT = 0x00 # Property data reported by devices.
COMMAND_SET = 0x01 # Property setting command data from the cloud.
COMMAND_REPORT_REPLY = 0x02 # Response data for devices reporting properties.
COMMAND_SET_REPLY = 0x03 # Response data for setting device properties.
Command_unkown = 0xff # Unknown commands.
ALINK_PROP_REPORT_METHOD = 'thing.event.property.post' # The topic for devices to upload pr
operty data to the cloud.
ALINK_PROP_SET_METHOD = 'thing.service.property.set' # The topic for IoT Platform to send a
property control command to devices.
SELF_DEFINE_TOPIC_UPDATE_FLAG = '/user/update'  # Custom topic: /user/update.
SELF_DEFINE_TOPIC_ERROR_FLAG = '/user/update/error' # Custom topic: /user/update/error.
# Example data:
# The device reports data
# Input parameters:
#    0x00000000010032010000000
# Output result:
#    {"method":"thing.event.property.post","id":"1","params":{"prop_float":0,"prop_int16":5
0,"prop_bool":1},"version":"1.0"}
# The device responds after setting properties.
# Input parameters:
#    0x0300223344c8
# Output result:
#    {"code":"200","data":{},"id":"2241348","version":"1.0"}
def raw_data_to_protocol(bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)
    fHead = uint8Array[0]
    jsonMap = {}
    if fHead == COMMAND_REPORT:
        jsonMap['method'] = ALINK_PROP_REPORT_METHOD
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        params = {}
        params['prop_int16'] = bytes_to_int(uint8Array[5:7])
        params['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        params['prop_float'] = bytes_to_int(uint8Array[8:])
        jsonMap['params'] = params
    elif fHead == COMMAND_SET_REPLY:
        jsonMap['version'] = '1.0'
        jsonMap['id'] = str(bytes_to_int(uint8Array[1:5]))
        jsonMap['code'] = str(bytes_to_int(uint8Array[5:]))
        jsonMap['data'] = {}
    return jsonMap
# Example data:
# IoT Platform pushes a command for setting properties to the device.
# Input parameters:
#    {"method":"thing.service.property.set","id":"12345","version":"1.0",
#    "params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1}}
# Output result:
#    0x0100003039014d0142f6e76d
# IoT Platform responds after the device reports properties.
# Input data:
#    {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{
```

```
}}
# Output result:
#    0x0200003039c8
def protocol_to_raw_data(json):
    method = json.get('method', None)
    id = json.get('id', None)
    version = json.get('version', None)
    payload_array = []
    if method == ALINK_PROP_SET_METHOD:
        params = json.get('params')
        prop_float = params.get('prop_float', None)
        prop_int16 = params.get('prop_int16', None)
        prop_bool = params.get('prop_bool', None)
        payload_array = payload_array + int_8_to_byte(COMMAND_SET)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_16_to_byte(prop_int16)
        payload_array = payload_array + int_8_to_byte(prop_bool)
        payload_array = payload_array + float_to_byte(prop_float)
    elif method == ALINK_PROP_REPORT_METHOD:
        code = json.get('code', None)
        payload_array = payload_array + int_8_to_byte(COMMAND_REPORT_REPLY)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)
    else:
        code = json.get('code')
        payload_array = payload_array + int_8_to_byte(COMMAD_UNKOWN)
        payload_array = payload_array + int_32_to_byte(int(id))
        payload_array = payload_array + int_8_to_byte(code)
    return payload_array
# Sample data
# Custom topic for reporting data :/user/update
# Input parameters:
#   topic: /{productKey}/{deviceName}/user/update
#   bytes: 0x00000000010032010000000
# Output parameters:
# {
#     "prop_float": 0,
#     "prop_int16": 50,
#     "prop_bool": 1,
#     "topic": "/{productKey}/{deviceName}/user/update"
#   }
def transform_payload(topic, bytes):
    uint8Array = []
    for byteValue in bytes:
        uint8Array.append(byteValue & 0xff)
    jsonMap = {}
    if SELF_DEFINE_TOPIC_ERROR_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['errorCode'] = bytes_to_int(uint8Array[0:1])
    elif SELF_DEFINE_TOPIC_UPDATE_FLAG in topic:
        jsonMap['topic'] = topic
        jsonMap['prop_int16'] = bytes_to_int(uint8Array[5:7])
        jsonMap['prop_bool'] = bytes_to_int(uint8Array[7: 8])
        jsonMap['prop_float'] = bytes_to_int(uint8Array[8:])
```

```
    return jsonMap
# byte to int.
def bytes_to_int(bytes):
    data = ['%02X' % i for i in bytes]
    return int(''.join(data), 16)
# Converts a byte array to a float without precision.
def bytes_to_float(bytes):
    data = []
    for i in bytes:
        t_value = '%02X' % i
        if t_value % 2 ! = 0:
            t_value += 0
        data.append(t_value)
    hex_str = ''.join(data)
    return struct.unpack('! f', hex_str.decode('hex'))[0]
# Converts an 8-bit integer to a byte array.
def int_8_to_byte(value):
    t_value = '%02X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'
    return hex_string_to_byte_array(t_value)
# Converts a 32-bit integer to a byte array.
def int_32_to_byte(value):
    t_value = '%08X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'
    return hex_string_to_byte_array(t_value)
# Converts a 16-bit integer to a byte array.
def int_16_to_byte(value):
    t_value = '%04X' % value
    if len(t_value) % 2 ! = 0:
        t_value += '0'
    return hex_string_to_byte_array(t_value)
# Converts a float into an integer array.
def float_to_byte(param):
    return hex_string_to_byte_array(struct.pack(">f", param).encode('hex'))
# Converts a hexadecimal string to a byte array.
def hex_string_to_byte_array(str_value):
    if len(str_value) % 2 ! = 0:
        return None
    cycle = len(str_value) / 2
    pos = 0
    result = []
    for i in range(0, cycle, 1):
        temp_str_value = str_value[pos:pos + 2]
        temp_int_value = int(temp_str_value, base=16)
        result.append(temp_int_value)
        pos += 2
    return result
```

2.3.4. PHP sample script

This article provides a script template and a sample script in PHP to parse Thing Specification Language (TSL) data.

Script template

You can write a script in PHP based on the following script template to parse TSL data:

> **Note** This template applies only to products whose **Data Format** parameter is set to **Custom**.

```
<?php
/**
 * Convert Alink data to data that can be recognized by a device when IoT Platform sends do
wnstream data.
 * $jsonObj: the input parameter. The value is a JSON object that indicates an association
array.
 * $rawData: the output parameter. The value is an array of integers within a value range o
f [0,255]. Array elements cannot be empty.
 */
function protocolToRawData($jsonObj)
{
    $rawData = array();
    return $rawData;
}
/**
 * Convert custom data of a device to Alink data when the device submits data to IoT Platfo
rm.
 * $rawData: the input parameter. The value is an array of integers.
 * $jsonObj: the output parameter. The value is a JSON object that indicates an association
array. The value of each key must be a string. However, the string cannot consist of only d
igits such as 10. Key values cannot be empty.
 */
function rawDataToProtocol($rawData)
{
    $jsonObj = array();
    return $jsonObj;
}
/**
 * Convert the data that is sent by a device to a custom topic to JSON data when the device
submits data to IoT Platform.
 * $topic: the input parameter that specifies the topic to which the device sends messages.
The value is a string.
 * $rawData: the input parameter. The value is an array of integers.
 * $jsonObj: the output parameter. The value is a JSON object that indicates an association
array. The value of each key must be a string. However, the string cannot consist of only d
igits such as 10. Key values cannot be empty.
 */
function transformPayload($topic, $rawData)
{
    $jsonObj = array();
    return $jsonObj;
}
```

Notes for script writing

- Do not use global variables or static variables. Otherwise, results may be inconsistent.
- The script is used to process data by using the two's complement operation. The complement range for values in the range of [-128,127] is [0,255]. For example, the complement of -1 is 255 in decimal.
- The rawDataToProtocol function parses the data that is submitted by devices. The input parameter of the function is an integer array. Use `0xFF` for the bitwise AND operation to obtain the complement. The returned result is an association array. Each key value must contain non-array characters. For example, if a key value is 10, the PHP array retrieves the integer 10.
- The protocolToRawData function parses downstream data sent by IoT Platform and returns an array. The array must be a common PHP array. Each element in the array must be an integer within a value range of [0,255].
- The transformPayload function parses data that is sent to custom topics. The input parameter of the function is an integer array. Use `0xFF` for the bitwise AND operation to obtain the complement. The returned result is an association array. Each key value must contain non-array characters. For example, if a key value is 10, the PHP array retrieves the integer 10.
- The PHP runtime environment is significantly strict with exception handling. If an error occurs, the subsequent logic is not implemented. To ensure code robustness, you must capture and process errors.

Sample script

The following sample script is written based on the properties and protocol that are defined in the Example for parsing TSL data topic.

For more information about the data types supported by TSL models, see Supported data types. After a device submits TSL property data or event data, IoT Platform generates a response whose data is in the Alink JSON format. Before IoT Platform sends the response to the device, IoT Platform uses the script to convert the format of the data to a format that can be recognized by the device. For more information about the Alink JSON format, see Device properties, events, and services.

```
<? php
/*
Sample data
Submit device data
Input parameters:
    0x0000000001003201
Output result:
    {"method":"thing.event.property.post","id":"1","params":{"prop_int16":50,"prop_bool":1}
,"version":"1.0"}
Response after property configurations
Input parameters:
    0x0300223344c8
Output result:
    {"code":"200","id":"2241348","version":"1.0"}
*/
function rawDataToProtocol($bytes)
{
    $data = [];
    $length = count($bytes);
    for ($i = 0; $i < $length; $i++) {
        $data[$i] = $bytes[$i] & 0xff;
    }
```

```
    $jsonMap = [];
    $fHead = $data[0]; //The command field.
    if ($fHead == 0x00) {
        $jsonMap['method'] = 'thing.event.property.post '; //The topic that is used to subm
it properties. Data format: ALink JSON.
        jsonMap['version'] = '1.0'; //The version of the protocol. This field is fixed. Dat
a format: ALink JSON.
        $jsonMap['id'] = '' . getInt32($data, 1); //The ID of the request. Data format: ALi
nk JSON.
        $params = [];
        $params['prop_int16'] = getInt16($data, 5); //The value of the prop_int16 property
of the product.
        params['prop_bool'] = $data[7]; //The value of the prop_bool property of the produc
t.
        jsonMap['params'] = params; //The params field. Data format: ALink JSON.
    } else if ($fHead == 0x03) {
        jsonMap['version'] = '1.0'; //The version of the protocol. This field is fixed. Dat
a format: ALink JSON.
        $jsonMap['id'] = '' . getInt32($data, 1); //The ID of the request. Data format: ALi
nk JSON.
        $jsonMap['code'] = getInt8($data, 5);
    }
    return $jsonMap;
}
/*
Sample data
Configure properties
Input parameters:
    {"method":"thing.service.property.set","id":"12345","version":"1.0","params":{"prop_int
16":333, "prop_bool":1}}
Output result:
    0x013039014d01
Response after data submission
Input data:
    {"method":"thing.event.property.post","id":"12345","version":"1.0","code":200,"data":{}
}
Output result:
    0x023039c8
*/
function protocolToRawData($json)
{
    $method = $json['method'];
    $id = $json['id'];
    $version = $json['version'];
    $payloadArray = [];
    if ($method == 'thing.service.property.set ') //Configure properties.
    {
        $params = $json['params'];
        $prop_int16 = $params['prop_int16'];
        $prop_bool = $params['prop_bool'];
        //Raw data is concatenated based on the custom protocol format.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0x01)));//The comm
and field.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id)))); //
```

```
The ID of the request. Data format: ALink JSON.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($prop_int16))); //
The value of the prop_int16 property.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($prop_bool))); //T
he value of the prop_bool property.
    } else if ($method == 'thing.event.property.post ') { //The response after data submiss
ion.
        $code = $json['code'];
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0x02))); //The com
mand field.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id)))); //
The ID of the request. Data format: ALink JSON.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($code)));
    } Else {//Unknown commands. Certain commands are not processed.
        $code = $json['code'];
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(0xff))); //The com
mand field.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex(intval($id)))); //
The ID of the request. Data format: ALink JSON.
        $payloadArray = concat($payloadArray, hexStringToByteArray(toHex($code)));
    }
    return $payloadArray;
}
/*
  Sample data
  Custom Topic for reporting data: /user/update
  Input parameters:
     topic: /{productKey}/{deviceName}/user/update
     bytes: 0x00000000010032010000000
  Output parameters:
  {
     "prop_float": 0,
     "prop_int16": 50,
     "prop_bool": 1,
     "topic": "/{productKey}/{deviceName}/user/update"
   }
 */
function transformPayload($topic, $bytes)
{
    $data = array();
    $length = count($bytes);
    for ($i = 0; $i < $length; $i++) {
        $data[$i] = $bytes[$i] & 0xff;
    }
    $jsonMap = array();
    if (strpos($topic, '/user/update/error') ! == false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['errorCode'] = getInt8($data, 0);
    } else if (strpos($topic, '/user/update') ! == false) {
        $jsonMap['topic'] = $topic;
        $jsonMap['prop_int16'] = getInt16($data, 5);
        $jsonMap['prop_bool'] = $data[7];
    }
    return $jsonMap;
```

```
}
function getInt32($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1], $bytes[$index + 2], $bytes[$index +
3]);
    return hexdec(byteArrayToHexString($array));
}
function getInt16($bytes, $index)
{
    $array = array($bytes[$index], $bytes[$index + 1]);
    return hexdec(byteArrayToHexString($array));
}
function getInt8($bytes, $index)
{
    $array = array($bytes[$index]);
    return hexdec(byteArrayToHexString($array));
}
function byteArrayToHexString($data)
{
    $hexStr = '';
    for ($i = 0; $i < count($data); $i++) {
        $hexValue = dechex($data[$i]);
        $tempHexStr = strval($hexValue);
        if (strlen($tempHexStr) === 1) {
            $hexStr = $hexStr . '0' . $tempHexStr;
        } else {
            $hexStr = $hexStr . $tempHexStr;
        }
    }
    return $hexStr;
}
function hexStringToByteArray($hex)
{
    $result = array();
    $index = 0;
    for ($i = 0; $i < strlen($hex) - 1; $i += 2) {
        $result[$index++] = hexdec($hex[$i] . $hex[$i + 1]);
    }
    return $result;
}
function concat($array, $data)
{
    return array_merge($array, $data);
}
function toHex($data)
{
    $var = dechex($data);
    $length = strlen($var);
    if ($length % 2 == 1) {
        $var = '0' . $var;
    }
    return $var;
}
```

2.4. Troubleshooting

This article describes how to debug a data parsing script on premises, and troubleshoot problems if IoT Platform fails to parse data by using the script.

Debug a script on premises

You can check whether a data parsing script is valid in IoT Platform. However, you cannot debug the script in IoT Platform. We recommend that you write and debug a script on premises, and then copy the script to the online editor in the IoT Platform console.

The following sample script is used for on-premises debugging. For more information about sample scripts, see Example for parsing TSL data. Specify the parameters as required when you use the script.

```
//Test Demo
function Test()
{
    //0x001232013fa00000
    var rawdata_report_prop = new Buffer([
        0x00, //The fixed command header. 0 indicates property data submission.
        0x00, 0x22, 0x33, 0x44, //The ID of the request.
        0x12, 0x32, //The value of the prop_int16 parameter. This value is a two-byte Int16
value.
        0x01, //The value of the prop_bool parameter. This value is a one-byte Boolean valu
e.
        0x3f, 0xa0, 0x00, 0x00 //The value of the prop_float parameter. This value is a fou
r-byte Float value.
    ]);
    rawDataToProtocol(rawdata_report_prop);
    var setString = new String('{"method":"thing.service.property.set","id":"12345","versio
n":"1.0","params":{"prop_float":123.452, "prop_int16":333, "prop_bool":1}}');
    protocolToRawData(JSON.parse(setString));
}
Test();
```

Troubleshoot online parsing problems

After a device is connected to IoT Platform and submits property data, you can view the data in the IoT Platform console if a data parsing script works properly. Find the required device in the device list and click **View** to go to the **Device Details** page. On this page, choose **TSL Data > Status**.

If the device has submitted data but the data is not displayed, choose **Maintenance > Device Log > Cloud run log** to view the logs and troubleshoot the problem.

You can perform the following steps to troubleshoot the problem:

1. Select a product, enter the DeviceName, and then click **Search**. Select *TSL Report* from the Workload Type drop-down list to query logs.
2. View the log entry that include payloads and parsed data.
3. Check the error code. For more information about error codes, see IoT Platform logs.
4. Check the script and submitted device data based on the error code.

The following errors may occur:

- The script does not exist.

  The error code 6200 is displayed in the log entry. For more information about the description of error codes, see IoT Platform logs. The error code 6200 indicates that the script does not exist. Check whether the script has been submitted to the console.

- The Alink method does not exist.

  The error code 6450 is displayed in the log entry. This error code indicates that the method parameter in the Alink data does not exist. For more information, see IoT Platform logs. The method parameter is missing after the custom formatted data or pass-through data is parsed to the standard Alink data.

  The following log entry shows the error message:

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

  The error message `alink method not exist` indicates that the method parameter does not exist in the Alink data. This error occurs if the definition of the method parameter is invalid in the data parsing script. You must modify the script.

3.Tags

You can use tags to manage products, devices, or device groups based on your business requirements. This article describes how to add tags.

Prerequisites

A product, a device, and a device group are created. For more information, see the following articles:

- Create a product
- Create a device
- Create a group

Context

In IoT scenarios, you need to manage a large number of products and devices. Alibaba Cloud IoT Platform provides tags to identify various products and devices and achieve centralized management. You can attach different tags to products, devices, or device groups and manage these resources based on the tags.

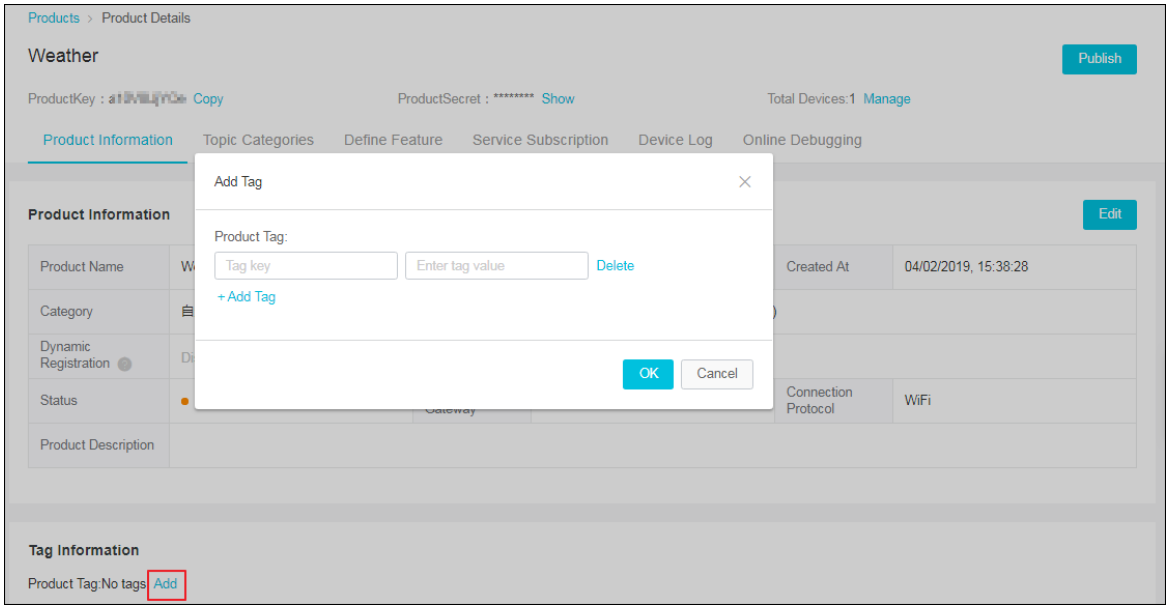
Usage notes

- The format of a tag is `Key:Value`.
- Each product, device, or device group can have up to 100 tags.

Add product tags

A product tag describes the information that is common to all devices of a product, such as the manufacturer, organization, physical size, and operating system of the product.

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Products**.
4. On the **Products** page, find the product to which you want to add a tag and click **View** in the Actions column.
5. Click **Edit** next to **Tag Information**.



6. In the **Edit Tag** dialog box, set the Tag key and Tag value parameters. Then, click **OK**.

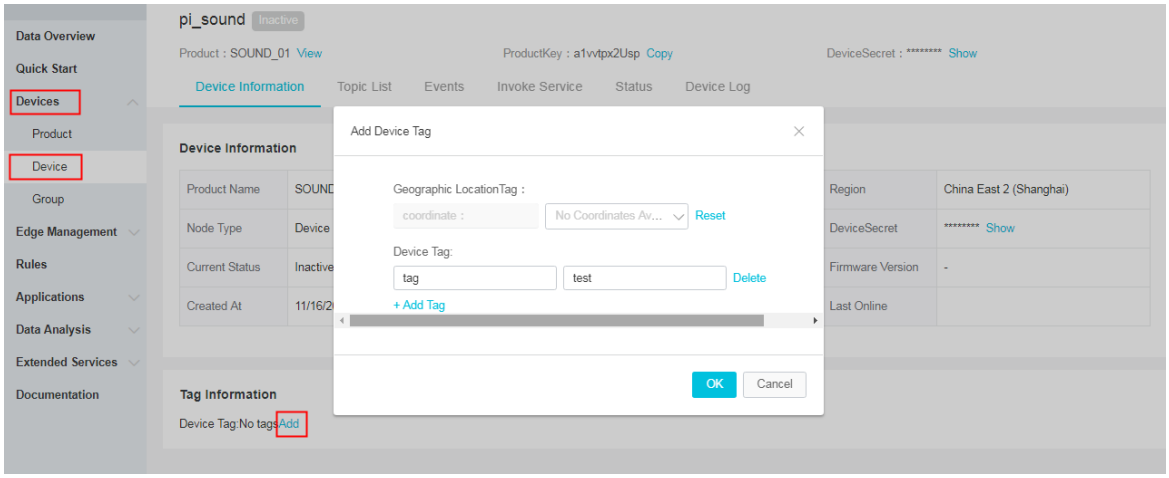
Parameter	Description
Tag key	The key of the tag. The key must be 1 to 30 characters in length and can contain letters, digits, and periods (.).
Tag value	The value of the tag. The value must be 1 to 128 characters in length and can contain letters, digits, underscores (_), hyphens (-), colons (:), and periods (.).

Add device tags

You can add unique tags to devices based on device features. This allows you to manage your devices with ease. For example, you can add the `room:201` tag to a smart meter in Room 201.

- You can forward device tags in IoT Platform. For more information about data formats, see Device tags.
- You can also add a device tag to the body of a message that is sent by the device. Then, you can send the message to other Alibaba Cloud services by using the data forwarding feature of the rules engine. For more information about data formats, see Submit device tag changes.

1. In the left-side navigation pane, choose **Devices > Devices**.
2. On the **Devices** page, find the device to which you want to add a tag and click **View** in the Actions column.
3. On the **Device Details** page, click **Edit** next to **Tag Information**.



4. In the **Edit Tag** dialog box, click **Add Tag** and set the Tag key and Tag value parameters.

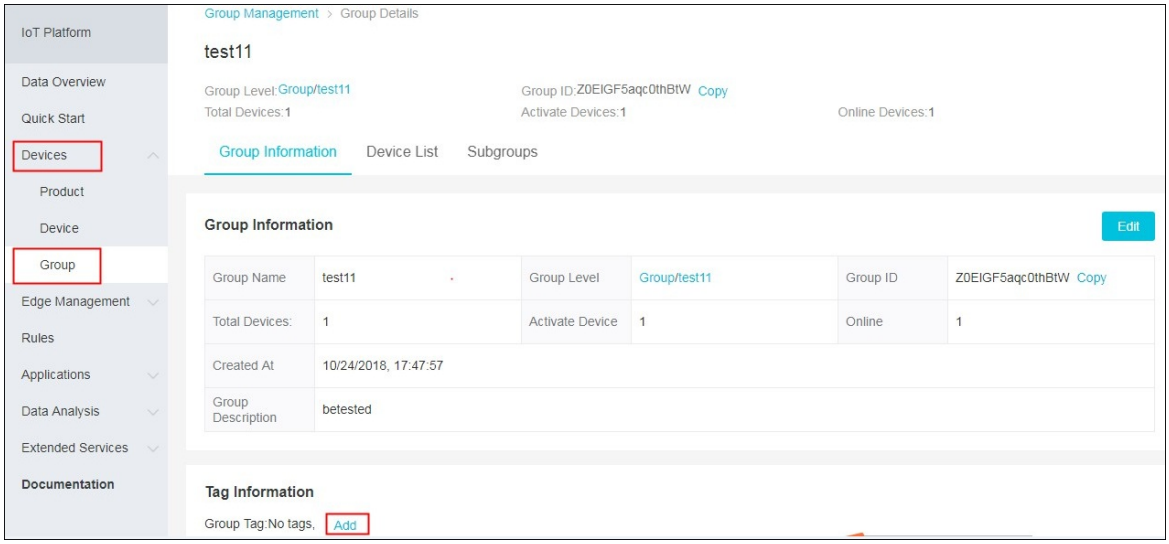
Parameter	Description
Tag key	The key of the tag. The key must be 1 to 30 characters in length and can contain letters, digits, forward slashes (/), underscores (_), hyphens (-), number signs (#), at signs (@), percent signs (%), ampersands (&), asterisks (*), and periods (.).
Tag value	The value of the tag. The value must be 1 to 128 characters in length and can contain letters, digits, Japanese characters, underscores (_), hyphens (-), number signs (#), at signs (@), percent signs (%), ampersands (&), colons (:), and periods (.).

5. Click **OK**.

Add group tags

Device groups are used to manage devices across products. A group tag describes the information that is common to all devices and subgroups of a device group, such as the region and location where the devices reside.

1. In the left-side navigation pane, choose **Devices > Groups**.
2. On the **Groups** page, find the device group to which you want to add a tag and click **View** in the Actions column.
3. Click **Edit** next to **Tag Information**.



4. In the **Edit Tag** dialog box, set the Tag key and Tag value parameters. Then, click **OK**.

Parameter	Description
Tag key	The key of the tag. The key must be 1 to 30 characters in length and can contain letters, digits, and periods (.).
Tag value	The value of the tag. The value must be 1 to 128 characters in length and can contain letters, digits, underscores (_), hyphens (-), colons (:), and periods (.).

Manage multiple tags at a time

You can create, edit, and delete tags in the IoT Platform console. You can also call the API operations provided by IoT Platform to manage tags or query products, devices, and groups based on tags. For more information, see List of operations by function.

4.Advanced search

If you need to search for and download a device list, including ProductKey and DeviceName, based on a specified condition in IoT Platform, you can use the advanced search feature. This feature allows you to use SQL-like statements to search for devices, such as online devices. This article describes how to perform an advanced search and the SQL-like syntax.

Limits

- This feature is available in the Japan (Tokyo) regions.
- Limits on features: Advanced search is not supported on the **Devices > Devices** page of the public instance.

Scenarios

Advanced search is supported in the following scenarios in the IoT Platform console:

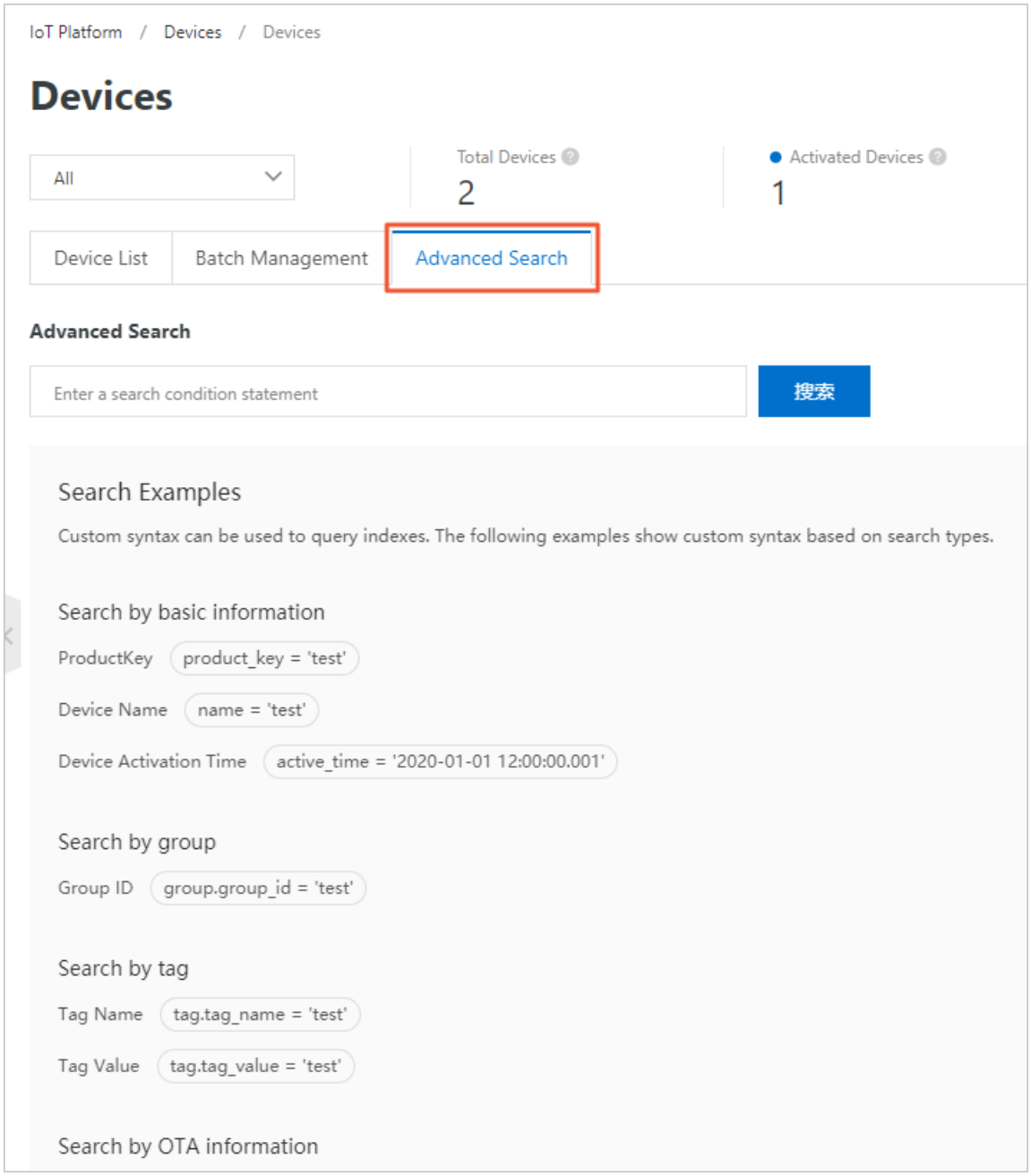
- Manage devices: On the **Advanced Search** tab of the **Devices > Devices** page, search for specified devices.

You can also call the QueryDeviceBySQL operation to perform an advanced search. API operation-based advanced search is not limited to the preceding scenarios.

Procedure

This article describes how to use the advanced search feature to add devices to a device group and search for devices based on TSL models.

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Devices** . Click the **Advanced Search** tab.
4. On the **Advanced Search** tab, use the advanced search feature to search for devices.



SQL-like syntax

A SQL-like statement consists of the WHERE clause and the ORDER BY clause (optional). The SELECT clause, the LIMIT clause, and the `WHERE` keyword of the WHERE clause are omitted. Each statement cannot exceed 400 characters in length.

Example:

```
product_key = "a1*****" order by active_time
```

WHERE clause

Syntax:

```
[condition1] AND [condition2]
```

The `WHERE` keyword is omitted.

You can specify a maximum of five conditions. Nesting is unsupported. For more information, see the search fields and operators in the following tables.

You can use the AND and OR logical operators. A maximum of five operators can be used.

ORDER BY clause (optional)

The ORDER BY clause is used for sorting. The fields that can be sorted include gmt_create, gmt_modified, and active_time.

This clause can be left unspecified. In this case, the results are randomly sorted.

Search fields

Field	Type	Description
product_key	text	The ProductKey of the product to which the device belongs.
iot_id	text	The ID of the device. By default, iot_id is returned.
name	text	The DeviceName of the device.
active_time	date	The time when the device was activated. The time uses the yyyy-MM-dd HH:mm:ss.SSS format, accurate to milliseconds.
nickname	text	The alias of the device.
gmt_create	date	The time when the device was created. The time uses the yyyy-MM-dd HH:mm:ss.SSS format, accurate to milliseconds.
gmt_modified	date	The time when the device information was last updated. The time uses the yyyy-MM-dd HH:mm:ss.SSS format, accurate to milliseconds.
status	text	The status of the device. Valid values: • ONLINE: The device is online. • OFFLINE: The device is offline. • UNACTIVE: The device is not activated. • DISABLE: The certificate is disabled.
group.group_id	text	The ID of the device group.
tag.tag_name	text	The tag key of the device.
tag.tag_value	text	The value of the device tag.
ota_module.name	text	The name of the OTA module. We recommend that you use this field with ota_module.version to specify the OTA module corresponding to the current OTA version number of the device.

Field	Type	Description
ota_module.version	text	The current OTA version number of the device. We recommend that you use this field with ota_module.name.

Operators

Operator	Supported data type
=	number, date, text, and keyword
>	number and date
<	number and date
LIKE	text

`LIKE` supports prefix matching, but does not support suffix matching or wildcard matching. A prefix must meet the following requirements:

- The prefix must be at least 4 characters and cannot contain any special characters, such as backslashes (\), forward slashes (/), ampersands (&), plus signs (+), hyphens (-), exclamation points (!), parentheses (), colons (:), tildes (~), brackets [], braces {}, asterisks (*), and question marks (?).
- The prefix must end with a percent sign (`%`).

Example:

```
product_key = "a1*********" and name LIKE "test%"
```

5.Device groups

IoT Platform allows you to group devices. You can use device groups to manage devices across products. This article describes how to create and manage device groups in the IoT Platform console.

Context

- Each group can have up to 100 subgroups.
- Groups can be nested in only three layers, which are groups, subgroups, and child subgroups.
- Each subgroup belongs to only one group.
- You can create or delete the nested relationship of groups, but cannot modify the nested relationship.
- You cannot delete a group that has one or more subgroups. To delete a group, you must first delete all of its subgroups.
- You can search for groups and subgroups by name. Fuzzy search is supported.

Procedure

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Groups**. The Groups page appears.
4. Click **Create Group**. In the Create Group dialog box, set the parameters and click **OK**.

   > **Note** Each Alibaba Cloud account can create up to 1,000 groups and subgroups.

Parameter	Description
Parent Group	The type of the group to be created. ◦ If you select Groups, the group to be created is a parent group. ◦ If you select a specific group, the group to be created is a subgroup of the selected group.
Group Name	In the Add Whitelist dialog box, set the Whitelist Name parameter. The name must be 4 to 30 characters in length, and can contain letters, digits, and underscores (_). The group name must be unique within your Alibaba Cloud account and cannot be modified after it is created.
Description	The description of the group. This parameter is optional.

5. Optional. On the **Device Details** page, add a tag to the group. A tag is a custom group identifier. This allows you to flexibly manage the group. For more information, see Add group tags.
6. On the **Group Details** page, click the **Device List** tab, and then click **Add Device to Group**. In the **Add Device to Group** panel, search for a device, select the device, and then click **OK**.

> **Note**
>
> - You can add up to 1,000 devices at a time. You can add up to 20,000 devices to each group.
> - Each device can be added to up to 10 groups.

In the **Add Device to Group** panel, you can click **All** or **Selected** in the upper-right corner.

- If you click **All**, all devices are displayed in the list.
- If you click **Selected**, only selected devices are displayed in the list.

After a device is added, you can search for the device on the **Device List** tab.

- If you select **All**, the tab displays all the devices in the group that belong to all the products within your Alibaba Cloud account. Fuzzy search is supported. You can enter a keyword to search for devices. For example, if you enter the keyword *test*, the search results may be the devices named test1, test2, and test3 within your Alibaba Cloud account.
- If you select a specific product, the tab displays all the devices that belong to the selected product. Fuzzy search is supported. You can enter a keyword to search for devices.

7. Optional. On the **Group Details** page, choose **Subgroups > Create Group** to create a subgroup.

   Subgroups are used for finer-grained device management. For example, for a group named Smart_home, you can create subgroups such as Smart_kitchen and Smart_bedroom. This helps you separately manage kitchen equipment and bedroom equipment. To configure a subgroup, perform the following steps:

   i. In the Create Group dialog box, select a parent group, enter a name and a description for the subgroup, and then click **OK**.
   ii. On the **Subgroups** tab, find the subgroup that you created and click **View** in the Actions column.
   iii. On the **Group Details** page of the subgroup, click the **Device List** tab. Then, click **Add Device to Group** and add devices to the subgroup.

   After you create the subgroup and add devices to the subgroup, you can manage the subgroup and the devices in it. You can also create child subgroups for the subgroup.

6.Device shadows

6.1. Overview

IoT Platform provides device shadows to cache device status. Online devices can receive commands from IoT Platform. If devices are offline, you can request commands from IoT Platform after the devices go online.

What is a device shadow?

A device shadow is a JSON file that is used to store the status submitted by a device and the desired device status obtained from applications.

Each device has only one shadow. A device can obtain or set the shadow in IoT Platform over Message Queuing Telemetry Transport (MQTT) to synchronize the status.

Scenarios

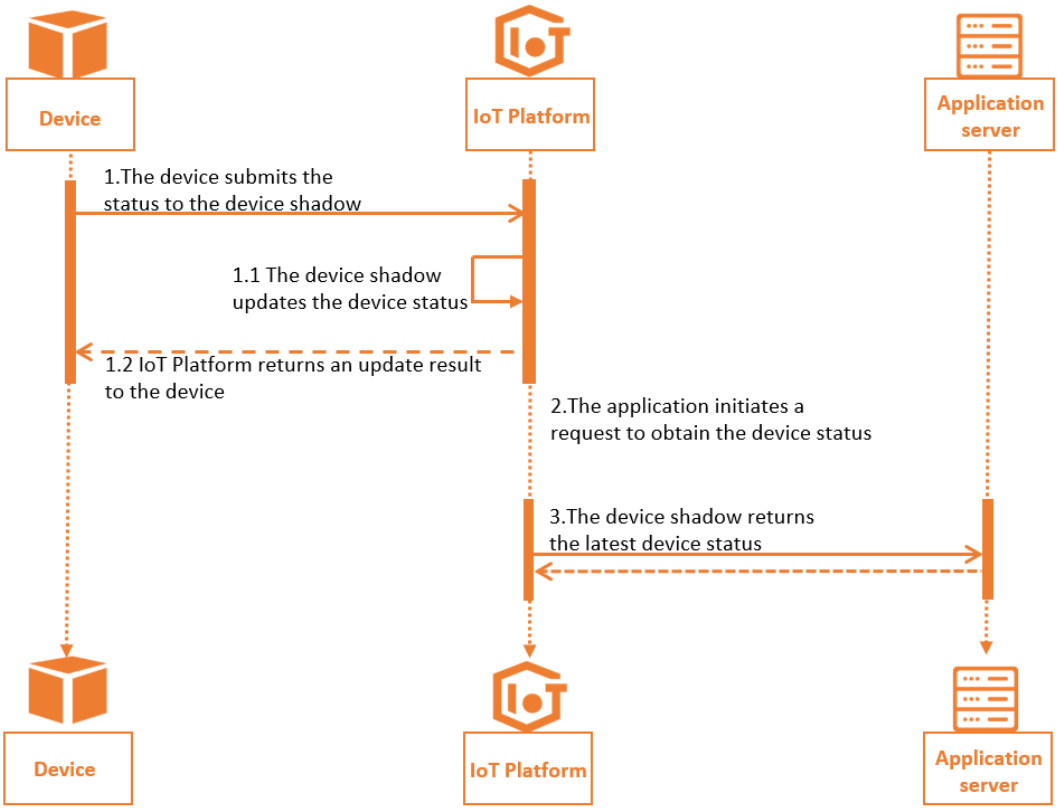
- Applications request the device status.

  Description:

  - A device frequently goes online and offline due to network instability. In this case, the device cannot process the requests from applications.
  - Multiple applications may request the device status at the same time. The device cannot process the concurrent requests and return responses to the applications.

  You can synchronize the status from the device to the shadow in IoT Platform. Applications can obtain the latest status from the shadow. This way, the applications are decoupled from the device.

  The following figure shows the process. For more information, see The device submits the status.

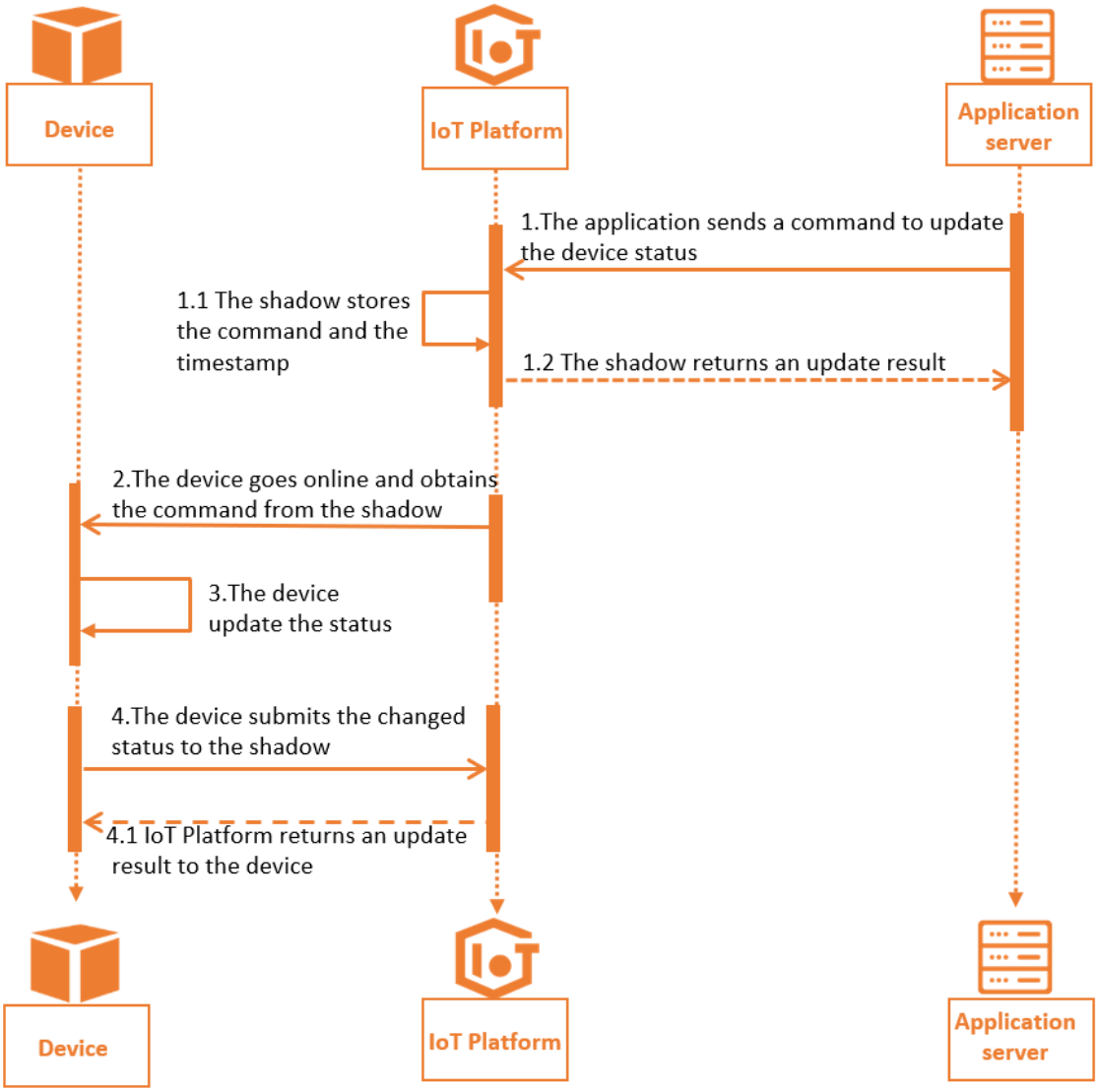


- Applications send commands to change the device status.

  Description: A device is offline, or the device frequently goes online and offline due to network instability. In this case, the commands from applications may fail to be sent to the device.

  You can store the commands with timestamps in the shadow. After the device goes online, the device obtains the commands from the shadow and determine whether to process the commands based on timestamps.

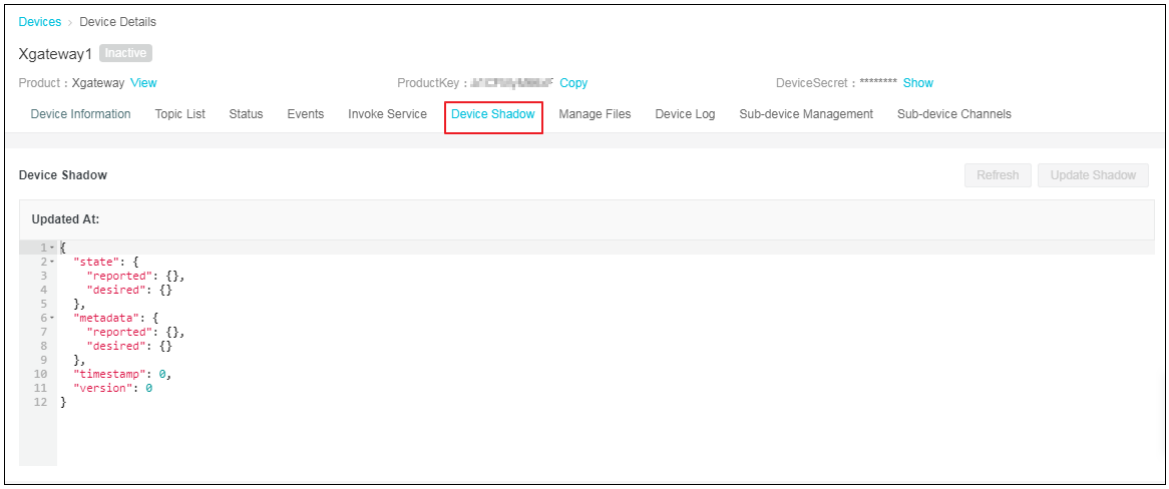
  The following figure shows the process. For more information, see Applications change the device status, The device requests the shadow, and The device deletes shadow properties.



View and update a device shadow

After you develop the device and connected devices, you can view and update the device shadow status in the IoT Platform console.

1. Log on to the IoT Platform console.
2.
3. In the left-side navigation pane, choose **Devices > Devices**.
4. On the Devices page, find the device that you want to manage and click **View** in the Actions column. The **Device Details** page appears.
5. Click the **Device Shadow** tab. On this tab, you can view the device shadow submitted by the device.



6. Click **Update Shadow**. Enter the desired status information in the `"desired"` section.

   For information about the format of device shadows, see JSON format of device shadows.

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

   If the device is offline, the shadow caches the desired status. After the device goes online, the device pulls the latest desired status from IoT Platform.

Related API operations

GetDeviceShadow: queries a device shadow.

UpdateDeviceShadow: updates a device shadow.

6.2. JSON format of device shadows

This article describes the JSON format of device shadows.

The following code snippet shows the sample JSON file of a device shadow:

```
{
    "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
}
```

The JSON file properties table describes the properties in the sample JSON file.

JSON file properties

Property	Description
desired	The desired status of the device. The JSON file of a device shadow contains the desired property only when you have specified the desired status. An application can directly write the desired property of the device without the need to connect to the device.
reported	The status that is reported by the device. The device can write the reported property to report its latest status. An application can obtain the status of the device by reading this property. The JSON file of a device shadow may not contain the reported property and is still valid.

Property	Description
metadata	The metadata information about the device status, which is automatically updated based on updates in the JSON file. The metadata information contains the timestamp of each property to provide accurate update time. Each timestamp is the number of seconds that have elapsed since January 1, 1970, 00:00:00 UTC.
timestamp	The latest time when the JSON file is updated.
version	The version number of the JSON file. When you request a version update 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 version, the device shadow is updated and the version property is also updated. Otherwise, the device shadow rejects the request. During updates, the value of the version property increments, which ensures that the file being updated is always the latest version. The version property is of the LONG type. To prevent a parameter overflow, you can enter `-1` to reset the version number to `0`.

Note

Arrays are supported in the JSON files of device shadows. When you update an array, you must update the whole array instead of only a part of the array.

The following example shows how to update an array:

- Initial status:

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

- Update:

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

- Final status:

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

7.NTP service

IoT Platform provides the Network Time Protocol (NTP) service for embedded devices that have limited resources. This allows the devices to obtain accurate clocks in real time.

How it works

The NTP service is based on the NTP protocol. IoT platform is used as the NTP server. The following steps describe the process of high-precision time calibration.

1. A device sends a message to IoT Platform by using a specified topic. The message includes the deviceSendTime parameter.
2. After IoT Platform receives the message, IoT Platform returns a response. The response includes the serverRecvTime and serverSendTime parameters.
3. When the device receives the response, the device generates the deviceRecvTime parameters based on the local time.
4. The device calculates the time difference between the device and IoT Platform and obtain the Time parameter. This parameter indicates the current time of the server.

> **Notice** The time of a device can be calibrated by using the NTP service only when the device is connected to IoT Platform.
>
> Otherwise, the device cannot obtain an accurate time after it is powered on. When the device establishes a TSL connection with IoT Platform, the certificate time fails to be verified.

Topics

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

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

${YourProductKey}* and *${YourDeviceName} specifies the ProductKey and DeviceName in the device certificate. You can obtain the device certificate on the **Device Details** page of the IoT Platform console .

Device connection

You can configure the NTP service only by using Link SDK for C. To download the sample code, see Obtain and use the SDK.

The following steps describe how to use the NTP service by using specified topics:

1. The device subscribes to the `/ext/ntp/${YourProductKey}/${YourDeviceName}/response` topic.
2. The device sends a QoS 0 message to the `/ext/ntp/${YourProductKey}/${YourDeviceName}/request` topic. The message includes a timestamp that indicates the current time of the device. The unit of the timestamp is milliseconds. The following code shows an example:

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

> **Note**
> - The data type of the timestamp can be Long or String. The Long data type is used be default.
> - The NTP service supports only QoS 0 messages.

3. The device receives a response from IoT Platform by using the `/ext/ntp/${YourProductKey}/${YourDeviceName}/response` topic. The following code shows an example:

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

4. The device calculates the accurate current time in the UNIX timestamp format.

   The device marks the time when it receives the response from the NTP server as ${deviceRecvTime}. The accurate current time of the device is calculated by using the following formula: `(${serverRecvTime} + ${serverSendTime} + ${deviceRecvTime} - ${deviceSendTime})/2`.

Example

> **Note** The timestamps that are sent by the device and the NTP server are of the same data type. For example, if the timestamp that is sent by the device is of the STRING type, the timestamp that is returned by the NTP server is also of the STRING type.

In this example, the timestamp of the device is 1571724098000. The timestamp of the NTP server is 1571724098100. The link latency is 10 milliseconds. The NTP server sends a response 5 milliseconds after it receives a message from the device.

Operation	Device time (milliseconds)	Server time (milliseconds)
The device sends a message	1571724098000 (deviceSendTime)	1571724098100
The NTP server receives the message	1571724098010	1571724098110 (serverRecvTime)
The NTP server sends a response	1571724098015	1571724098115 (serverSendTime)
The device receives the response	1571724098025 (deviceRecvTime)	1571724098125

The device calculates the accurate current time by using the following formula: `(1571724098110 + 1571724098115 + 1571724098025 - 1571724098000)/2 = 1571724098125`.

If the device uses the timestamp that is returned by IoT Platform 1571724098115, a link latency of 10 milliseconds between the device and IoT Platform is not calculated.

8.Error codes for devices

This topic describes the error codes that IoT Platform may return to devices.

Common error codes

General-purpose error codes

Error code	Cause	Solution
400	An error occurred when the system processed the request.	Submit a ticket.
429	Traffic throttling is triggered due to frequent requests.	Submit a ticket.
460	The data submitted by the device is empty, the format of the parameter is invalid, or the number of parameters exceeds the limit.	Check the parameters based on the data formats that are described in the Alink protocol topic.
500	An unknown error occurred in IoT Platform.	Submit a ticket.
5005	An error occurred when the system queried the product information.	Check the product information in the console and make sure that the ProductKey is valid.
5244	An error occurred when the system queried the metadata of the LoRaWAN-based product.	Submit a ticket.
6100	An error occurred when the system queried the information about the specified device.	Check the device information on the Devices page of the IoT Platform console.
6203	An error occurred when the system parsed the topic.	Submit a ticket.
6250	An error occurred when the system queried the product information.	Check the product information in the console and make sure that the ProductKey is valid.
6204	The specified device is disabled. You cannot perform operations on this device.	Check the device status on the Devices page of the IoT Platform console.
6450	The method parameter is missing after the custom data is parsed to the standard Alink format.	On the **Device Log** page of the console or in the on-premises log file of the device, check whether the data submitted by the device contains the method parameter.

Error code	Cause	Solution
6760	The error message returned because a system exception occurred.	Submit a ticket.

Error codes for parsing scripts

Error code	Cause	Troubleshooting
26001	The system failed to find a parsing script.	Go to the **Data Parsing** tab in the IoT Platform console and make sure that the script is submitted. **Note** You cannot run scripts that are not submitted.
26002	The script runs as expected, but the script content is invalid. For example, the script contains syntax errors.	Use the same data to test the script. Check the error message and modify the script. We recommend that you test the script on an on-premises device before you submit the script to IoT Platform.
26006	The script runs as expected. However, the script content is invalid. The script must contain the protocolToRawData and rawDataToProtocol methods. If the methods are missing, an error occurs.	Go to the **Data Parsing** tab in the console and check whether the protocolToRawData and rawDataToProtocol methods exist.
26007	The script runs as expected, 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 rawDataToProtocol method must return a JSON object. If the response is not in the required format, an error occurs.	Test the script in the console or on an on-premises device and check whether the format of the response is valid.
26010	Traffic throttling is triggered due to frequent requests.	Submit a ticket.

Error codes for Thing Specification Language (TSL) models

Status code	Cause	Troubleshooting
5159	The system failed to query the defined properties when the system verified parameters based on the TSL model.	Submit a ticket.

Status code	Cause	Troubleshooting
5160	The system failed to query the defined events when the system verified parameters based on the TSL model.	Submit a ticket.
5161	The system failed to query the defined services when the system verified parameters based on the TSL model.	Submit a ticket.
6207	The Alink data submitted 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 the Device properties, events, and services topic when you submit data.
6300	The method parameter that is required by the Alink protocol does not exist in the standard Alink data or in the parsed custom data that is submitted by the device.	On the **Device Log** page of the console, or in the on-premises log file of the device, check whether the data submitted by the device contains the method parameter.
6301	If the system verifies parameters based on the TSL model, the data type is specified as an array. However, the type of data submitted by the device is not an array.	Go to the **Define Feature** tab in the console, and check the data type defined in the TSL model. Submit data based on the required data type.
6302	The required input parameters of the service are not specified.	Log on to the console and check the TSL model. Make sure that the required input parameters are specified.
6306	If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The parameter value is not in the defined range.	
6307	The input parameters do not comply with the 32-bit float data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The parameter value is not in the defined range.	

Status code	Cause	Troubleshooting
6308	The input parameters do not comply with the Boolean data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The parameter value is not in the defined range.	Log on to the console and check the TSL model. Make sure that the data types of the input parameters are the same as the data types that are defined in the TSL model, and the parameter values are within the value range that is defined in the TSL model.
6310	The input parameters do not comply with the text data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The length of the parameter exceeds the limit that is defined in the TSL model.	
6322	The input parameters do not comply with the 64-bit float data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The parameter value is not in the defined range.	
6304	The input parameters do not exist in the structure defined in the TSL model.	
6309	The input parameters do not comply with the enumeration data specifications defined in the TSL model.	

Status code	Cause	Troubleshooting
6311	The input parameters do not comply with the date data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The input data is not a UTC timestamp.	Log on to the console and check the TSL model. Make sure that the data types of the input parameters are valid.
6312	The input parameters do not comply with the structure data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The data types of the input parameters are different from the data types that are defined in the TSL model. • The number of parameters that are contained in the structure is different from the number of parameters that are defined in the TSL model.	
6320	The specified property cannot be found when the system queries the TSL data 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 of the property, event, or service is not specified.	Submit a ticket.
6317	The parameters required in the TSL model are not configured, such as the type and specs parameters.	Submit a ticket.
6324	The input parameters do not comply with the array data specifications defined in the TSL model. If the system verifies parameters based on the TSL model, the following errors may occur: • The elements in the array do not follow the array syntax that is defined in the TSL model. • The number of elements in the array exceeds the maximum number that is defined in the TSL model.	• Go to the Product Details page of the IoT Platform console. On the Define Feature tab, check the array syntax that is defined in the TSL model. • View the upstream message logs to check the number of array elements in the data that is submitted by the device.

Status code	Cause	Troubleshooting
6325	The data types of the elements in the array are not supported by IoT Platform. Only the following data types of elements are supported: integer, 32-bit float, 64-bit float, double, text, and structure.	Check whether the data types of the elements are supported by IoT Platform.
6326	The format of the time field submitted by the device is invalid.	Follow the data formats that are described in the Device properties, events, and services topic when you submit data.
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	A system error occurred when the system parsed the TSL model.	Submit a ticket.
6313		
6329		
6323		
6316		
6314		
6301		

Error codes for device registration

- Register directly connected devices
  - Request topic: `/sys/${productKey}/${deviceName}/thing/sub/register`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/sub/register_reply`

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

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

Error code	Cause	Troubleshooting
6288	Dynamic registration is disabled for the device.	Log on to the console and enable dynamic registration on the Product Details page.

Error code	Cause	Troubleshooting
6619	The device is bound to another gateway.	Go to the Device Details tab in the IoT Platform console and check whether the device is bound to a gateway.

- Register directly connected devices based on unique-certificate-per-product verification

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

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

Error code	Cause	Troubleshooting
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 that is used to calculate the signature is not supported by IoT Platform.	Use an algorithm that is supported by the signMethod parameter. For more information, see Register devices.
6600	An error occurred when the system parsed the signature.	Use a supported algorithm to calculate a signature and then verify the signature. For more information, see Register devices.
6289	The device is activated.	Log on to the console and check the status of the device.

Error codes for topological relationships

- Add topological relationships
  - Request topic: `/sys/${productKey}/${deviceName}/thing/topo/add`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/topo/add_reply`

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

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

Error code	Cause	Troubleshooting
401	The system failed to verify the signature when you add the topological relationship.	Use a supported algorithm to calculate a signature and then verify the signature. For more information, see Manage topological relationships.

Error code	Cause	Troubleshooting
6402	The gateway and sub-device are the same device. When you add a topological relationship, you cannot add the current gateway to as its sub-device.	View the information about all existing sub-devices, and check whether the gateway shares the same information with a sub-device.
6400	The number of sub-devices that you add to the gateway exceeds the limit.	Log on to the console and check the number of existing sub-devices of the gateway on the **Sub-device Management** tab. For more information about the limits, see Limits.

- Delete topological relationships
  - Request topic: `/sys/${productKey}/${deviceName}/thing/topo/delete`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/topo/delete_reply`

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

  The following table describes the cause and solution of the error that may occur when you delete a topological relationship between a gateway and a sub-device. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting
6401	The topological relationship does not exist.	Log on to the console and click **Devices** in the left-side navigation pane. Then, click the **Sub-device Management** tab on the **Device Details** page of the gateway. You can view the information about the sub-device.

- Obtain topological relationships
  - Request topic: `/sys/${productKey}/${deviceName}/thing/topo/get`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/topo/get_reply`

  Error codes: 460, 429, 500, and 6203

  For more information about these error codes, see the "Common error codes" section of this topic.
- A gateway submits the information about a detected sub-device
  - Request topic: `/sys/${productKey}/${deviceName}/thing/list/found`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/list/found_reply`

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

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

Error code	Cause	Troubleshooting
6280	The sub-device name submitted by the gateway is invalid. The name must be 4 to 32 characters in length and can contain letters, digits, and underscores (_).	Check whether the sub-device name submitted by the gateway is valid.

Error codes for sub-device connections and disconnections

- A sub-device goes online

  Topic to which error messages are sent: `/ext/session/{productKey}/{deviceName}/combine/logout_reply`

  Error codes: 460, 429, 6100, 6204, 6287, 6401, 500, 9241, and 9240

- A sub-device goes offline

  Topic to which error messages are sent: `/ext/session/{productKey}/{deviceName}/combine/logout_reply`

  Error codes: 460, 520, and 500

- A sub-device is forced to go offline

  Topic to which error messages are sent: `/ext/error/{productKey}/{deviceName}`

  Error codes: 427, 521, 522, and 6401

- A sub-device fails to send a message

  Topic to which error messages are sent: `/ext/error/{productKey}/{deviceName}`

  Error code: 520

- A gateway fails to submit multiple messages from sub-devices to IoT Platform

  Topic to which error messages are sent: `/sys/${productKey}/${deviceName}/proxy/batch_post_reply`

  Error code: 9242

The following table describes the causes and solutions of the errors that may occur when a sub-device goes online or offline. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting

Error code	Cause	Troubleshooting
427	The device is forced to go offline because the device certificate is used by another device. IoT Platform identifies a device only based on the device certificate (ProductKey, DeviceName, and DeviceSecret). • The same device certificate is burned on multiple devices. • The network or power supply of the device is unstable. The device immediately reconnected to IoT Platform after an instantaneous network outage or power failure. In this case, IoT Platform identifies the reconnected device as a new device. Even if the error message is returned, the device can work as expected.	Go to the **Device Details** page in the console and check the time when the device was last connected to IoT Platform. You can determine whether the same device certificate information is used to connect another device to IoT Platform.
428	The number of sub-devices that you added to the specified gateway exceeds the limit. For more information, see Limits.	Check the number of sub-devices that are added to the gateway.
521	The device is deleted.	Go to the **Devices** page in the console and check whether the device is deleted.
522	The device is disabled.	Check the device status on the **Devices** page of the IoT Platform console.
520	An error occurred in 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 by using the current gateway.	
6287	An error occurred when the system verified the signature based on the ProductSecret or DeviceSecret.	Use a supported algorithm to calculate a signature and then verify the signature. For more information, see Connect or disconnect sub-devices.

Error code	Cause	Troubleshooting
1914	The number of sub-devices that you want to connect to or disconnect from IoT Platform at a time exceeds the limit. You can connect or disconnect up to five sub-devices at a time.	Check whether the number of sub-devices that you want to connect to or disconnect from IoT Platform at a time exceeds the limit.
1913	The gateway is disconnected from IoT Platform.	Check the reason why the gateway is disconnected from IoT Platform based on logs.
9242	The number of messages that are submitted by the gateway for the sub-devices at a time exceeds the limit, which is 50.	Check whether the number of submitted sub-device messages exceeds the limit.
9241	When you use a gateway to connect a sub-device to IoT Platform, you cannot modify the device type of an online device. For more information about device types, see the conntype parameter in Connect an MQTT client to IoT Platform.	Check whether the gateway and sub-device are online.
9240	When you connect or disconnect multiple sub-devices, the number of sub-devices in a gateway in status-related mode exceeds the limit, which is 10,000.	Check whether the number of sub-devices that you want to connect to or disconnected from IoT Platform at a time exceeds the limit.

Error codes for properties, events, and services

- A device submits properties
  - Pass-through data format:
    - Request topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - Response topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`
  - Alink data format:
    - Request topic: `/sys/${productKey}/${deviceName}/thing/event/property/post`
    - Response topic: `/sys/${productKey}/${deviceName}/thing/event/property/post_reply`

  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 describes the cause and solution of the error that may occur when a device submits properties. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting

Error code	Cause	Troubleshooting
6106	The number of properties that are submitted by the device exceeds the limit. A device can submit up to of 200 properties at a time.	Log on to the console, choose **Maintenance > Device Log**, and then check the number of properties that are submitted by the device. You can also check the information in the on-premises log file of the device.

- A device submits events
  - Pass-through data format:
    - Request topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - Response topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`
  - Alink data format:
    - Default module
      - Request topic: `/sys/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post`
      - Response topic: `/sys/${productKey}/${deviceName}/thing/event/${tsl.event.identifier}/post_reply`
    - Custom module:
      - Request topic: `/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:${tsl.event.identifier}/post`
      - Response topic: `/sys/${productKey}/${deviceName}/thing/event/${tsl.functionBlockId}:${tsl.event.identifier}/post_reply`

  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 of this topic.
- A gateway submits the data of multiple sub-devices at a time
  - Pass-through data format:
    - Request topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw`
    - Response topic: `/sys/${productKey}/${deviceName}/thing/model/up_raw_reply`
  - Alink data format:
    - Request topic: `/sys/${productKey}/${deviceName}/thing/event/property/pack/post`
    - Response topic: `/sys/${productKey}/${deviceName}/thing/event/property/pack/post_reply`

  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 describes the causes and solutions of the errors that may occur when the gateway submits the data of multiple sub-devices at a time. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting
6401	The topological relationship does not exist.	Go to the **Sub-device Management** tab in the console and check the information about the sub-device.
6106	The number of properties that are submitted by the device exceeds the limit. A device can submit up to of 200 properties at a time.	Log on to the console, choose **Maintenance > Device Log**, and then check the number of properties submitted by the device. You can also check the information in the on-premises log file of the device.
6357	The amount of data submitted by the gateway exceeds the limit. A gateway can submit the data of up to 20 sub-devices at a time.	Check the data records in the on-premises log file of the device.
6356	The number of events submitted by the gateway exceeds the limit. A gateway can submit up to 200 events at a time.	Check the data records in the on-premises log file of the device.

Error codes that may appear when devices specify desired device property values

- A device obtains desired property values
  - Request topic: `/sys/${productKey}/${deviceName}/thing/property/desired/get`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/property/desired/get_reply`

  Error codes: 460, 6104, 6661, and 500

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

Error code	Cause	Troubleshooting
6104	The number of properties contained in the request exceeds the limit. A request can contain up to 200 properties.	Log on to the console, choose **Maintenance > Device Log**, and then check the number of properties in the submitted data. You can also check the information in the on-premises log file of the device.
6661	An error occurred when the system queried the desired property values. The error message returned because a system exception occurred.	Submit a ticket.

- A device clears desired property values
  - Request topic: `/sys/${productKey}/${deviceName}/thing/property/desired/delete`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/property/desired/delete_reply`

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

Error codes for device tags

- A device submits tag information
  - Request topic: `/sys/${productKey}/${deviceName}/thing/deviceinfo/update`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/deviceinfo/update_reply`

  Error codes: 460 and 6100
- A device deletes tag information
  - Request topic: `/sys/${productKey}/${deviceName}/thing/deviceinfo/delete`
  - Response topic: `/sys/${productKey}/${deviceName}/thing/deviceinfo/delete_reply`

  Error codes: 460 and 500

Error codes that may appear when devices obtain TSL models

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

Error codes: 460, 5159, 5160, and 5161

Error codes that may appear when devices request update package information

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

> **Note** The topic that is used to obtain update package information is the same as that used to return responses.

Error codes: 429, 9112, and 500

The following table describes the cause and solution of an error that may occur when a device requests the update package information. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting
9112	The system failed to query information about the specified device.	On the Devices page of the IoT Platform console, check whether the device information is valid.

Error codes that may appear when devices request configuration data

- Request topic: `/sys/${productKey}/${deviceName}/thing/config/get`
- Response topic: `/sys/${productKey}/${deviceName}/thing/config/get_reply`

Error codes: 460, 500, 6713, and 6710

The following table describes the causes and solutions of the errors that may appear when a device requests the configuration data. For more information about other error codes, see the "Common error codes" section of this topic.

Error code	Cause	Troubleshooting
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 then 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 then check whether you have edited the configuration file for the specified product.

Error codes that may appear when you connect devices to IoT Platform by using a Paho MQTT library

If a device is unexpectedly disconnected from IoT Platform after you use an open source Paho MQTT library to connect the device to IoT Platform, you can view the error logs and error codes for the issue. For more information, see Error codes.