Alibaba Cloud IoT Platform

Developer Guide (Devices)

Issue: 20190802

Legal disclaimer

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

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

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

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

Generic conventions

Table -1: Style conventions

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

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

Contents

1 Download device SDKs

IoT Platform provides multiple device SDKs to help you develop your devices and connect them to IoT Platform. If you want to develop your own SDK, see Communications over Alink protocol for Alink data information.

Prerequisites

Before you develop a device SDK, you must complete all console configurations and obtain all necessary information (such as the device certificate information and topic information). For details, see the User Guide.

Use SDKs provided by IoT Platform

You can use an SDK provided by IoT Platform and configure the SDK according to your business requirements and the protocol you want to use. For more information, see Documents of Link Kit SDKs.

Develop an SDK based on the Alink protocol

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

2 Authenticate devices

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

IoT Platform supports the following authentication methods:

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

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

Table 2-1: Comparison of authentication methods

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

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

2.1 Authenticate devices

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

IoT Platform supports the following authentication methods:

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

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

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

Table 2-2: Comparison of authentication methods

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

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

2.2 Unique-certificate-per-device authentication

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

Context

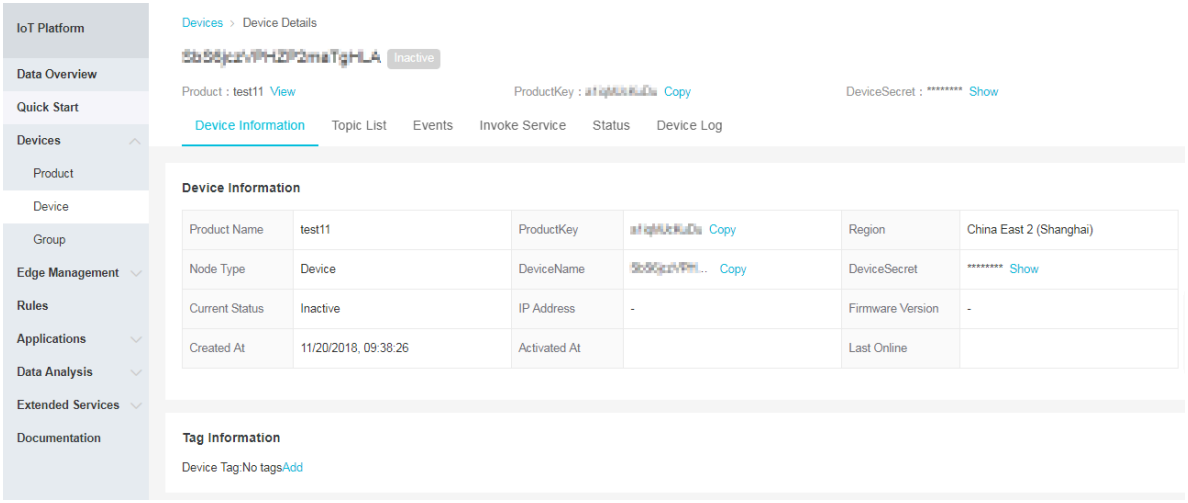
The unique-certificate-per-device authentication method is a secure authentication method. We recommend that you use this authentication method.

Workflow of unique-certificate-per-device authentication:

Procedure

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

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



3. Install the certificate to the device.

   Follow these steps:

   a) Download a device-side SDK.

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

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

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

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

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

2.3 Unique-certificate-per-product authentication

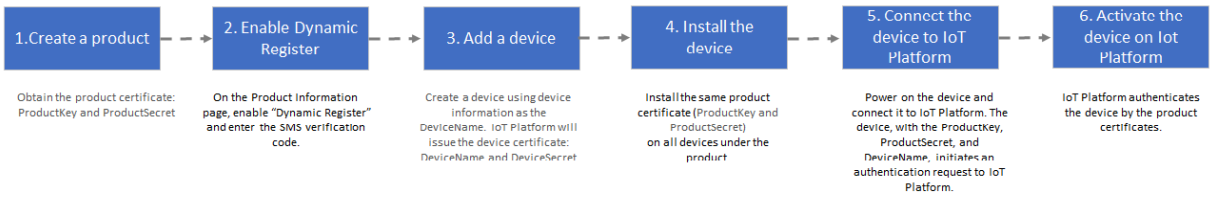
Using unique-certificate-per-product authentication method requires that devices of a product have been installed with a same firmware in which a product certificate (ProductKey and ProductSecret) has been installed. When a device initiates an activation request, IoT Platform authenticates the product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

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

Context

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

Workflow of unique-certificate-per-product authentication:

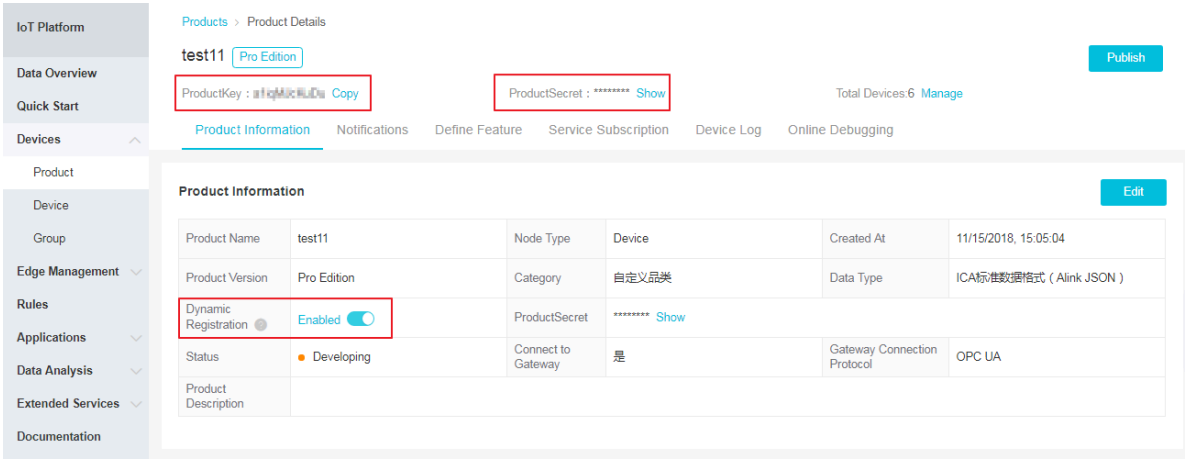


Procedure

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

   > **Note:**

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



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

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

4. Install the product certificate to the device.

   Follow these steps:

   a) Download a device-side SDK.

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

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

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

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

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

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

 Note:

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

3 Protocols for connecting devices

3.1 MQTT standard

Supported versions

The Alibaba Cloud IoT Platform currently supports MQTT-based connections. Both MQTT versions 3.1 and 3.1.1 are supported. For more information about these protocols, see MQTT 3.1.1 and MQTT 3.1.

Comparisons between IoT Platform based MQTT and standard MQTT

- IoT Platform supports MQTT packets including PUB, SUB, PING, PONG, CONNECT, DISCONNECT, and UNSUB.
- Supports cleanSession.
- Does not support will and retain msg.
- Does not support QoS 2.
- Supports the RRPC sychronization mode based on native MQTT topics. The server can call the device and obtain a device response result at the same time.

Security levels

Supports secure connections over protocols such as TLS version 1, TLS version 1.1, and TLS version 1.2.

- TCP channel plus encrypted chip (ID^2 hardware integration): High security.
- TCP channel plus symmetric encryption (uses the device private key for symmetric encryption): Medium security.
- TCP (the data is not encrypted): Low security.

Topic standards

After you have created a product, all devices under the product have access to the following topic categories by default:

- /${productKey}/${deviceName}/update pub
- /${productKey}/${deviceName}/update/error pub
- /${productKey}/${deviceName}/get sub
- /sys/${productKey}/${deviceName}/thing/# pub&sub

- /sys/${productKey}/${deviceName}/rrpc/# pub&sub
- /broadcast/${productKey}/# pub&sub

Each topic rule is a topic category. Topic categories are isolated based on devices. Before a device sends a message, replace deviceName with the deviceName of your own device. This prevents the topic from being sent to another device with the same deviceName. The topics are as follows:

- pub: The permission to submit data to topics.
- sub: The permission to subscribe to topics.
- Topic categories with the following format: /${productKey}/${deviceName}/xxx: Can be expanded or customized in the IoT Platform console
- Topic categories that begin with "/sys": The application protocol communication standards established by the system. User customization is disabled. The topic should comply with the Alibaba Cloud Alink protocol.
- Topic categories with the following format: /sys/${productKey}/${deviceName}/thing/xxx: The topic category is used by gateway and sub-devices. It is used in gateway scenarios.
- Topic categories that begin with "/broadcast": Broadcast topics
- /sys/${productKey}/${deviceName}/rrpc/request/${messageId}: Used to synchronize requests. The server dynamically generates a topic for the message ID. The device can subscribe to topic categories with wildcard characters.
- /sys/${productKey}/${deviceName}/rrpc/request/+: After a message is received, a pub message is sent to /sys/${productKey}/${deviceName}/rrpc/response/${messageId}. The server sends a request and receives a response at the same time.

3.2 Establish MQTT connections over TCP

This topic describes how to establish MQTT connections over TCP by using two methods: direct connection and connection after HTTPS verification.

Note:

When you configure MQTT CONNECT packets:

- Do not use the same device certificate (ProductKey, DeviceName, and DeviceSecret) for multiple physical devices for connection authentication. This is because when a new device initiates authentication to IoT Platform, a device that is already

connected to IoT Platform using the same device certificate will be brought offline. Later, the device which was brought offline will try to connect again, causing the newly connected device to be brought offline instead.

- In MQTT connection mode, open-source SDKs automatically reconnect to IoT Platform after they are brought offline. You can check the actions of devices by viewing the device logs.

Direct MQTT client connection

1. We recommend that you use the TLS protocol for encryption, because it provides better security. Click here to download the TLS root certificate.
2. Connect devices to the server using the MQTT client. For connection methods, see Open-source MQTT client references. For more information about the MQTT protocol, see http://mqtt.org.

   Note:
   Alibaba Cloud does not provide technical support for third-party code.
3. Establish an MQTT connection.

Connection domain name	`${ YourProduc tKey }. iot - as - mqtt . ${ YourRegion Id }. aliyuncs . com : 1883` Replace *${YourProductKey}* with your ProductKey. Replace *${YourRegionId}* with the region ID of your device. For information about regions and zones, see Regions and zones.

Variable header: Keep Alive	The Keep Alive parameter must be included in the CONNECT packet. The allowed range of Keep Alive value is 30-1200 seconds. If the value of Keep Alive is not in this range, IoT Platform will reject the connection. We recommend that you set a value larger than 300 seconds. If the Internet connection is not stable, set a larger value.

Parameters in an MQTT CONNECT packet	`mqttClient Id : clientId +"| securemode = 3 , signmethod = hmacsha1 , timestamp = 132323232 |" mqttUserna me : deviceName +"&"+ productKey mqttPasswo rd : sign_hmac ( deviceSecr et , content )`
	
	`mqttPasswo rd` : Sort the parameters to be submitted to the server alphabetically and then encrypt the parameters based on the specified sign method.
	
	The content value is a string that is built by sorting and concatenating the ProductKey, DeviceName, timestamp (optional) and clientId in alphabetical order, without any delimiters.
	
	· clientId: The client ID is a device identifier. We recommend that you use the MAC address or the serial number of the device as the client ID. The length of the client ID must be within 64 characters.
	· timestamp: The 13-digit timestamp of the current time. This parameter is optional.
	· mqttClientId: Extended parameters are placed between vertical bars (|).
	· signmethod: The signature algorithm. Valid values: hmacmd5, hmacsha1, and hmacsha256. Default value: hmacmd5.
	· securemode: The current security mode. Value options: 2 (TLS connection) and 3 (TCP connection).
	
	Example:
	
	Suppose that `clientId = 12345 , deviceName = device , productKey = pk , timestamp = 789 , signmethod = hmacsha1 , deviceSecr et = secret` .
	
	The MQTT CONNECT packet sent over TCP is as follows:
	
	`mqttclient Id = 12345 | securemode = 3 , signmethod = hmacsha1 , timestamp = 789 | mqttUserna me = device & pk mqttPasswo rd = hmacsha1 (" secret " "`

Use HTTPS for device authentication. The authentication URL is `https://iot-auth.${YourRegionId}.aliyuncs.com/auth/devicename`. Replace *${YourRegionId}* with the region ID of your device. For more information about regions, see Regions and zones.

- Request parameters

Parameter	Required	Description
productKey	Yes	The unique identifier of the product. You can view it in the IoT Platform console.
deviceName	Yes	The device name. You can view it in the IoT Platform console.
sign	Yes	The signature. The format is hmacmd5(deviceSecret, content). The content value is a string that is built by sorting and concatenating of all the parameters (except version, sign, resources, and signmethod) that need to be submitted to the server in alphabetical order.
signmethod	No	The signature algorithm. Valid values: hmacmd5, hmacsha1, and hmacsha256. Default value: hmacmd5.
clientId	Yes	The client ID. The length must be within 64 characters.
timestamp	No	Timestamp. Timestamp verification is not required.
resources	No	The resource that you want to obtain, such as MQTT. Use commas (,) to separate multiple resource names.

- Response parameters

Parameter	Required	Description
iotId	Yes	The connection tag that is issued by the server. It is used to specify a value for the user name for the MQTT CONNECT packet.

Parameter	Required	Description
iotToken	Yes	The token is valid for seven days. It is used as the password for the MQTT CONNECT packet.
resources	No	The resource information. The extended information includes the MQTT server address and CA certificate information.

- Request example using x-www-form-urlencoded:

```
POST  / auth / devicename   HTTP / 1 . 1
Host :  iot - auth . cn - shanghai . aliyuncs . com
Content - Type :  applicatio  n / x - www - form - urlencoded
Content - Length :  123
productKey = 123 & sign = 123 & timestamp = 123 & version =
default & clientId = 123 & resouces = mqtt & deviceName = test
sign  =  hmac_md5 ( deviceSecr  et ,  clientId12  3deviceNam
etestprodu  ctKey123ti  mestamp123 )
```

- Response example:

```
HTTP / 1 . 1   200   OK
Server :  Tengine
Date :  Wed ,  29   Mar   2017   13 : 08 : 36   GMT
Content - Type :  applicatio  n / json ; charset = utf - 8
Connection :  close
{
     " code " :  200 ,
     " data " : {
        " iotId " : " 42Ze0mk355  6498a1AlTP ",
        " iotToken " : " 0d7fdeb9dc  1f4344a2cc  0d45edcb0b  cb ",
        " resources " : {
            " mqtt " : {
               " host " : " xxx . iot - as - mqtt . cn - shanghai
 . aliyuncs . com ",
               " port " :  1883
        }
      }
      },
      " message " : " success "
}
```

2. Establish an MQTT connection.

   a. Download the root.crt file of IoT Platform. We recommend that you use TLS 1.2.

   b. Connect the device client to the Alibaba Cloud MQTT server using the returned MQTT host address and port of device authentication.

   c. Establish a connection over TLS. The device client authenticates the IoT Platform server by CA certificates. The IoT Platform server authenticates the device client by the information in the MQTT CONNECT packet. In the packet, username=iotId, password=iotToken, clientId=custom device identifier (we

recommend that you use the MAC address or the device serial number as the device identifier).

If the iotId or iotToken is invalid, then the MQTT connection fails. The connect acknowledgment (ACK) flag you receive is 3.

The error codes are described as follows:

- 401: request auth error. This error code is returned when the signature is mismatched.
- 460: param error. Parameter error.
- 500: unknown error. Unknown error.
- 5001: meta device not found. The specified device does not exist.
- 6200: auth type mismatch. The authentication type is invalid.

MQTT Keep Alive

In a keep alive interval, the device must send at least one packet, including ping requests.

If IoT Platform does not receive any packets in a keep alive interval, the device is disconnected from IoT Platform and needs to reconnect to the server.

The keep alive time must be in a range of 30 to 1200 seconds. We recommend that you set a value larger than 300 seconds.

3.3 Establish MQTT connections over WebSocket

IoT Platform supports MQTT over WebSocket. You can first use the WebSocket protocol to establish a connection, and then use the MQTT protocol to communicate on the WebSocket channel.

Context

Using WebSocket has the following advantages:

- Allows browser-based applications to establish persistent connections to the server.
- Uses port 433, which allows messages to pass through most firewalls.

Procedure

1. Certificate preparation

   The WebSocket protocol includes WebSocket and WebSocket Secure. Websocket and WebSocket Secure are used for unencrypted and encrypted connections, respectively. Transport Layser Security (TLS) is used in WebSocket Secure connections. Like a TLS connection, a WebSocket Secure connection requires a root certificate.

2. Client selection

   IoT Platform provides Java MQTT SDK. You can directly use this client SDK by replacing the connect URL with a URL that is used by WebSocket. For clients that use other language versions or connections without using the official SDK, see Open-source MQTT clients. Make sure that the client supports WebSocket.

3. Connections

   An MQTT over WebSocket connection has a different protocol and port number in the connect URL from an MQTT over TCP connection. MQTT over WebSocket connections have the same parameters as MQTT over TCP connections. The securemode parameter is set to 2 and 3 for WebSocket Secure connections and WebSocket connections, respectively.

   - Connection domain for Shanghai region: ${YourProductKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com:443

     Replace ${YourProductKey} with your ProductKey.

   - Variable header: Keep Alive

     The Keep Alive parameter must be included in the CONNECT packet. The allowed range of Keep Alive value is 30-1200 seconds. If the value of Keep Alive is not in this range, IoT Platform will reject the connection. We recommend that you set a value larger than 300 seconds. If the Internet connection is not stable, set a larger value.

     In a keep alive interval, the device must send at least one packet, including ping requests.

     If IoT Platform does not receive any packets in a keep alive interval, the device is disconnected from IoT Platform and needs to reconnect to the server.

   - An MQTT Connect packet contains the following parameters:

```
mqttClient  Id :  clientId +"| securemode = 3 , signmethod =
hmacsha1 , timestamp = 132323232 |"
mqttUserna  me :  deviceName +"&"+ productKey
mqttPasswo  rd :   sign_hmac ( deviceSecr  et , content ).
Sort   and   concatenat  e   the   input   parameters   in
alphabetic  al   order   and   then   encrypt   the   parameters
  using   the   specified   sign   method .
The   value   of   content   is   a   string   that   is   built
  by   sorting   and   concatenat  ing   the   parameters   sent
  to   the   server   ( productKey ,  deviceName ,  timestamp ,
and   clientId ).
```

Where,

- clientId: Specifies the client ID up to 64 characters. We recommend that you use the MAC address or SN code.
- timestamp: (Optional) Specifies the current time in milliseconds.
- mqttClientId: Parameters within `||` are extended parameters.
- signmethod: Specifies a signature algorithm.
- securemode: Specifies the secure mode. Values include 2 (WebSocket Secure) and 3 (WebSocket).

The following are examples of MQTT Connect packets. Suppose the parameter values are:

```
clientId = 12345 ,  deviceName = device ,  productKey = pk ,
timestamp = 789 ,  signmethod = hmacsha1 ,  deviceSecr  et = secret
```

- For a WebSocket connection:
  - Connection domain

    ```
    ws :// pk . iot - as - mqtt . cn - shanghai . aliyuncs . com :
    443
    ```

  - Connection parameters

    ```
    mqttclient  Id = 12345 | securemode = 3 , signmethod =
    hmacsha1 , timestamp = 789 |
    mqttUserna  me = device & pk
    ```

```
mqttPasswr  od = hmacsha1 (" secret "," clientId12
345deviceN  amedevicep  roductKeyp  ktimestamp  789 ").
toHexStrin  g ();
```

- For a WebSocket Secure connection:
  - Connection domain

```
wss :// pk . iot - as - mqtt . cn - shanghai . aliyuncs . com
: 443
```

  - Connection parameters

```
mqttclient  Id = 12345 | securemode = 2 , signmethod =
hmacsha1 , timestamp = 789 |
mqttUserna  me = device & pk
mqttPasswr  od = hmacsha1 (" secret "," clientId12
345deviceN  amedevicep  roductKeyp  ktimestamp  789 ").
toHexStrin  g ();
```

3.4 Examples of creating signatures for MQTT connections

This topic provides sample signature code for you to develop your devices, so that your devices can communicate with IoT Platform over MQTT without using the device SDK provided by Alibaba Cloud.

Description

We recommend that you use a device SDK provided by Alibaba Cloud. If a device SDK in any language is used, you do not need to configure your own signature mechanism. To view the SDK download path, access Download device SDKs.

If you use other methods to connect your devices to IoT Platform, note the following information:

- You must take the responsibility to ensure connection stability and maintain the keepalive and reconnection mechanisms for MQTT connections.
- Alibaba Cloud will not provide related technical support for any possible connection issues.
- If you want to use IoT Platform features, such as OTA, TSL models, and unique -certificate-per-product authentication, you must compile your own code to implement these features. This may consume a lot of development time and bug fixing time.

Sample code for signature calculation

If you do not use the device SDKs provided by Alibaba Cloud IoT Platform, click the following links to access the corresponding sample code pages.

- sign_mqtt.c: the sample code used to implement the signature function.
- sign_api.h: the sample code that defines the data structure used by the signature function.
- sign_sha256.c: the sample code that defines the algorithm implementations that can be used by the signature function. If you have HMACSHA256 implementation on your own platform, you do not need to compile this code file. However, you must provide the `utils_hmac_sha256()` function that can be called by the APIs defined in `sign_mqtt.c`.
- sign_test.c: the sample code used to test the signature function.

Description of the signature function

Function prototype	int32_t IOT_Sign_MQTT(iotx_mqtt_region_types_t region, iotx_dev_meta_info_t *meta, iotx_sign_mqtt_t *signout);
Description	Outputs the information required to connect to Alibaba Cloud IoT Platform based on the input information about the IoT device authentication. The connection information contains the domain name, MQTT client ID, MQTT username, and MQTT password. Then, you can provide the information to the MQTT client to connect to Alibaba Cloud IoT Platform.

Input parameters	The following input parameters are included: · region: the Alibaba Cloud endpoints to which the specified device connects. Sample code: `typedef enum { IOTX_CLOUD_REGION_SHANGHAI , /* Shanghai */ IOTX_CLOUD_REGION_SINGAPORE , /* Singapore */ IOTX_CLOUD_REGION_JAPAN , /* Japan */ IOTX_CLOUD_REGION_USA_WEST , /* America */ IOTX_CLOUD_REGION_GERMANY , /* Germany */ IOTX_CLOUD_REGION_CUSTOM , /* Custom setting */ IOTX_CLOUD_DOMAIN_MAX /* Maximum number of domain */ } iotx_mqtt_region_types_t ;` · meta: the identity authentication information of the specified device. Note: The API caller must allocate memory for meta. Sample code: `typedef struct _iotx_dev_meta_info { char product_key [IOTX_PRODUCT_KEY_LEN + 1]; char product_secret [IOTX_PRODUCT_SECRET_LEN + 1]; char device_name [IOTX_DEVICE_NAME_LEN + 1]; char device_secret [IOTX_DEVICE_SECRET_LEN + 1]; } iotx_dev_meta_info_t ;` The parameters in this example are described as follows: - product_key: the ProductKey of the product to which the device belongs. - product_secret: the ProductSecret of the product to which the device belongs. - device_name: the device name. - device_secret: the device secret.

Output parameters	signout: the output data, which is used to establish an MQTT connection. Sample code: `typedef struct { char hostname [ DEV_SIGN_H OSTNAME_MA XLEN ]; uint16_t port ; char clientid [ DEV_SIGN_C LIENT_ID_M AXLEN ]; char username [ DEV_SIGN_U SERNAME_MA XLEN ]; char password [ DEV_SIGN_P ASSWORD_MA XLEN ]; } iotx_sign_ mqtt_t ;` The parameters in this example are described as follows: · hostname: the complete domain name of the Alibaba Cloud IoT endpoint. · port: the port number of the Alibaba Cloud IoT endpoint. · clientid: the client ID that must be specified to establish an MQTT connection. · username: the username that must be specified to establish an MQTT connection. · password: the password that must be specified to establish an MQTT connection.
Return values	· 0: indicates that the call is successful. · -1: indicates that the call has failed because the input parameters are invalid.

Example on how to use the signature function

In this example, the test code in sign_test.c is used.

```
# include < stdio . h >
# include < string . h >
# include " sign_api . h " // Defines all data structures
 that are used in the signature function .

// The following macros are used to define the Alibaba
   Cloud authentica tion informatio n of the device :
 ProductKey , ProductSec ret , DeviceName , and DeviceSecr et .
// In actual product developmen t , the device authentica
 tion informatio n must be encrypted by the device
 manufactur er and stored in the flash of the device
   or a file .
// These macros are read and used after the device
 is powered on .
# define EXAMPLE_PR ODUCT_KEY " a1X2bEn ****"
# define EXAMPLE_PR ODUCT_SECR ET " 7jluWm1zql 7b ****"
```

```
# define   EXAMPLE_DE  VICE_NAME      " example1 "
# define   EXAMPLE_DE  VICE_SECRE  T    " ga7XA6KdlE  eiPXQPpRbA
 jOZXwG8y ****"

 int   main ( int   argc ,  char  * argv [])
{
     iotx_dev_m  eta_info_t   meta_info ;
     iotx_sign_  mqtt_t   sign_mqtt ;

     memset (& meta_info ,  0 ,  sizeof ( iotx_dev_m  eta_info_t ));
    // Use   the   following   code   to   copy   the   previously
 defined   device   identity   informatio  n   to   meta_info .
     memcpy ( meta_info . product_ke  y ,  EXAMPLE_PR  ODUCT_KEY ,
 strlen ( EXAMPLE_PR  ODUCT_KEY ));
     memcpy ( meta_info . product_se  cret ,  EXAMPLE_PR  ODUCT_SECR
 ET ,  strlen ( EXAMPLE_PR  ODUCT_SECR  ET ));
     memcpy ( meta_info . device_nam  e ,  EXAMPLE_DE  VICE_NAME ,
 strlen ( EXAMPLE_DE  VICE_NAME ));
     memcpy ( meta_info . device_sec  ret ,  EXAMPLE_DE  VICE_SECRE  T
 ,  strlen ( EXAMPLE_DE  VICE_SECRE  T ));

    // Call   the   signature   function   to   generate   various
 data   that   is   required   to   establish   an   MQTT   connection
 .
     IOT_Sign_M  QTT ( IOTX_CLOUD  _REGION_SH  ANGHAI , & meta_info ,
 & sign_mqtt );

    ...

}
```

3.5 CoAP standard

Protocol version

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

Channel security

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

Open-source client reference

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

Note:

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

Alibaba Cloud CoAP agreement

- Do not use a question mark (?) to set a parameter.

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

3.6 Establish connections over CoAP

IoT Platform supports connections over CoAP. CoAP is suitable for resource-constrained, low-power devices, such as NB-IoT devices. This topic describes how to connect devices to IoT Platform over CoAP and two supported authentication methods, which are DTLS and symmetric encryption.

Use the symmetric encryption method

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

   Note:

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

2. Authenticate the device.

   Request message:

```
POST / auth
Host : ${ YourProduc tKey }. coap . cn - shanghai . link .
aliyuncs . com
Port : 5682
Accept : applicatio n / json or applicatio n / cbor
Content - Format : applicatio n / json or applicatio n /
cbor
payload : {" productKey ":" a1NUjcV ****"," deviceName ":"
ff1a11e7c0 8d4b3db2b1 500d8e0e55 "," clientId ":" a1NUjcV
****& ff1a11e7c0 8d4b3db2b1 500d8e0e55 "," sign ":" F9FD53EE0C
D010FCA40D 14A9FE ******", " seq ":" 10 "}
```

   Table 3-1: Parameter description

Parameter	Description
Method	The request method. The supported method is POST.
URL	/auth.

Parameter	Description
Host	The endpoint address. The format is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`. Replace `${YourProductKey}` with the ProductKey value of the device.
Port	The port number. Set the value to `5682`.
Accept	The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.
Content-Format	The encoding format of the data that the device sends to IoT Platform. Currently, application/json and application/cbor are supported.
payload	The device information for authentication, in JSON format. For more information, see the following table payload parameters.

Table 3-2: payload parameters

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

Parameter	Required	Description
sign	Yes	Signature. The signature algorithm is `hmacmd5 ( DeviceSecr et , content )`. The value of `content` is a string that is built by sorting and concatenating all the parameters (except `version` , `sign` , `resources` , and `signmethod`) that need to be submitted to the server in alphabetical order, without any delimiters. Signature calculation example: `sign = hmac_md5 ( mRPVdzSMu2 nVBxzK77ER PIMxSYIv ****, clientIda1 NUjcV ****& ff1a11e7c0 8d4b3db2b1 500d8e0e55 deviceName ff1a11e7c0 8d4b3db2b1 500d8e0e55 productKey a1NUjcV **** seq10times tamp152444 8722000 )`
signmethod	No	The algorithm type. The supported types are hmacmd5 and hmacsha1. The default value is hmacmd5.
clientId	Yes	The device identifier, which can be any string up to 64 characters in length. We recommend that you use the MAC address or the SN code of the device as the clientId.
timestamp	No	The timestamp. Currently, timestamp is not verified.

Response example:

```
{" random ":" ad2b3a5eb5 1d64f7 "," seqOffset ": 1 ," token ":"
 MZ8m37hp01 w1SSqoDFzo 0010500d00 . ad2b "}
```

Table 3-3: Response parameters

Parameter	Description
random	The encryption key used for data communication.
seqOffset	The authentication sequence offset.

Parameter	Description
token	The returned token after the device is authenticated.

3. The device sends data.

   Request message:

```
POST  / topic /${ topic }
Host : ${ YourProduc  tKey }. coap . cn - shanghai . link .
aliyuncs . com
Port :  5682
Accept :  applicatio  n / json   or   applicatio  n / cbor
Content - Format :  applicatio  n / json   or   applicatio  n /
cbor
payload : ${ your_data }
CustomOpti  ons :  number : 2088 ( token ),  2089 ( seq )
```

Table 3-4: Request parameters

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

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

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

Establish DTLS connections

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

   Note:

   · *${YourProductKey}*: Replace it with the ProductKey value of the device.

   · *${port}*: The port number. Set the port number to 5684 for DTLS connections.

2. Download the root certificate.

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

   Request message:

```
POST  / auth
Host : ${ YourProduc  tKey }. coap . cn - shanghai . link .
aliyuncs . com
Port :  5684
Accept :  applicatio  n / json   or   applicatio  n / cbor
Content - Format :  applicatio  n / json   or   applicatio  n /
cbor
payload : {" productKey ":" ZG1EvTEa7N  N "," deviceName ":"
NlwaSPXsCp  TQuh8FxBGH "," clientId ":" mylight100  0002 "," sign
":" bccb3d2618  afe74b3eab  12b94042f8  7b "}
```

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

   Response example:

```
response : {" token ":" f131028107  56432e85df  d351eeb41c  04 "}
```

Table 3-5: Return codes

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

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

4. The device sends data.

   The device publishes data to a specified topic.

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

   Currently, only topics with the permission to publish messages can be used for publishing data, for example, `/${ YourProduc tKey }/${ YourDevice Name }/ pub`. Specifically, if a device name is device, and its product key is a1GFjLP3xxC, the device can send data through the address `a1GFjLP3xx C . coap . cn - shanghai . link . aliyuncs . com : 5684 / topic / a1GFjLP3xx C / device / pub`.

   Request message:

```
POST / topic /${ topic }
Host : ${ YourProduc tKey }. coap . cn - shanghai . link .
aliyuncs . com
Port : 5684
Accept : applicatio n / json or applicatio n / cbor
Content - Format : applicatio n / json or applicatio n /
cbor
payload : ${ your_data }
CustomOpti ons : number : 2088 ( token )
```

Table 3-6: Request parameters

Parameter	Required	Description
Method	Yes	The request method. The supported request method is POST.
URL	Yes	`/ topic /${ topic }` Replace the variable `${ topic }` with the device topic which will be used to publish data.

Parameter	Required	Description
Host	Yes	The endpoint address. The format is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`. Replace `${YourProductKey}` with the ProductKey value of the device.
Port	Yes	Set the value to `5684`.
Accept	Yes	The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.
Content-Format	Yes	The encoding format of the data that the device sends to IoT Platform. Currently, application/json and application/cbor are supported.
CustomOptions	Yes	· Number: 2088. · The value of token is the token information returned after auth is called to authenticate the device. Note: Token information is required every time the device sends data. If the token is lost or expires, initiate a device authentication request again to obtain a new token.

3.7 HTTP standard

HTTP protocol versions

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

Channel security

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

- Does not support passing parameters with question marks (?).
- Resource discovery is currently not supported.
- Only HTTPS is supported.

- The URI standard, the HTTP URI resources, and the MQTT topic must be consistent. See MQTT standard.

3.8 Establish connections over HTTP

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

Restrictions

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

Procedure

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

1. Authenticate the device to obtain the device token.

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

   Authentication request:

```
POST  / auth   HTTP / 1 . 1
Host :  iot - as - http . cn - shanghai . aliyuncs . com
Content - Type   applicatio  n / json
body : {" version ":" default "," clientId ":" mylight100  0002
"," signmethod ":" hmacsha1 "," sign ":" 4870141D40  67227128CB
B4377906C3  731CAC221C "," productKey ":" ZG1EvTEa7N  N ","
deviceName ":" NlwaSPXsCp  TQuh8FxBGH "," timestamp ":" 1501668289
957 "}
```

Table 3-7: Parameters

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

Table 3-8: Fields in body

Field	Required?	Description
productKey	Yes	The unique identifier of the product to which the device belongs. You can obtain this information from the Device Details page of the IoT platform console.
deviceName	Yes	The device name. You can obtain this informatio n from the Device Details page of the IoT platform console.
clientId	Yes	The client ID, a string of up to 64 characters. We recommend that you use the MAC address or SN code as the client ID.

Field	Required?	Description
timestamp	No	The timestamp. A request is valid within 15 minutes after the timestamp is created. The timestamp is in the format of numbers. The value is the number of milliseconds that have elapsed since 00:00, January 1, 1970 (GMT).
sign	Yes	The signature value. The signature algorithm is in the format of `hmacmd5 ( deviceSecr et , content )`. The value of `content` is a string that contains all the parameters to be reported to IoT Platform except `version` , `sign` , and `signmethod` . These parameters are sorted in alphabetical order and spliced without any separators. Signature example: If `clientId = 12345` , `deviceName = device` , `productKey = pk` , `timestamp = 789` , `signmethod = hmacsha1` , and `deviceSecr et = secret` , then the signature algorithm is `hmacsha1 (" secret "," clientId12 345deviceN amedevicep roductKeyp ktimestamp 789 "). toHexStrin g ();`. In this example, binary data will be converted to a case-insensitive hexadecimal string.
signmethod	No	The algorithm type. The type can be hmacmd5 or hmacsha1. If you do not specify this parameter, the default value is hmacmd5.

Field	Required?	Description
version	No	The version number. If you do not specify this parameter, the value is "default".

Sample response:

```
 body :
{
  " code ":  0 ,//  The   status   code
  " message ": " success ", //  The   message
  " Info ": {
    " token ":  " 6944e5bfb9  2e4d4ea391  8d1eda3942  f6 "
  }
}
```

Note:

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

Table 3-9: Error codes

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

2. Send data to IoT Platform.

   The device sends data to a specific topic.

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

   For example, a topic category is `/${ YourProduc tKey }/${ YourDevice Name }/ user / pub`. If the device name is device123, and its ProductKey is a1GFjLPXXXX, the device can send data through `https :// iot - as - http . cn - shanghai . aliyuncs . com / topic / a1GFjLPXXX X / device123 / user / pub`.

   Upstream data request:

```
POST / topic /${ topic } HTTP / 1 . 1
Host : iot - as - http . cn - shanghai . aliyuncs . com
password :${ token }
Content - Type : applicatio n / octet - stream
body : ${ your_data }
```

Table 3-10: Parameter description

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

   Sample response:

```
body :
{
  " code ": 0 , // The status code
```

```
  " message ": " success ", //  The   message
  " Info ": {
    " messageId ":  8926876279  16247040 ,
  }
}
```

Table 3-11: Error codes

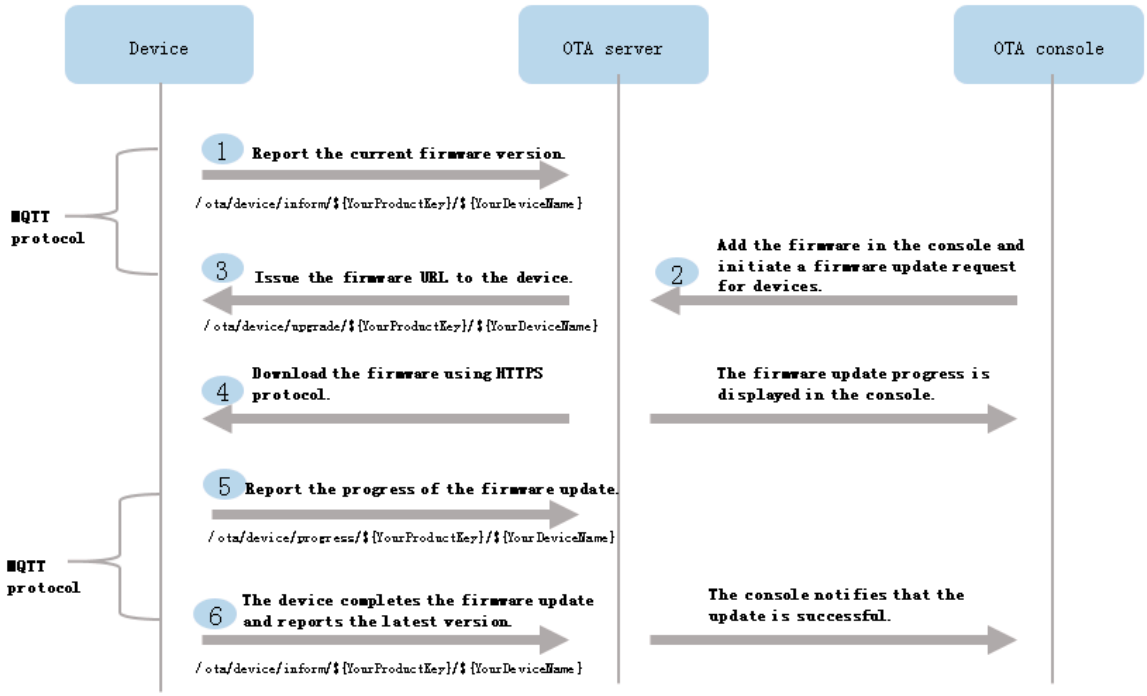
Code	Message	Description
10000	common error	Unknown error.
10001	param error	A parameter error occurred.
20001	token is expired	The token has expired. You must call `auth` to re-authenticate the device and obtain a new token.
20002	token is null	The request header does not contain any token information.
20003	check token error	An error occurred while obtaining identity information according to the token. You must call `auth` to re-authenticate the device and obtain a new token.
30001	publish message error	An error occurred while reporting data.
40000	request too many	Too many requests. The throttling policy limits the number of requests.

4 OTA updates

Devices in IoT Platform support Over-The-Air (OTA) updates. This topic introduces the process of OTA updates, the topics used in OTA updates, and the data formats.

OTA update process

The process of a firmware OTA update over MQTT protocol is shown as the following figure:



Topics for firmware update:

- Devices publish messages to the following topic to report firmware versions to IoT Platform.

```
/ ota / device / inform /${ YourProduc  tKey }/${ YourDevice  Name }
```

- Devices subscribe to the following topic to receive notifications of firmware updates from IoT Platform.

```
/ ota / device / upgrade /${ YourProduc  tKey }/${ YourDevice  Name
 }
```

- Devices publish messages to the following topic to report the progress of firmware updates to IoT Platform.

```
/ ota / device / progress /${ YourProduc  tKey }/${ YourDevice  Name
 }
```

Note:

- Devices do not periodically send firmware versions to IoT Platform. Instead, they send their firmware versions to IoT Platform only when they start.
- Even if you have triggered firmware updates for devices in the console, this does not mean that the devices have updated successfully.

  The IoT Platform firmware update system receives update progress reports from the devices when they are updating, (that is, when the update status of the devices are Updating).
- You can view the device firmware version to check whether the OTA update is successful.
- An offline device cannot receive any update notifications from the OTA server.

  When the device connects to IoT Platform again, the device notifies the OTA server that it is online. After the server receives the notification, it determines whether the device requires an update. If an update is required, the server sends the update message to the device.

Data format of messages

For OTA development and code examples, see the documentations in Link Kit SDK.

1. When devices connects to the OTA service, they report their firmware versions.

   The topic for devices to report firmware versions over MQTT protocol is `/ ota / device / inform /${ YourProduc tKey }/${ YourDevice Name }`. Message example:

   ```
   {
     " id ":  1 ,
     " params ": {
       " version ": " 1 . 0 . 0 "
     }
   }
   ```

   - `id` : The message ID.
   - `version` : The current firmware version of the device.

2. In the IoT Platform console, upload the firmware update file, verify the file using some devices, and then trigger firmware updates for all the devices of a product.

   For more information, see Firmware update.

3. When you trigger a batch update in the console, devices of the product will receive the URL of the firmware file.

   Devices subscribe to the topic `/ ota / device / upgrade /${ YourProduc tKey }/${ YourDevice Name }` to receive update messages. Then, when you initiate firmware update requests to devices, the devices receive the URL of the firmware file from this topic. A message example is as follows:

   ```
   {
     " code ": " 1000 ",
     " data ": {
       " size ":  432945 ,
       " version ": " 2 . 0 . 0 ",
       " url ": " https :// iotx - ota - pre . oss - cn - shanghai
    . aliyuncs . com / nopoll_0 . 4 . 4 . tar . gz ?     Expires =
    1502955804 & OSSAccessK  eyId = XXXXXXXXXX  XXXXXXXXXX & Signature
    = XfgJu7P6DW  WejstKJgXJ  EH0qAKU % 3D & security -   token
    = CAISuQJ1q6  Ft5B2yfSjI  pK6MGsyN1J  x5jo6mVnfB  glIPTvlvt5
    D50Tz2IHtI  f3NpAusdsv  03nWxT7v4f  lqFyTINVAE  vYZJOPKGrG
    R0DzDbDasu  mZsJbo4f % 2FMQBqEaXP  S2MvVfJ % 2BzLrf0ceu
    sbFbpjzJ6x  aCAGxypQ12  iN % 2B % 2Fr6 % 2F5gdc9FcQ  SkL0B8ZrFs
    KxBltdUROF  bIKP % 2BpKWSKuGf  LC1dysQcO1  wEP4K % 2BkkMqH8Ui  c3h
    % 2Boy % 2BgJt8H2Pp  Hhd9NhXuV2  WMzn2 % 2FdtJOiTkn  xR7ARasaBq
    helc4zqA % 2FPPlWgAKv  kXba7aIoo0  1fV4jN5JXQ  fAU8KLO8tR
    jofHWmojNz  BJAAPpYSSy  3Rvr7m5efQ  rrybY1lLO6  iZy % 2BVio2VSZD
    xshI5Z3McK  ARWct06MWV  9ABA2TTXXO  i40BOxuq % 2B3JGoABXC  54TOlo7
    % 2F1wTLTsCU  qzzeIiXVOK  8CfNOkfTuc  MGHkeYeCdF  km % 2FkADhXAnr
    nGf5a4FbmK  MQph2cKsr8  y8UfWLC6Iz  vJsClXTnbJ  BMeuWIqo5z
    IynS1pm7gf % 2F9N3hVc6 % 2BEeIk0xfl  2tycsUpbL2  FoaGk6BAF8
    hWSWYUXsv5  9d5Uk % 3D ",
       " md5 ": " 93230c3bde  425a9d7984  a594ac55ea  1e "
   ```

```
  },
  " id ":  1507707025 ,
  " message ": " success "
}
```

- `size` : The size of the file.
- `md5` : The firmware content encrypted by MD5, which is a 32-bit hex string.
- `url` : The URL of the firmware file. The URL is available for 24 hours. The devices must download the firmware file within 24 hours after the URL is generated.
- `version` : The firmware version.

4. Devices download the firmware from the URL over HTTPS protocol.

 **Note:**
The firmware URL will be released within 24 hours.

During the firmware downloading process, the devices report progress to IoT Platform using the topic `/ ota / device / progress /${ YourProduc  tKey }/ ${ YourDevice  Name }`. Message example:

```
{
  " id ":  1
  " params ": {
    " step ":" 1 ",
    " desc ":"  xxxxxxxx  "
  }
}
```

- `id` : The message ID.
- `step` :
  - [1, 100]: Values in this range indicate the download progress ratio.
  - -1: Failed to update.
  - -2: Failed to download the firmware.
  - -3: The device authentication failed.
  - -4: Failed to install the firmware.
- `desc` : Description of the update progress. If an error occurs, the error message is displayed in this parameter.

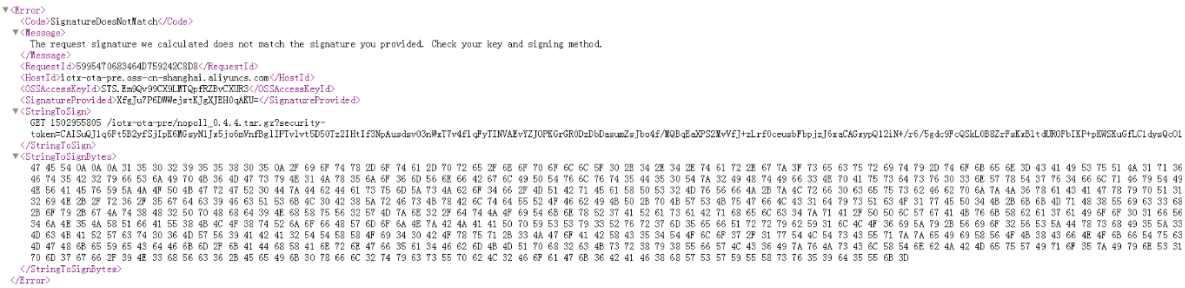
5. After devices have been updated, they report the new firmware version using this topic `/ ota / device / inform /${ YourProduc  tKey }/${ YourDevice`

`Name }`. If the reported version is the same as the version defined in the firmware update file, then the update is successful.

> **Note:**
> The reported version is the only identifier that can determine whether the update is successful. Even if the reported progress is 100%, if the device does not report the new firmware version to IoT Platform, then the update has failed.

Errors

- Signature error. If the firmware URL received by the device is incomplete or the URL content has been manually modified, the following error occurs:



- Failed to download the firmware file. The firmware file URL is expired. The URL is only available for 24 hours after its generation.

```
▼<Error>
    <Code>AccessDenied</Code>
    <Message>Request has expired.</Message>
    <RequestId>5995498D7444FA88AE536F77</RequestId>
    <HostId>iotx-ota-pre.oss-cn-shanghai.aliyuncs.com</HostId>
    <Expires>2017-08-17T07:43:24.000Z</Expires>
    <ServerTime>2017-08-17T07:45:17.000Z</ServerTime>
  </Error>
```

5 Develop devices based on Alink Protocol

5.1 Communications over Alink protocol

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

For SDKs provided by IoT Platform, see Device SDKs.

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

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

Connect devices to IoT Platform

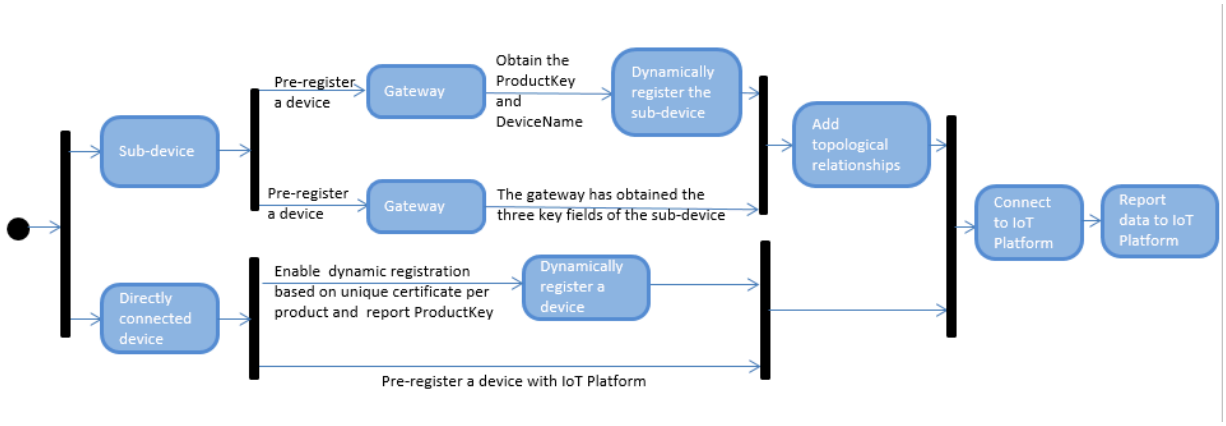
As shown in the following figure, devices can be connected to IoT Platform as directly connected devices or sub-devices. The connection process involves the following key steps: authenticate the device, establish a connection, and the device reports data to IoT Platform.

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

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

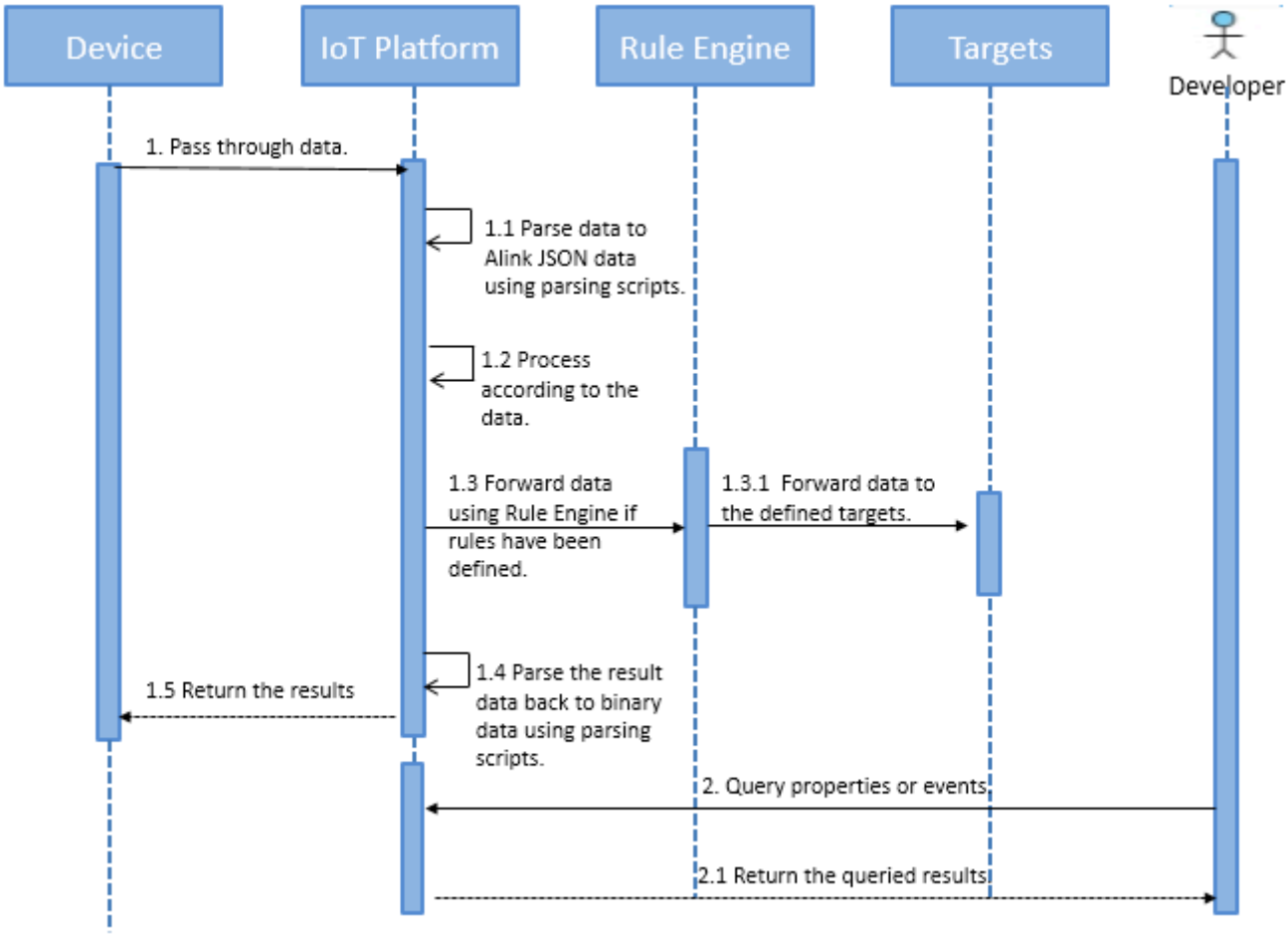
Sub-devices connect to IoT Platform through their gateways. Sub-devices can be connected to IoT Platform by using the following methods:

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



Devices report properties or events

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



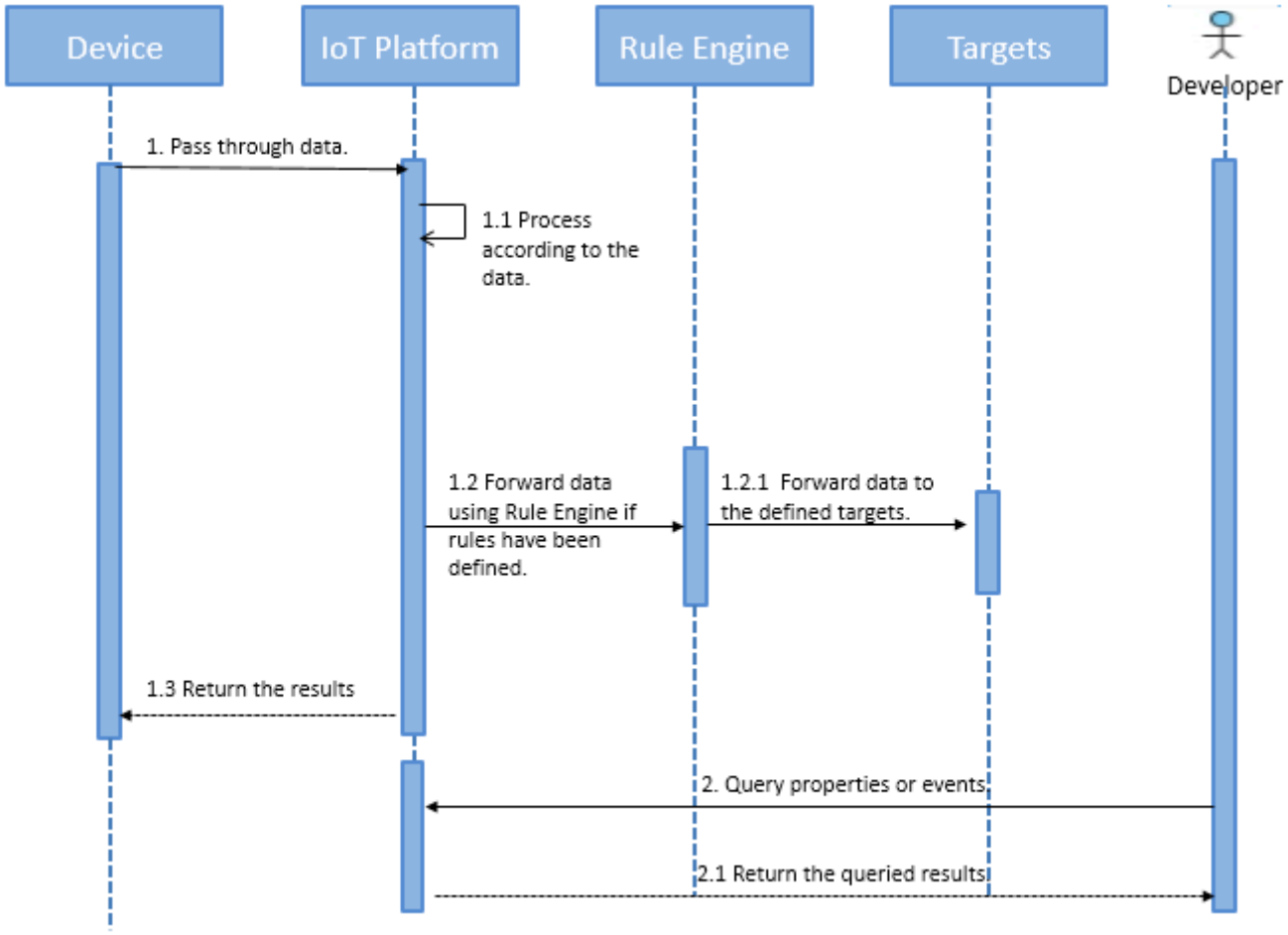
1. The device reports raw data to IoT Platform using the topic for passing through data.
2. IoT Platform parses the received data using the data parsing script that you have submitted in the IoT Platform console. The `rawDataToP rotocol` method in the script is called to convert the raw data reported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

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

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

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

- Non-pass through (Alink JSON) data



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

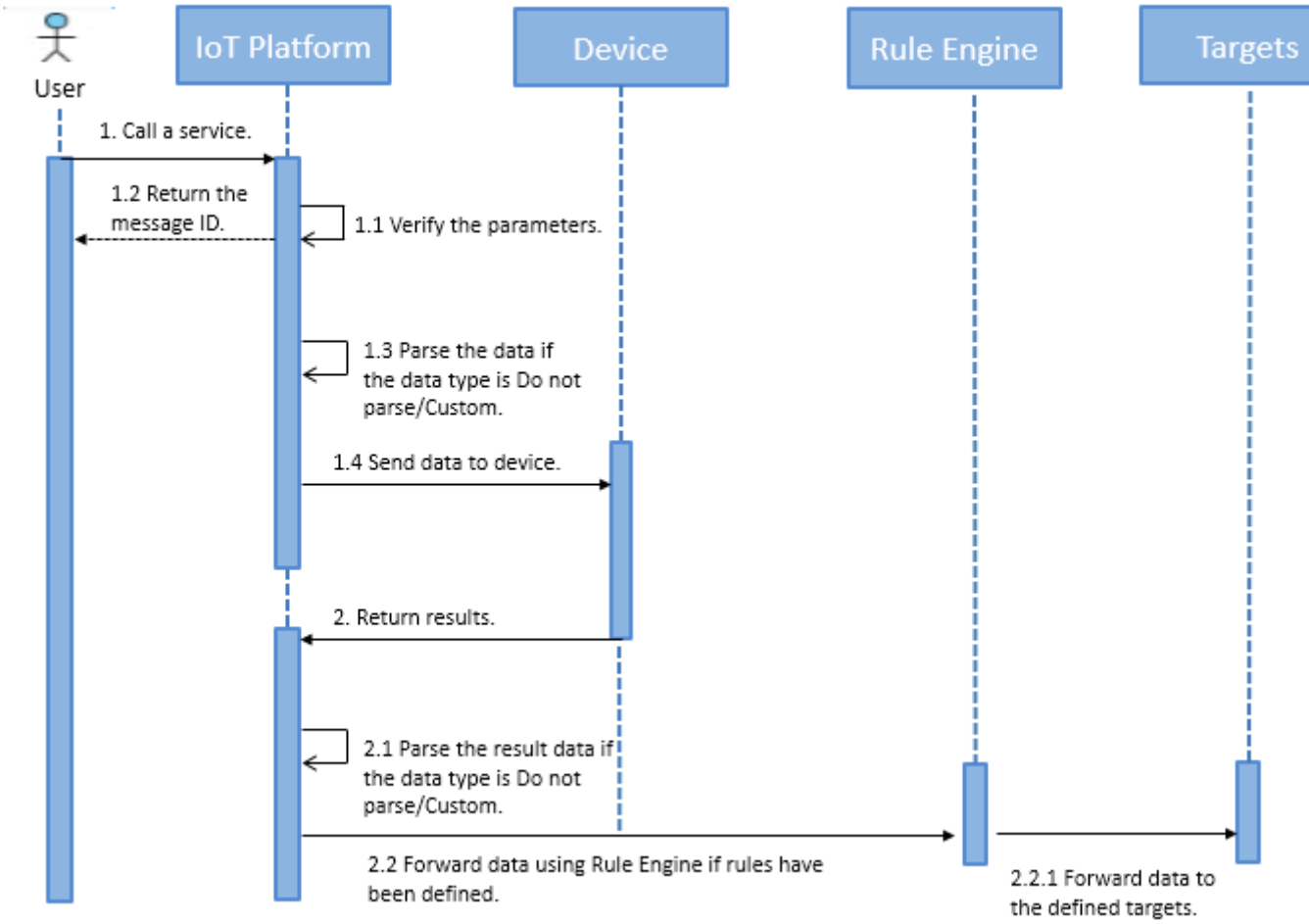
> **Note:**
> If you have configured rules for data forwarding, the data will be forwarded to the targets according to the rules.
>
> - When you configure SQL statements for rules, to obtain the device properties, specify the data topic to be `/ sys /{ productKey }/{ deviceName }/ thing / event / property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

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

3. IoT Platform returns the results to the device.
4. You can query the device property data using the API QueryDevicePropertyData and query the device event data using the API QueryDeviceEventData.

Call device services or set device properties

- Call device services or set device properties asynchronously



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

Note:

- Call the API SetDeviceProperty to set a property asynchronously.

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

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

Note:

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

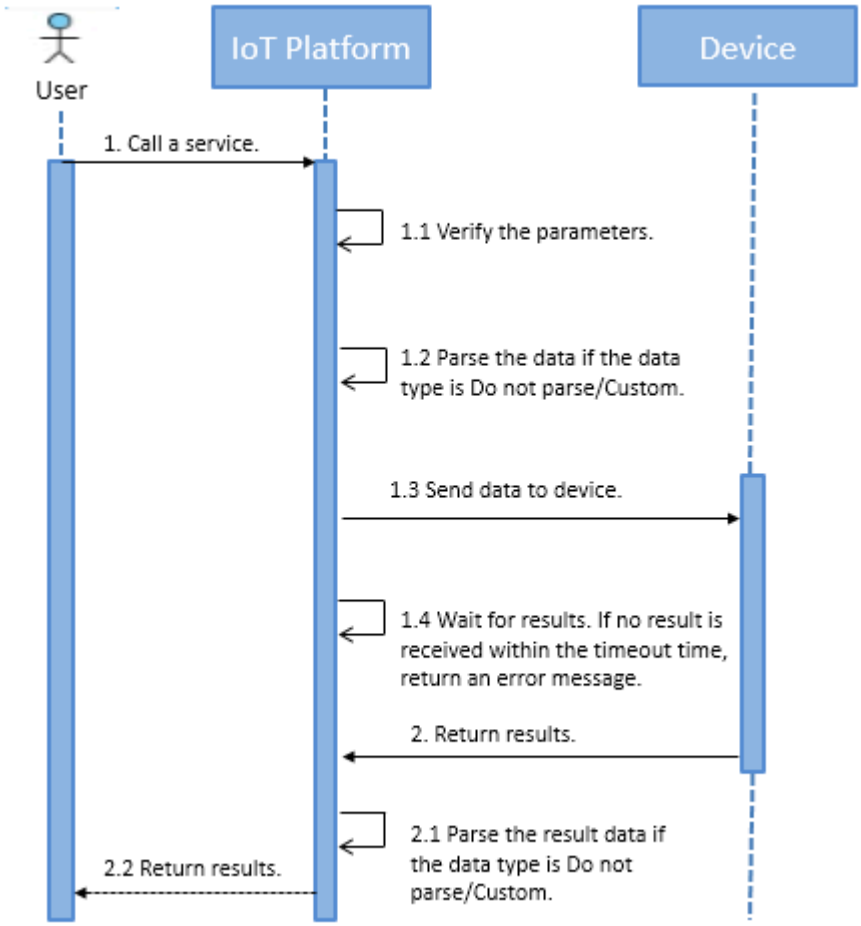
4. IoT Platform sends the data to the device, and then the device handles the request asynchronously.

Note:

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

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

- Call services using the synchronous method.



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

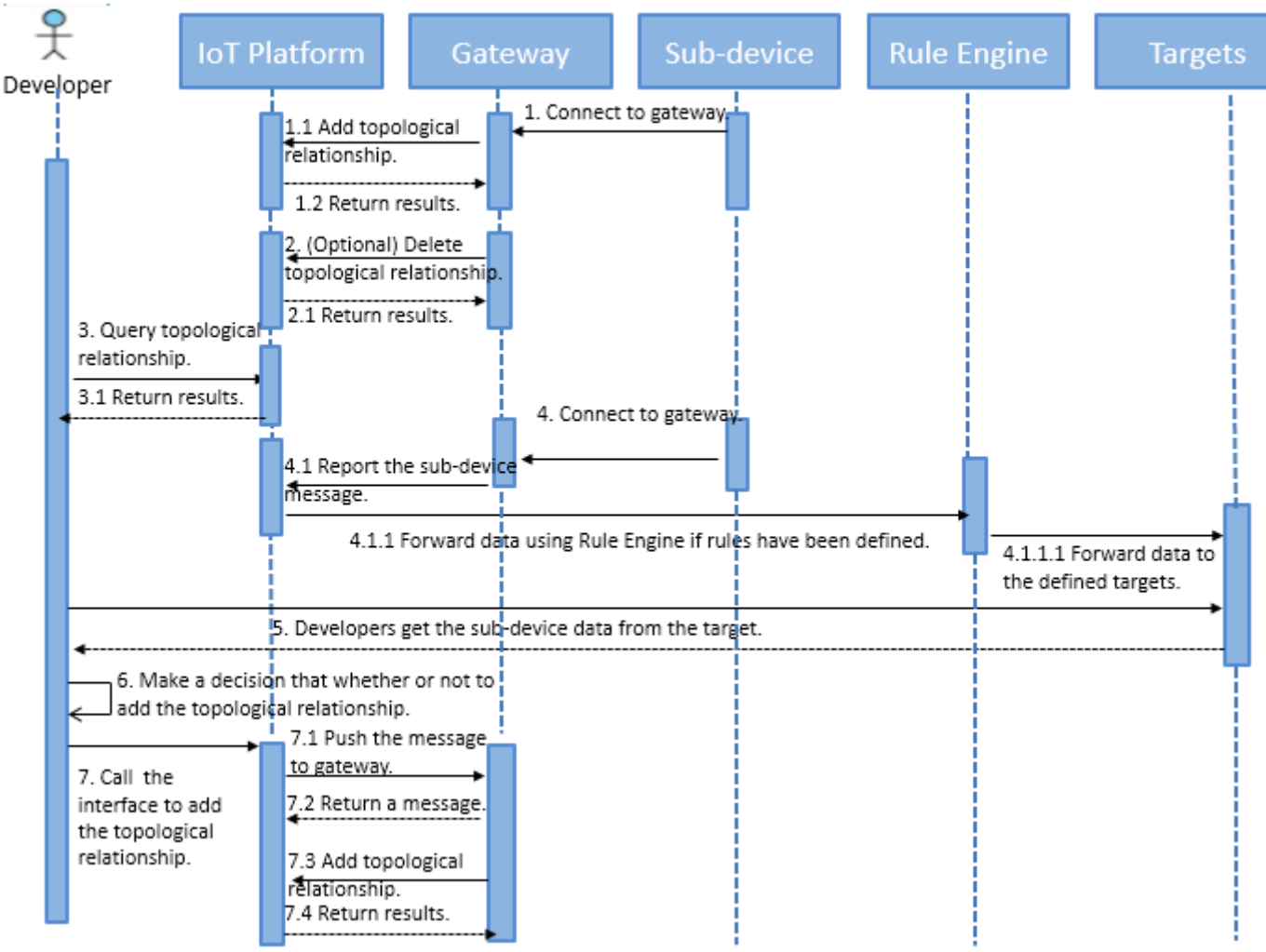
Note:

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

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

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

Build topological relationships between gateways and sub-devices.



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

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

2. Also, a gateway can send a message using the topic for deleting topological relationship messages to notify IoT Platform to remove a sub-device from the gateway.
3. Call the API GetThingTopo to query topological relationships of devices.
4. If you use the rules engine to forward device messages to another Alibaba Cloud service, and you receive device messages from that service, the process of building a topological relationship is as the following.

   a. The gateway device reports the information of the sub-device that has been detected to IoT Platform.
   b. IoT Platform receives the message and then forwards the message to the data forwarding target that you have specified when you were configuring the rule.
   c. You obtain the sub-device information from the data forwarding target service and then determine whether or not to build the topological relationship. Call the API NotifyAddThingTopo to send a request for building topological relationship to IoT Platform.
   d. IoT Platform receives the request from NotifyAddThingTopo, and then pushes the request to the gateway.
   e. The gateway receives the request and builds the topological relationship with the sub-device.

Note:

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

5.2 Device identity registration

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

The following methods are available for identity registration:

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

Dynamically register a sub-device

Upstream

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 "
    }
  ]
}
```

Response message

```
{
  " id ": " 123 ",
  " code ":  200 ,
  " data ": [
    {
      " iotId ": " 12344 ",
      " productKey ": " 1234556554 ",
      " deviceName ": " deviceName  1234 ",
      " deviceSecr  et ": " xxxxxx "
    }
  ]
}
```

Parameter description

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

Parameter	Type	Description
productKey	String	ID of the product to which the sub-device belongs.
iotId	String	Unique identifier of the sub-device.
deviceSecret	String	DeviceSecret key.
code	Integer	Result code.

Error messages

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

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

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

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

Request message

```
POST  / auth / register / device    HTTP / 1 . 1
Host :  iot - auth . cn - shanghai . aliyuncs . com
Content - Type :  applicatio  n / x - www - form - urlencoded
Content - Length :  123
productKey = 1234556554 & deviceName = deviceName  1234 & random =
567345 & sign = adfv123hdf  dh & signMethod = HmacMD5
```

Response message

```
{
  " code ":  200 ,
  " data ": {
    " productKey ": " 1234556554 ",
    " deviceName ": " deviceName  1234 ",
```

```
    " deviceSecr  et ": " adsfweafds  f "
  },
  " message ": " success "
}
```

Parameter description

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

Sign the parameters

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

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

Example:

```
sign  =  hmac_sha1 ( productSec  ret ,  deviceName  deviceName
1234produc  tKey123455  6554random  123 )
```

5.3 Add a topological relationship

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

IoT Platform verifies the identity and the topological relationship during connection. If the verification is successful, IoT Platform establishes a logical connection with the sub-device and associates the logical connection with the physical connection of the gateway. The sub-device uses the same protocols as a directly connected device for

data upload and download. Gateway information is not required to be included in the protocols.

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

Add topological relationships of sub-devices

Upstream

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

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 ",
      " sign ": " xxxxxx ",
      " signmethod ": " hmacSha1 ",
      " timestamp ": " 1524448722  000 ",
      " clientId ": " xxxxxx "
    }
  ]
}
```

Response data format when using the Alink protocol

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

Parameter description

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

Parameter	Type	Description
params	List	Input parameters of the request.
deviceName	String	Device name. The value is the name of the sub-device.
productKey	String	Product ID. The value is the ID of the product to which the sub-device belongs.
sign	String	Signature. Signature algorithm: Sort all the parameters (except for `sign` and `signMethod`) that will be submitted to the server in lexicographical order, and then connect the parameters and values in turn (no connect symbols). Sign the signing parameters by using the algorithm specified by the signing method. For example, in the following request, sort the parameters in `params` in alphabetic order and then sign the parameters. `sign = hmac_md5 ( deviceSecret , clientId12 3deviceNametestprodu ctKey123timestamp152 4448722000 )`
signmethod	String	Signing method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.
clientId	String	Identifier of a sub-device. This parameter is optional and may have the same value as ProductKey or DeviceName.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error messages

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

Delete topological relationships of sub-devices

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

Upstream

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / topo / delete`
- Reply topic: `/ sys /{ productKey }/{ deviceName }/ thing / topo / delete_rep ly`

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 "
    }
  ]
}
```

Response data format when using the Alink protocol

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

Parameter description

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

Error messages

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

Obtain topological relationships of sub-devices

Upstream

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

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

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {}
```

```
}
```

Response data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " code ":  200 ,
  " data ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 "
    }
  ]
}
```

Parameter description

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

Error messages

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

Report new sub-devices

Upstream

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

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

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 "
    }
  ]
}
```

Response data format when using the Alink protocol

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

Parameter description

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

Error messages

Error code	Error message	Description
460	request parameter error	The request parameters are incorrect.
6250	product not found	The specified product to which the sub-device belongs does not exist.
6280	devicename not meet specs	The name of the sub-device is invalid. The device name must be 4 to 32 characters in length and can contain letters, digits, hyphens (-), underscores (_), at signs (@), periods (.), and colons (:).

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

Downstream

- **Rquest topic:** `/ sys /{ productKey }/{ deviceName }/ thing / topo / add / notify`
- **Reply topic:** `/ sys /{ productKey }/{ deviceName }/ thing / topo / add / notify_rep ly`

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

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " deviceName  1234 ",
      " productKey ": " 1234556554 "
    }
  ],
  " method ": " thing . topo . add . notify "
```

```
}
```

Response data format when using the Alink protocol

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

Parameter description

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

5.4 Connect and disconnect sub-devices

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

 Note:

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

Connect a sub-device to IoT Platform

 Note:

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

Upstream

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

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

Request message

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

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

Response message:

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

Request Parameters

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers, and the message IDs must be unique within the device.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	The unique identifier of the product to which the device belongs.
sign	String	Signature of the sub-device. Sub-devices use the same signature rules as gateways. Sign algorithm: 1. Sort all the parameters (except `sign` and `signMethod` and `cleanSessi on`) to be submitted to the server in alphabetical order, and then concatenate the parameters and values in turn (without any delimiters). 2. Then, sign the parameters by using the algorithm specified by signMethod and the DeviceSecret of the sub-device. Example: `sign = hmac_md5 ( deviceSecr et , clientId12 3deviceNam etestprodu ctKey123ti mestamp123 )`
signMethod	String	Sign method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.
clientId	String	The device identifier. The value of this parameter can be the value of ProductKey and DeviceName.
cleanSession	String	· A value of true indicates that when the sub-device is offline, messages sent based on QoS=1 method will be cleared. · A value of false indicates that when the sub-device is offline, messages sent based on QoS=1 method will not be cleared.

Response parameters

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

Error messages

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

Disconnect a sub-device from IoT Platform

Upstream

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

- Response topic: `/ ext / session /{ productKey }/{ deviceName }/ combine / logout_rep ly`

Note:

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

Request message:

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

Note:

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

Response message:

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

Request Parameters

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

Parameter	Type	Description
productKey	String	The unique identifier of the product to which the device belongs.

Response parameters

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

Error messages

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

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

5.5 Device properties, events, and services

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

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

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

Devices report properties

Report data (Do not parse/Custom)

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw`
- Response topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw_rep ly`

The raw data of a request message:

> Note:
> In raw data, the request `method thing.event.property.post` must be included.

```
0x02000000 7b00
```

Response message from IoT Platform:

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

Report Data (Alink JSON)

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

Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " Power ": {
      " value ": " on ",
      " time ":  1524448722  000
    },
    " WF ": {
      " value ":  23 . 6 ,
      " time ":  1524448722  000
    }
  }
}
```

Table 5-1: Request Parameters

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

Response message:

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

```
}
```

Table 5-2: Response parameters

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

Table 5-3: Error codes

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

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

Set device properties

Push data to devices (Do not parse/Custom)

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

Push data to devices (Alink JSON)

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

Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " temperatur  e ": " 30 . 5 "
  },
  " method ": " thing . service . property . set "
}
```

Table 5-4: Request Parameters

Parameter	Type	Description
id	String	The message ID. IoT Platform generates IDs for downstream messages.
version	String	The protocol version. Currently, the value is 1.0.

Parameter	Type	Description
params	Object	The property parameters. In the preceding request example, the property to be set is `{ " temperatur e ": " 30 . 5 " }`
method	String	Request method. The value is `thing . service . property . set .`

Response message:

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

Table 5-5: Response parameters

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

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

Devices report events

Report data (Do not parse/Custom)

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw`
- Response topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw_rep ly`

The raw data of a request message:

Note:

In raw data, the request `method thing.event.{tsl.event.identifier}.post` must be included. `tsl . event . identifier` indicates the event identifier in the TSL.

```
 0xff000000  7b00
```

Response message from IoT Platform:

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

Report Data (Alink JSON)

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

Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " value ": {
      " Power ": " on ",
      " WF ": " 2 "
    },
    " time ":  1524448722  000
  }
}
```

Table 5-6: Request Parameters

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

Parameter	Type	Description
value	Object	The event information. In the preceding request example, the events are: `{ " Power ": " on ", " WF ": " 2 " }`
time	Long	The UTC timestamp when the event occurs.

Response message:

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

Table 5-7: Response parameters

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

Examples

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

```
{
  " schema ": " https :// iot - tsl . oss - cn - shanghai . aliyuncs .
 com / schema . json ",
  " link ": "/ sys /${ productKey }/ airConditi  on / thing /",
  " profile ": {
    " productKey ": " al12345678  9 ",
    " deviceName ": " airConditi  on "
  },
```

```
  " events ": [
    {
      " identifier ": " alarm ",
      " name ": " alarm ",
      " desc ": " Fan   alarm ",
      " type ": " alert ",
      " required ":  true ,
      " outputData ": [
        {
          " identifier ": " errorCode ",
          " name ": " ErrorCode ",
          " dataType ": {
            " type ": " text ",
            " specs ": {
              " length ": " 255 "
            }
          }
        }
      ],
      " method ": " thing . event . alarm . post "
    }
  ]
}
```

Request message of reporting an event:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " value ": {
      " errorCode ": " error "
    },
    " time ":  1524448722  000
  }
}
```

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

Call device services

Push data to devices (Do not parse/Custom)

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / down_raw`
- Response topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / down_raw_r  eply`

Push data to devices (Alink JSON)

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

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

Service calling methods

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

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

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

Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " Power ": " on ",
    " WF ": " 2 "
  },
  " method ": " thing . service .{ tsl . service . identifier }"
}
```

Table 5-8: Request Parameters

Parameter	Type	Description
id	String	The message ID. IoT Platform generates IDs for downstream messages.
version	String	The protocol version. Currently, the value is 1.0.

Parameter	Type	Description
params	Map	The parameters used to call a service, including the identifier and value of the service. Example: `{ " Power ": " on ", " WF ": " 2 " }`
method	String	Request method. Note: `tsl . service . identifier` indicates the identifier of the service in TSL. For information about how to define a TSL, see Overview.

Response message:

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

Table 5-9: Response parameters

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

Examples

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

```
{
  " schema ": " https :// iotx - tsl . oss - ap - southeast - 1 .
 aliyuncs . com / schema . json ",
  " profile ": {
    " productKey ": " testProduc  t01 "
  },
  " services ": [
    {
      " outputData ": [
        {
          " identifier ": " OldWeight ",
          " dataType ": {
            " specs ": {
              " unit ": " kg ",
              " min ": " 0 ",
              " max ": " 200 ",
              " step ": " 1 "
            },
            " type ": " double "
          },
          " name ": " OldWeight "
        },
        {
          " identifier ": " CollectTim  e ",
          " dataType ": {
            " specs ": {
              " length ": " 2048 "
            },
            " type ": " text "
          },
          " name ": " CollectTim  e "
        }
      ],
      " identifier ": " SetWeight ",
      " inputData ": [
        {
          " identifier ": " NewWeight ",
          " dataType ": {
            " specs ": {
              " unit ": " kg ",
              " min ": " 0 ",
              " max ": " 200 ",
              " step ": " 1 "
            },
            " type ": " double "
          },
          " name ": " NewWeight "
        }
      ],
      " method ": " thing . service . SetWeight ",
      " name ": " SetWeight ",
      " required ":  false ,
      " callType ": " async "
    }
  ]
```

```
}
```

Request message of a service call:

```
{
  " method ": " thing . service . SetWeight ",
  " id ": " 105917531 ",
  " params ": {
    " NewWeight ":  100 . 8
  },
  " version ": " 1 . 0 . 0 "
}
```

Response message:

```
{
  " id ": " 105917531 ",
  " code ":  200 ,
  " data ": {
    " CollectTim  e ": " 1536228947  682 ",
    " OldWeight ":  100 . 101
  }
}
```

Gateway devices report data

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

Note:

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

Report data (Do not parse/Custom)

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw`
- Response topic: `/ sys /{ productKey }/{ deviceName }/ thing / model / up_raw_rep  ly`

The raw data of a request message:

Note:

In raw data, the request `method thing.event.property.pack.post` must be included.

```
 0xff000000  7b00
```

Response message from IoT Platform:

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

Report data (Alink JSON)

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

Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " properties ": {
      " Power ": {
        " value ": " on ",
        " time ":  1524448722  000
      },
      " WF ": {
        " value ": { },
        " time ":  1524448722  000
      }
    },
    " events ": {
      " alarmEvent  1 ": {
        " value ": {
          " param1 ": " on ",
          " param2 ": " 2 "
        },
        " time ":  1524448722  000
      },
      " alertEvent  2 ": {
        " value ": {
          " param1 ": " on ",
          " param2 ": " 2 "
        },
        " time ":  1524448722  000
      }
    },
    " subDevices ": [
      {
        " identity ": {
```

```
          " productKey ": "",
          " deviceName ": ""
        },
        " properties ": {
          " Power ": {
            " value ": " on ",
            " time ":  1524448722  000
          },
          " WF ": {
            " value ": { },
            " time ":  1524448722  000
          }
        },
        " events ": {
          " alarmEvent  1 ": {
            " value ": {
              " param1 ": " on ",
              " param2 ": " 2 "
            },
            " time ":  1524448722  000
          },
          " alertEvent  2 ": {
            " value ": {
              " param1 ": " on ",
              " param2 ": " 2 "
            },
            " time ":  1524448722  000
          }
        }
      }
    ]
  }
}
```

Table 5-10: Request Parameters

Parameter	Type	Description
id	String	The message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	The protocol version. Currently, the value is 1.0.
params	Object	The request parameters.
properties	Object	The information about a property, including property identifier, value and time when the property was generated.
events	Object	The information about an event, including event identifier, value and time when the event was generated.
subDevices	Object	The sub-device information.

Parameter	Type	Description
productKey	String	The ProductKey of a sub-device.
deviceName	String	The name of a sub-device.

Response message:

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

Table 5-11: Response parameters

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

5.6 Desired device property values

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

Obtain desired property values

Upstream data in Alink JSON format

A device requests the desired property values from IoT Platform.

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

Request format

```
{
    " id " : " 123 ",
    " version ": " 1 . 0 ",
    " params " : [
        " power ",
        " temperatur  e "
    ]
}
```

Response format

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

Table 5-12: Request parameters

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

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

Table 5-13: Response parameters

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

Table 5-14: Parameters in data

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

Clear desired property values

Upstream data in Alink JSON format

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

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

Request format

```
{
    " id " : " 123 ",
    " version ": " 1 . 0 ",
    " params " : [{
  " power ": {
   " version ":  1
  },
  " temperatur  e ": {
  }
 }
}
```

Response format

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

```
}
```

Table 5-15: Request parameters

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

Table 5-16: Parameters in params

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

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

Table 5-17: Response parameters

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

5.7 Disable and delete devices

Gateways can disable and delete their sub-devices.

Disable devices

Downstream

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

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {},
  " method ": " thing . disable "
```

Response message

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

Parameter description

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

Enable devices

Downstream

- Request Topic: `/ sys /{ productKey }/{ deviceName }/ thing / enable`
- Reply topic: `/ sys /{ productKey }/{ deviceName }/ thing / enable_rep ly`

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {},
  " method ": " thing . enable "
}
```

Response message

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

Parameter description

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

Delete devices

Downstream

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / delete`
- Reply topic: `/ sys /{ productKey }/{ deviceName }/ thing / delete_rep ly`

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {},
  " method ": " thing . delete "
}
```

Response message

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

Parameter description

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

5.8 Device tags

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

Report tags

Upstream

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " attrKey ": " Temperatur  e ",
      " attrValue ": " 36 . 8 "
    }
  ]
}
```

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently , the value can only be 1.0.
params	Object	Request parameters. This parameter can contain a maximum of 200 items.

Parameter	Type	Description
attrKey	String	Tag name. · Length: Up to 100 bytes. · Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_). · The tag name must start with an English letter or underscore (_).
attrValue	String	Tag value.
code	Integer	Result code. A value of 200 indicates the request is successful.

Error codes

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

Delete tags

Upstream

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

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " attrKey ": " Temperatur  e "
```

```
    }
  ]
}
```

Response message

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

Parameter description

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

Error messages

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

5.9 TSL model

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

- Request topic: `/ sys /{ productKey }/{ deviceName }/ thing / dsltemplat e / get`
- Reply topic: `/ sys /{ productKey }/{ deviceName }/ thing / dsltemplat e / get_reply`

The Allink data format of a request

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {}
}
```

The Allink data format of a response

```
{
  " id ": " 123 ",
  " code ":  200 ,
  " data ": {
    " schema ": " https :// iot - tsl . oss - cn - shanghai . aliyuncs
 . com / schema . json ",
    " link ": "/ sys / 1234556554 / airConditi  on / thing /",
    " profile ": {
      " productKey ": " 1234556554 ",
      " deviceName ": " airConditi  on "
    },
    " properties ": [
      {
        " identifier ": " fan_array_  property ",
        " name ": " Fan   array   property ",
        " accessMode ": " r ",
        " required ":  true ,
        " dataType ": {
          " type ": " array ",
          " specs ": {
            " size ": " 128 ",
            " item ": {
              " type ": " int "
            }
          }
        }
      }
```

```
    ],
    " events ": [
      {
        " identifier ": " alarm ",
        " name ": " alarm ",
        " desc ": " Fan   alert ",
        " type ": " alert ",
        " required ":  true ,
        " outputData ": [
          {
            " identifier ": " errorCode ",
            " name ": " Error   code ",
            " dataType ": {
              " type ": " text ",
              " specs ": {
                " length ": " 255 "
              }
            }
          }
        ],
        " method ": " thing . event . alarm . post "
      }
    ],
    " services ": [
      {
        " identifier ": " timeReset ",
        " name ": " timeReset ",
        " desc ": " Time   calibratio  n ",
        " inputData ": [
          {
            " identifier ": " timeZone ",
            " name ": " Time   zone ",
            " dataType ": {
              " type ": " text ",
              " specs ": {
                " length ": " 512 "
              }
            }
          }
        ],
        " outputData ": [
          {
            " identifier ": " curTime ",
            " name ": " Current   time ",
            " dataType ": {
              " type ": " date ",
              " specs ": {}
            }
          }
        ],
        " method ": " thing . service . timeReset "
      },
      {
        " identifier ": " set ",
        " name ": " set ",
        " required ":  true ,
        " desc ": " Set   properties ",
        " method ": " thing . service . property . set ",
        " inputData ": [
          {
            " identifier ": " fan_int_pr  operty ",
            " name ": " Integer   property   of   the   fan ",
            " accessMode ": " rw ",
            " required ":  true ,
```

```
            " dataType ": {
              " type ": " int ",
              " specs ": {
                " min ": " 0 ",
                " max ": " 100 ",
                " unit ": " g / ml ",
                " unitName ": " Millilitte  r "
              }
            }
          }
        ],
        " outputData ": []
      },
      {
        " identifier ": " get ",
        " name ": " get ",
        " required ":  true ,
        " desc ": " Get   properties ",
        " method ": " thing . service . property . get ",
        " inputData ": [
          " array_prop  erty ",
          " fan_int_pr  operty ",
          " batch_enum  _attr_id ",
          " fan_float_  property ",
           " fan_double  _property ",
          " fan_text_p  roperty ",
          " Maid  ",
          " batch_bool  ean_attr_i  d ",
          " fan_struct  _property "
        ],
        " outputData ": [
          {
            " identifier ": " fan_array_  property ",
            " name ": " Fan   array   property ",
            " accessMode ": " r ",
            " required ":  true ,
            " dataType ": {
              " type ": " array ",
              " specs ": {
                " size ": " 128 ",
                " item ": {
                  " type ": " int "
                }
              }
            }
          }
        ]
      }
    ]
  }
}
```

Parameter descriptions:

Parameter	Type	Description
id	String	Message ID. You need to define IDs for upstream messages using numbers , and the message IDs must be unique within the device.
version	String	Protocol version. Currently , the value is 1.0.
params	Object	Leave this parameter empty.
productKey	String	ProductKey. In the example, the ProductKey is 1234556554.
deviceName	String	Device name. In the example, the device name is airCondition.
data	Object	TSL model of the device. For more information, seeOverview

Error codes

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

5.10 Firmware update

For information about the firmware update, see OTA updates and Firmware update.

Report the firmware version

Upstream

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

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

Request message

```
{
  " id ":  1 ,
  " params ": {
    " version ": " 1 . 0 . 1 "
  }
}
```

Parameter description

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

Push firmware information

Downstream

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

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

Request message

```
{
  " code ": " 1000 ",
  " data ": {
    " size ":  432945 ,
    " version ": " 2 . 0 . 0 ",
    " url ": " https :// iotx - ota - pre . oss - cn - shanghai .
aliyuncs . com / nopoll_0 . 4 . 4 . tar . gz ? Expires = 1502955804
& OSSAccessK  eyId = XXXXXXXXXX  XXXXXXXXXX & Signature = XfgJu7P6DW
WejstKJgXJ  EH0qAKU % 3D & security - token = CAISuQJ1q6  Ft5B2yfSjI
pK6MGsyN1J  x5jo6mVnfB  glIPTvlvt5  D50Tz2IHtI  f3NpAusdsv
03nWxT7v4f  lqFyTINVAE  vYZJOPKGrG  R0DzDbDasu  mZsJbo4f %
2FMQBqEaXP  S2MvVfJ % 2BzLrf0ceu  sbFbpjzJ6x  aCAGxypQ12  iN % 2B
% 2Fr6 % 2F5gdc9FcQ  SkL0B8ZrFs  KxBltdUROF  bIKP % 2BpKWSKuGf
LC1dysQcO1  wEP4K % 2BkkMqH8Ui  c3h % 2Boy % 2BgJt8H2Pp  Hhd9NhXuV2
WMzn2 % 2FdtJOiTkn  xR7ARasaBq  helc4zqA % 2FPPlWgAKv  kXba7aIoo0
1fV4jN5JXQ  fAU8KLO8tR  jofHWmojNz  BJAAPpYSSy  3Rvr7m5efQ
rrybY1lLO6  iZy % 2BVio2VSZD  xshI5Z3McK  ARWct06MWV  9ABA2TTXXO
i40BOxuq % 2B3JGoABXC  54TOlo7 % 2F1wTLTsCU  qzzeIiXVOK  8CfNOkfTuc
MGHkeYeCdF  km % 2FkADhXAnr  nGf5a4FbmK  MQph2cKsr8  y8UfWLC6Iz
vJsClXTnbJ  BMeuWIqo5z  IynS1pm7gf % 2F9N3hVc6 % 2BEeIk0xfl
2tycsUpbL2  FoaGk6BAF8  hWSWYUXsv5  9d5Uk % 3D ",
```

```
    " md5 ": " 93230c3bde  425a9d7984  a594ac55ea  1e ",
    " sign ": " 93230c3bde  425a9d7984  a594ac55ea  1e ",
    " signMethod ": " Md5 "
  },
  " id ":  1507707025 ,
  " message ": " success "
}
```

Parameter description

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

Report update progress

Upstream

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

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

Request message

```
{
  " id ":  1 ,
  " params ": {
```

```
    " step ": "- 1 ",
    " desc ": " Firmware   update   has   failed .  No   firmware
 informatio  n   is   available ."
  }
}
```

Parameter description

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

Request firmware information from IoT Platform

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

Request message

```
{
  " id ":  1 ,
  " params ": {
    " version ": " 1 . 0 . 1 "
  }
}
```

Parameter description

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

Response message:

- Message with firmware information:

```
{
  " code ": " 1000 ",
  " data ": {
    " size ":  93796291 ,
    " sign ": " f8d85b250d  4d787a9f48  3d89a97473  48 ",
    " version ": " 1 . 0 . 1 . 9 . 20171112 . 1432 ",
    " url ": " https :// the_firmwa  re_url ",
    " signMethod ": " Md5 ",
    " md5 ": " f8d85b250d  4d787a9f48  3d89a97473  48 "
  },
  " id ":  8758548588  458 ,
  " message ": " success "
}
```

- No firmware file for update

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

5.11 Remote configuration

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

Device requests configuration information from IoT Platform

Upstream

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

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

Request message

```
{
  " id ":  123 ,
  " version ": " 1 . 0 ",
  " params ": {
    " configScop  e ": " product ",
    " getType ": " file "
  }
}
```

Response message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " code ":  200 ,
  " data ": {
    " configId ": " 123dagdah ",
    " configSize ":  1234565 ,
    " sign ": " 123214adfa  dgadg ",
    " signMethod ": " Sha256 ",
    " url ": " https :// iotx - config . oss - cn - shanghai .
 aliyuncs . com / nopoll_0 . 4 . 4 . tar . gz ? Expires = 1502955804
 & OSSAccessK  eyId = XXXXXXXXXX  XXXXXXXXXX & Signature = XfgJu7P6DW
 WejstKJgXJ  EH0qAKU % 3D & security - token = CAISuQJ1q6  Ft5B2yfSjI
 pK6MGsyN1J  x5jo6mVnfB  glIPTvlvt5  D50Tz2IHtI  f3NpAusdsv
 03nWxT7v4f  lqFyTINVAE  vYZJOPKGrG  R0DzDbDasu  mZsJbo4f %
 2FMQBqEaXP  S2MvVfJ % 2BzLrf0ceu  sbFbpjzJ6x  aCAGxypQ12  iN % 2B
 % 2Fr6 % 2F5gdc9FcQ  SkL0B8ZrFs  KxBltdUROF  bIKP % 2BpKWSKuGf
 LC1dysQcO1  wEP4K % 2BkkMqH8Ui  c3h % 2Boy % 2BgJt8H2Pp  Hhd9NhXuV2
 WMzn2 % 2FdtJOiTkn  xR7ARasaBq  helc4zqA % 2FPPlWgAKv  kXba7aIoo0
 1fV4jN5JXQ  fAU8KLO8tR  jofHWmojNz  BJAAPpYSSy  3Rvr7m5efQ
 rrybY1lLO6  iZy % 2BVio2VSZD  xshI5Z3McK  ARWct06MWV  9ABA2TTXXO
 i40BOxuq % 2B3JGoABXC  54TOlo7 % 2F1wTLTsCU  qzzeIiXVOK  8CfNOkfTuc
 MGHkeYeCdF  km % 2FkADhXAnr  nGf5a4FbmK  MQph2cKsr8  y8UfWLC6Iz
 vJsClXTnbJ  BMeuWIqo5z  IynS1pm7gf % 2F9N3hVc6 % 2BEeIk0xfl
 2tycsUpbL2  FoaGk6BAF8  hWSWYUXsv5  9d5Uk % 3D ",
    " getType ": " file "
  }
}
```

Parameter description

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

Parameter	Type	Description
version	String	Protocol version. Currently , the value is 1.0.
configScope	String	Configuration scope. Currently, IoT Platform supports only product dimension configuration. Value: product.
getType	String	Desired file type of the configuration. Currently, the supported type is file. Set the value to file.
configId	String	ID of the configuration.
configSize	Long	Size of the configuration file, in bytes.
sign	String	Signature value.
signMethod	String	Signing method. The supported signing method is Sha256.
url	String	The OSS address where the configuration file is stored .
code	Integer	Result code. A value of 200 indicates that the operation is successful, and other status codes indicate that the operation has failed.

Error codes

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

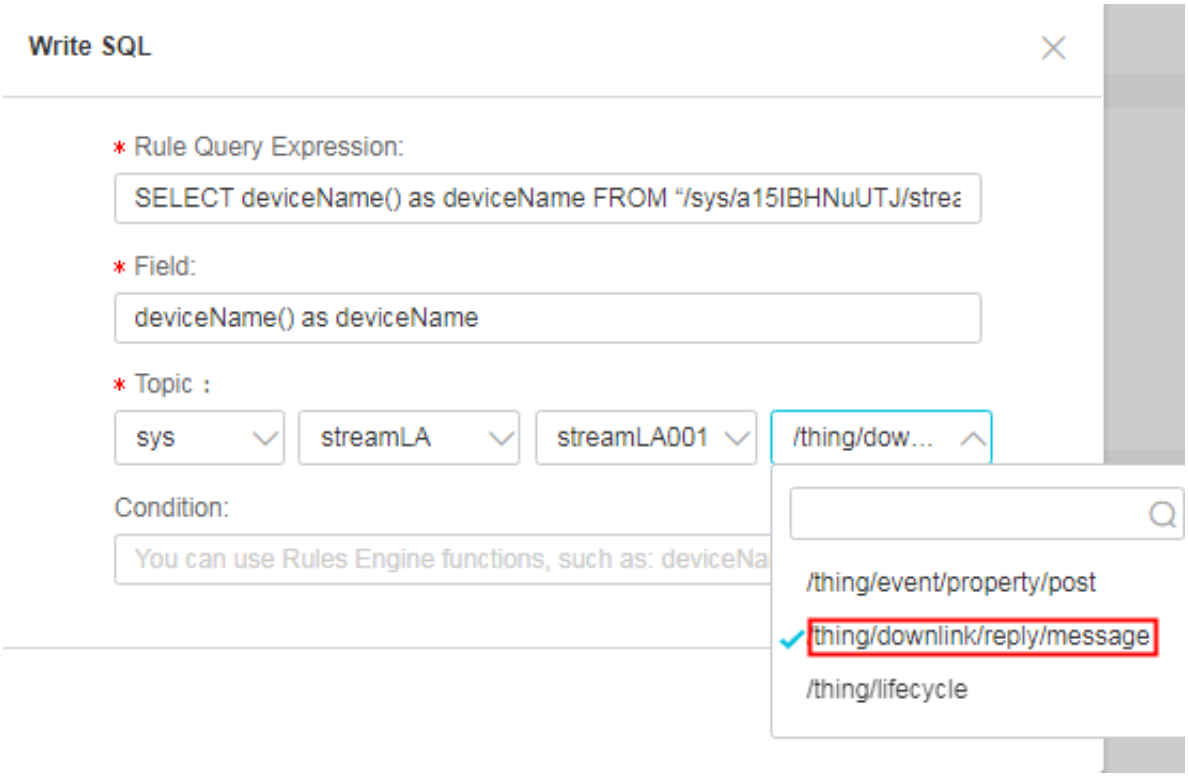
Push configurations in the IoT Platform console to devices.

Downstream

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

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

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



Request message:

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
```

```
  " params ": {
    " configId ": " 123dagdah ",
    " configSize ":  1234565 ,
    " sign ": " 123214adfa  dgadg ",
    " signMethod ": " Sha256 ",
    " url ": " https :// iotx - config . oss - cn - shanghai .
 aliyuncs . com / nopoll_0 . 4 . 4 . tar . gz ? Expires = 1502955804
 & OSSAccessK  eyId = XXXXXXXXXX  XXXXXXXXXX & Signature = XfgJu7P6DW
 WejstKJgXJ  EH0qAKU % 3D & security - token = CAISuQJ1q6  Ft5B2yfSjI
 pK6MGsyN1J  x5jo6mVnfB  glIPTvlvt5  D50Tz2IHtI  f3NpAusdsv
 03nWxT7v4f  lqFyTINVAE  vYZJOPKGrG  R0DzDbDasu  mZsJbo4f %
 2FMQBqEaXP  S2MvVfJ % 2BzLrf0ceu  sbFbpjzJ6x  aCAGxypQ12  iN % 2B
 % 2Fr6 % 2F5gdc9FcQ  SkL0B8ZrFs  KxBltdUROF  bIKP % 2BpKWSKuGf
 LC1dysQcO1  wEP4K % 2BkkMqH8Ui  c3h % 2Boy % 2BgJt8H2Pp  Hhd9NhXuV2
 WMzn2 % 2FdtJOiTkn  xR7ARasaBq  helc4zqA % 2FPPlWgAKv  kXba7aIoo0
 1fV4jN5JXQ  fAU8KLO8tR  jofHWmojNz  BJAAPpYSSy  3Rvr7m5efQ
 rrybY1lLO6  iZy % 2BVio2VSZD  xshI5Z3McK  ARWct06MWV  9ABA2TTXXO
 i40BOxuq % 2B3JGoABXC  54TOlo7 % 2F1wTLTsCU  qzzeIiXVOK  8CfNOkfTuc
 MGHkeYeCdF  km % 2FkADhXAnr  nGf5a4FbmK  MQph2cKsr8  y8UfWLC6Iz
 vJsClXTnbJ  BMeuWIqo5z  IynS1pm7gf % 2F9N3hVc6 % 2BEeIk0xfl
 2tycsUpbL2  FoaGk6BAF8  hWSWYUXsv5  9d5Uk % 3D ",
    " getType ": " file "
  },
  " method ": " thing . config . push "
}
```

Response message

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

Parameter description

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

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

5.12 Common codes on devices

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

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

6 Error codes for device SDKs

This topic describes error codes for device SDKs.

Common error codes

Table 6-1: Common error codes and descriptions

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

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

Table 6-2: Common error codes about parsing scripts

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

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

Table 6-3: Common error codes about TSL models

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

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

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

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

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

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

Device registration error codes

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

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

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

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

Unique-certificate-per-product authentication

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

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

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

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

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

Request topic: /sys/{productKey}/{deviceName}/thing/proxy/provisioning/product_register.

Error codes: 460, 400, 6250, 6288, 6600, 6292, and 6203.

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

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

Topology error codes

Add topological relationships

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

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

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

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

Delete topological relationships

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

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

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

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

Obtain topological relationships

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

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

The gateway reports a detected sub-device

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

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

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

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

Sub-device connection and disconnection error codes

A sub-device connects to IoT Platform

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

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

A sub-device automatically disconnects from IoT Platform

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

Error codes: 460, 520, and 500.

A sub-device is disconnected from IoT Platform by force

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

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

A sub-devices fails to send a message

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

Error code: 520.

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

Error code	Cause	Solution
427	The device frequently reconnects to IoT Platform. The same device certificate information is used to connect another device to IoT Platform. This disconnects the current device from IoT Platform.	Navigate to the Device Details page in the console and check when the device was most recently connected to IoT Platform. You can then determine whether the same device certificate information is used to connect another device to IoT Platform.
428	The number of sub-devices that you have added to the specified gateway has reached the upper limit. Currently, you can add up to 1,500 sub-devices to each gateway.	Check the number of sub-devices that you have added to the gateway.
521	The device has been deleted.	Navigate to the Devices page in the console and check whether the device has been deleted.
522	The device has been disabled.	Navigate to the Devices page in the console and check the status of the device.
520	An error occurred with the session between the sub-device and IoT Platform. · The specified session does not exist because the sub-device is not connected to IoT Platform, or the sub-device is already disconnected from IoT Platform. · The session exists, but the session is not established through the current gateway.	
6287	An error occurred while verifying the signature based on the ProductSecret or DeviceSecret.	Use the supported algorithms to calculate and verify the signature, as described in Connect and disconnect sub-devices.

Property, event, and service error codes

A device reports a property

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

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

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

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

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

The device reports an event

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

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

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

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

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

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

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

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

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

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

Error codes about desired device property values

Obtain desired property values

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

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

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

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

A device clears the desired property values

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

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

Device tag error codes

A device reports tag information

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

Error codes: 460 and 6100.

A device deletes tag information

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

Error codes: 460 and 500.

Error codes about obtaining TSL models

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

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

Error codes about querying firmware information

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

Note:

- In the AbstractAlinkJsonMessageConsumer class that provides methods for upgrading firmware, only the `DeviceOtaUpgradeReqConsumer` method returns error codes. Other methods do not return error codes or data.
- The topic used to query firmware information is the same as that used to return responses.

Error codes: 429, 9112, and 500.

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

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

Error codes about obtaining the configuration information

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

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

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

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

7 Error codes for sub-device development

This article describes errors that may occur during sub-device development.

Introduction

- When an IoT Platform service error occurs on a directly-connected device, the device is notified of the error when the TCP connection is closed.
- In the case that a communication error occurs on a sub-device connected to IoT Platform through a gateway and the gateway is still physically connected to IoT Platform, the gateway must send an error message through the gateway connection to notify the sub-device of the error.

Response format

When a communication error has occurred between a sub-device and IoT Platform , IoT Platform sends an MQTT error message to the gateway through the gateway connection.

The format of the topic varies depending on the scenario. The JSON format of the message content is as follows:

```
{
 id :  Message   ID   specified   in   the   request   parameters
 code :  Error   code  ( the   success   code   is   200 )
 message :  Error   message
}
```

Sub-device failed to go online

The error message is sent to topic `/ ext / session /{ gw_product  Key }/{ gw_deviceN  ame }/ combine / login_repl  y` .

Table 7-1: Error codes

Code	Message	Description
460	request parameter error	Invalid parameter format, for example, invalid JSON format or invalid authentication parameters.

Code	Message	Description
429	too many requests	Authentication requests have been denied. This error occurs when a device initiates authentication requests to IoT Platform too frequently or a sub-device has come online more than five times in one minute.
428	too many subdevices under gateway	The number of sub-devices connected to a gateway has reached the maximum. Currently, up to 1500 sub-devices can be connected to a gateway.
6401	topo relation not exist	No topological relationship has been established between the sub-device and the gateway.
6100	device not found	The specified sub-device does not exist.
521	device deleted	The sub-device has already been deleted.
522	device forbidden	The specified sub-device has been disabled.
6287	invalid sign	Authentication failed due to invalid username or password.
500	server error	An exception occurs on IoT Platform.

Sub-device automatically goes offline

The error message is sent to topic `/ ext / session /{ gw_product  Key }/{ gw_deviceN  ame }/ combine / logout_rep  ly` .

Table 7-2: Error codes

Code	Message	Description
460	request parameter error	Invalid parameter format, for example, invalid JSON format or invalid parameters.
520	device no session	The sub-device session does not exist, because the sub-device has gone offline or has never been connected to IoT Platform..
500	server error	An exception occurs on IoT Platform.

Sub-device forced to go offline

The error message is sent to topic `/ ext / error /{ gw_product  Key }/{ gw_deviceN  ame }`.

Table 7-3: Error codes

Code	Message	Description
427	device connect in elsewhere	Disconnection of current session. When another device uses the same device certificate of ProductKey, DeviceName, and DeviceSecret to connect to IoT Platform, the current device is forced offline.
521	device deleted	The device has been deleted.
522	device forbidden	The device has been disabled.
6401	topo relation not exist	The topological relationship between the sub-device and the gateway has been deleted.

Sub-device failed to send message

The error message is sent to topic /ext/error/{gw_productKey}/{gw_deviceName}.

Table 7-4: Error codes

Code	Message	Description
520	device session error	Sub-device session error. · The sub-device session does not exist. The sub -device is not connected to IoT Platform or has gone offline. · The sub-device session exists, however, the session is not established through the current gateway.