Alibaba Cloud
IoT Platform

Developer Guide (Devices)

Issue: 20190305

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 informatio n, 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 quickly connect them to the Cloud. As an alternative to SDKs, you can also use *Communications over Alink protocol* for development.

Prerequisites

Before developing devices, finish all console configurations, and obtain necessary informations such as the device details and topic information. For more information, see the User Guide.

Device SDKs

Select a device SDK according to the protocol and the features that you require. We recommend that you use C SDK as it provides more features.

Note:

If you have specific development requirements that cannot be met by the current SDKs, you can develop according to the *Communications over Alink protocol*.

	C SDK	Java SDK	Android SDK	iOS SDK	HTTP/2 SDK	General protocol
MQTT	√	√	√	√		
CoAP	√					
HTTP/HTTPS	√					
HTTP/2					√	
Other protocols						√
Device certification: unique-certificate-per-device authentication	√	√	√	√	√	√
Device certification: unique-certificate-per-product authentication	√		√			
OTA development	√					

	C SDK	Java SDK	Android SDK	iOS SDK	HTTP/2 SDK	General protocol
Connecting sub-devices to IoT Platform	√					
Device shadow	√	√	√			
Device development based on TSL	√		√			
Remote configuration	√					

Supported platforms

Click *here* to view and query the platforms supported by Alibaba Cloud IoT Platform.

If the platform you want to use is not supported by IoT Platform, please open an issue on the *Github* page.

Download SDKs

- C SDK

Version number	Release date	Develop t environ t	Downloa link	Updates
V2.2.1	2018/09/03	GNU make on 64-bit Linux	*RELEASED_V 2.2.1*	Added supports for connecting devices to WiFi and using open-source applications to locally control devices. - Added supports for countdown routine before devices go offline. - Added supports for OTA using iTls to download firmware files.

Version number	Release date	Developı t environı t	Downloa link	Updates
V2.1.0	2018/03/31	GNU make on 64-bit Linux	*RELEASED_V2_10_20180331.zip*	Added support for CMake: You can use QT or VS2017 on Linux or Windows to open a project and compile software in CMake compiling method. - Added support for TSL model definition on IoT Platform: You can set `FEATURE_CMP_ENABLED = y` and `FEATURE_DM_ENABLED = y` to define TSL models to provide API operations for properties, events, and services. - Added support for unique-certificate-per-product: You can set `FEATURE_SUPPORT_PRODUCT_SECRET = y` to enable unique-certificate-per-product authentication and streamline the production queuing process. - Added support for iTLS: You can set `FEATURE_MQTT_DIRECT_NOTLS = y` and `FEATURE_MQTT_DIRECT_NOITLS = n` to enable ID^2 encryption. You can use iTLS to establish data connections to enhance security and reduce memory consumption. - Added support for remote configuration: You can set `FEATURE_SERVICE_OTA_ENABLED = y` and `FEATURE_SERVICE_COTA_ENABLED = y` to enable the cloud to push configuration information to devices. - Optimized sub-device management of gateways: Added some features.

- Java SDK

Supported protocol	Update history	Download link
MQTT	2017-05-27: Added support for device authentication in the China (Shanghai) region. Added the device shadow demo on the Java client.	*iotx-sdk-mqtt-java*: The Java version that supports MQTT is only a demo of open-source library implementation. It is used only for reference.

  Instructions: See *Java SDK*.

- iOS SDK

  Download link:

  - *https://github.com/CocoaPods/Specs.git*
  - *https://github.com/aliyun/aliyun-specs.git*

  Instructions:*iOS SDK*

- HTTP/2 SDK

  Download link: *iot-http2-sdk-demo*.

- General protocol

  Instructions: See *General protocol*.

- Other open-source libraries

  Download link: *https://github.com/mqtt/mqtt.github.io/wiki/libraries*

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.

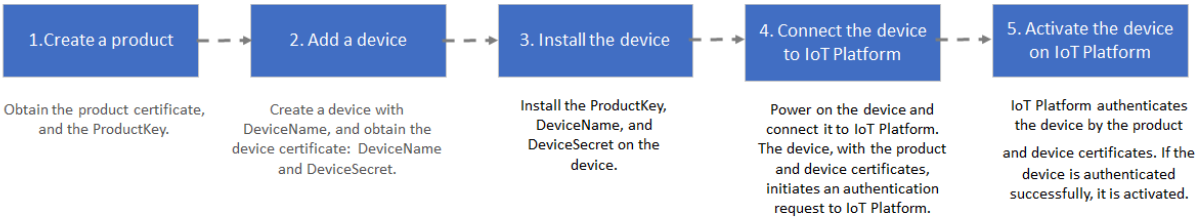
2.2 Unique-certificate-per-device authentication

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:



1.Create a product: Obtain the product certificate, and the ProductKey.

2. Add a device: Create a device with DeviceName, and obtain the device certificate: DeviceName and DeviceSecret.

3. Install the device: Install the ProductKey, DeviceName, and DeviceSecret on the device.

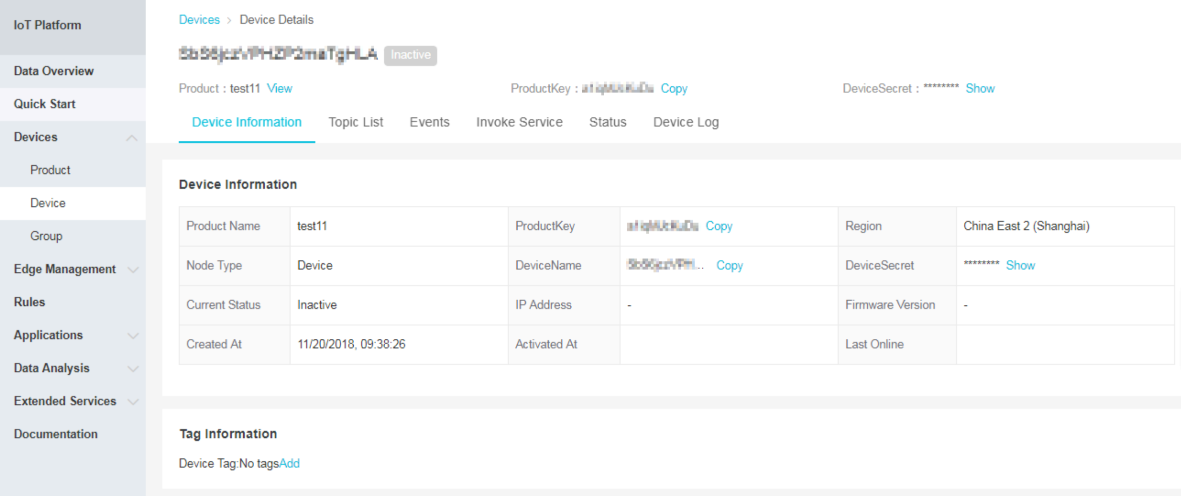
4. Connect the device to IoT Platform: Power on the device and connect it to IoT Platform. The device, with the product and device certificates, initiates an authentication request to IoT Platform.

5. Activate the device on IoT Platform: IoT Platform authenticates the device by the product and device certificates. If the device is authenticated successfully, it is activated.

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.



[redacted] Inactive

Product : test11 View   ProductKey : [redacted] Copy   DeviceSecret : ******** Show

Device Information

Product Name	test11	ProductKey	[redacted] Copy	Region	China East 2 (Shanghai)
Node Type	Device	DeviceName	[redacted].. Copy	DeviceSecret	******** Show
Current Status	Inactive	IP Address	-	Firmware Version	-
Created At	11/20/2018, 09:38:26	Activated At		Last Online	

Tag Information

Device Tag:No tagsAdd

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

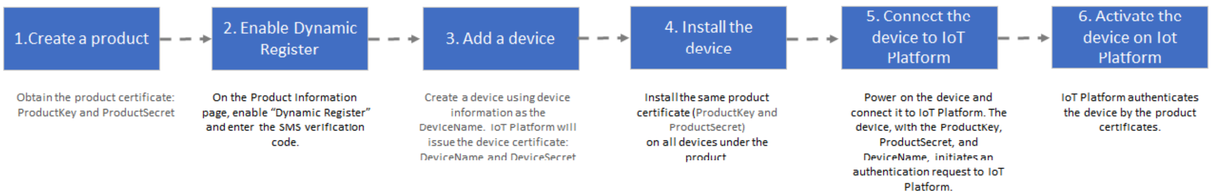
Using unique-certificate-per-product authentication method requires that devices of a product have been installed with a same firmware in which a product certificate (ProductKey and ProductSecret) has been installed. When a device initiates an activation request, IoT Platform authenticates the product 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 Registration` to reject authentication requests from new devices.
- The unique-certificate-per-product method is used to obtain the DeviceSecret of devices from IoT Platform. The DeviceSecret is only issued once. The device stores the DeviceSecret for future use.

Workflow of unique-certificate-per-product authentication:

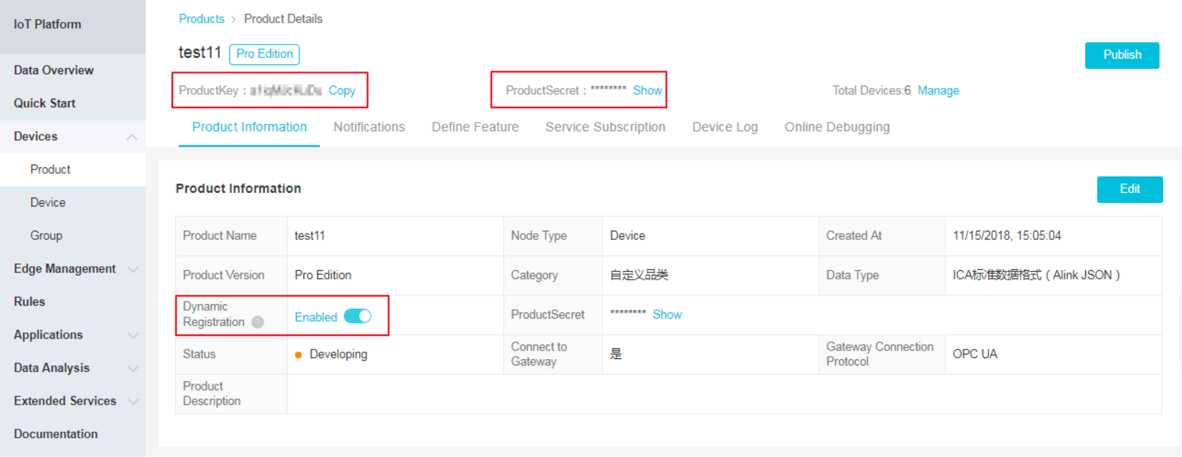

1.Create a product
Obtain the product certificate: ProductKey and ProductSecret
2. Enable Dynamic Register
On the Product Information page, enable "Dynamic Register" and enter the SMS verification code.
3. Add a device
Create a device using device information as the DeviceName. IoT Platform will issue the device certificate: DeviceName and DeviceSecret
4. Install the device
Install the same product certificate (ProductKey and ProductSecret) on all devices under the product
5. Connect the device to IoT Platform
Power on the device and connect it to IoT Platform. The device, with the ProductKey, ProductSecret, and DeviceName, initiates an authentication request to IoT Platform.
6. Activate the device on Iot Platform
IoT Platform authenticates the device by the product certificates.

Procedure

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

   Note:

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



ProductKey : [redacted] Copy

ProductSecret : ******** Show

Total Devices:6 Manage

Product Information

Product Name	test11	Node Type	Device	Created At	11/15/2018, 15:05:04
Product Version	Pro Edition	Category	自定义品类	Data Type	ICA标准数据格式（Alink JSON）
Dynamic Registration	Enabled	ProductSecret	******** Show		
Status	● Developing	Connect to Gateway	是	Gateway Connection Protocol	OPC UA
Product Description					

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 device authentication methods: the MQTT client and the HTTPS protocol.

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.

Connect the MQTT client to IoT Platform using defined domain names

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 ","`
	`clientId12  345deviceN  amedevicep  roductKeyp`
	`ktimestamp  789 "). toHexStrin  g (); // The`
	`  toHexStrin  g ()  function   converts   a`
	`  binary   string   to   a   hexadecima  l`
	`string .  The   string   is   case - insensitiv`
	`e .`
	
	The encrypted password is as follows:
	
	`FAED82A3D6  02B37EB0EA  8B7892E24A  477E851A14`

Use HTTPS for device authentication. The authentication URL is `https :// iot - auth . ${ YourRegion Id }. aliyuncs . com / auth / devicename` . Replace *${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 over WebSocket connections

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 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*.

3.5 Establish communication over the CoAP protocol

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.

CoAP-based connection procedure

The following figure shows the procedure for connecting NB-IoT devices to IoT Platform.

The procedure is as follows:

1. Integrate an Alibaba Cloud IoT Platform SDK into the NB-IoT module of device clients. Specifically, in the IoT Platform console, you need to register products and devices, obtain the unique device certificates (that is, the ProductKey, DeviceName , and DeviceSecret components), and then install the certificates to the devices.
2. Establish a connection over your target carriers' cellular networks for NB-IoT devices to connect to IoT Platform. We recommend that you contact your local carrier to make sure that the NB-IoT network is available in the region where your devices are located.
3. After the devices are connected to IoT Platform, a machine-to-machine (M2M) platform manages the data traffic and fees incurred by the NB-IoT devices. The M2M platform is operated by your specified carrier.
4. Over the CoAP/UDP protocol, devices send data to IoT Platform in real time. IoT Platform is a secure service that can connect and manage data for hundreds of millions of NB-IoT devices. Then, through Rules Engine of IoT Platform, the device data can be forwarded to other Alibaba Cloud product instances, such as big data products, ApsaraDB for RDS, Table Store, and other products.
5. Use the APIs and message pushing services provided by IoT Platform to forward data to supported service instances and quickly integrate device assets and actual applications.

Establish DTLS connections

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

   Note:

   - *`${YourProductKey}`*: Replace this variable with the ProductKey value of the device.
   - *`${port}`*: The port number. Set the port number to 5684 for DTLS connections.
2. Download the *root certificate*.
3. Authenticate the device. Call `auth` to authenticate the device and obtain the device token. Token information is required when the device sends data to IoT Platform.

   Request message:

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5684
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"ZG1EvTEa7NN","deviceName":"NlwaSPXsCpTQuh8FxBGH","clientId":"mylight1000002","sign":"bccb3d2618afe74b3eab12b94042f87b"}
```

Table 3-1: Parameter description

Parameter	Description
Method	The request method. The supported method is POST.
URL	`/auth`.
Host	The endpoint address. The endpoint format is `${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com`. Replace the variable *`${YourProductKey}`* with the ProductKey value of the device.
Port	Set the value to `5684`.
Accept	The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.
Content-Format	The encoding format of the data that the device sends to IoT Platform. Currently, application/json and application/cbor are supported.

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

Table 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. Value options: · `0` : After receiving a request from the device client, the server processes data and then returns the result with an acknowledgment (ACK). · `1` : After receiving a request from the client, 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.
sign	Yes	The 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.
signmethod	No	The algorithm type. The supported types are hmacmd5 and hmacsha1.

Parameter	Required	Description
clientId	Yes	The device client ID, 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:

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

Table 3-3: 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.
4.15	Unsupported Content-Format	no payload	The value of Content-Format parameter is not in a supported format.
5.00	Internal Server Error	no payload	The authentication request is timed out or an error occurred on the authentication server.

4. The device sends data.

   The device publishes data to a specified topic. In the IoT Platform console, on the Topic Categories tab page of the product, you can create topic categories.

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

   Request message:

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

Table 3-4: Request parameters

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

Parameter	Required	Description
Content-Format	Yes	The encoding format of the data that the device sends to IoT Platform. The server does not verify this parameter. 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.

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 ":" a1NUjcVkHZ  S "," deviceName ":"
ff1a11e7c0  8d4b3db2b1  500d8e0e55 "," clientId ":" a1NUjcVkHZ
```

```
 S & ff1a11e7c0  8d4b3db2b1  500d8e0e55 "," sign ":" F9FD53EE0C
 D010FCA40D  14A9FEAB81  E0 ", " seq ":" 10 "}
```

For more information about parameters (except for `Port` parameter, where the port for this method is 5682) and payload content, see *Parameter description*.

Response example:

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

Table 3-5: Response parameters

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

3. The device sends data.

Request message:

```
 POST  / topic /${ topic }
 Host : ${ 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 3-6: Request parameters

Parameter	Required	Description
Method	Yes	The request method. The supported 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.

Parameter	Required	Description
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.
payload	Yes	The encrypted data that is to be sent. Encrypt the data using the Advanced Encryption Standard (AES) algorithm.

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

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

3.6 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*.

3.7 Establish communication over the HTTPS protocol

IoT Platform supports HTTPS connections. It does not support HTTP connections.

Description

- The HTTPS server endpoint is `https :// iot - as - http . cn - shanghai . aliyuncs . com`.
- Currently, only the region cn-shanghai supports HTTPS connections.
- Only the HTTPS protocol is supported.
- The standards for HTTPS topics are the same as the standards for MQTT topics in *MQTT standard*. Devices send data to IoT Platform through `https :// iot - as - http . cn - shanghai . aliyuncs . com / topic /${ topic }`. The value of ${topic} can be the same topic used in MQTT communications. You cannot specify parameters in the format of `? query_String = xxx`.
- The size of data from devices is limited to 128 KB.

Procedure

1. Connect to the HTTPS server.

   The endpoint address: `https :// iot - as - http . cn - shanghai . aliyuncs . com`

2. Authenticate the device to get the device token.

   Device authentication request message example:

```
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: Parameter description

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

Parameter	Description
URL	`/auth` URL address. Only HTTPS is supported.
Host	The endpoint address: `iot-as-http.cn-shanghai.aliyuncs.com`
Content-Type	The format of the data that the device sends to IoT Platform. Only application/json is supported.
body	The device information for authentication, in JSON format. For more information, see the following table *Parameters in body*.

Table 3-8: Parameters in body

Parameter	Required?	Description
productKey	Yes	The unique identifier of the product to which the device belongs. You can obtain this information on the device details page in the IoT Platform console.
deviceName	Yes	The device name. You can obtain this informatio n on the device details page in the IoT Platform console.
clientId	Yes	The device client ID. It can be any string up to 64 characters in length. We recommend that you use either the MAC address or the SN code as the clientId.
timestamp	No	Timestamp. The request is valid within 15 minutes after the timestamp is created.

Parameter	Required?	Description
sign	Yes	Signature. The signature algorithm is in the format of `hmacmd5 ( DeviceSecr et , content )`. The value of `content` is a string that is built by sorting and concatenating all the parameters (except `version` , `sign` , and `signmethod`) that need to be submitted to the server in alphabetical order, without any delimiters. 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 hexadecimal data.
signmethod	No	The algorithm type. The supported types are hmacmd5 and hmacsha1. The default value is hmacmd5.
version	No	The version. If you leave this blank, the value is default.

Response example:

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

Note:

- The returned token can be cached locally.

- Token information is required every time when the device reports data to IoT Platform. If the token is lost or expires, initiate a device authentication request again to obtain a new token.

Table 3-9: Error codes

Code	Message	Description
10000	common error	Unknown error.
10001	param error	A parameter exception occurred during the request.
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.

3. The device sends data to IoT Platform.

   The device send data to the specified topic.

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

   For example, a topic category is `/${ YourProduc tKey }/${ YourDevice Name }/ pub`. If a device name is device123, and its product key is a1GFjLP3xxC, the device sends data through `https :// iot - as - http . cn - shanghai . aliyuncs . com / topic / a1GFjLP3xx C / device123 / pub`.

   Request message format:

```
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 request method is POST.

Parameter	Description
URL	`/ topic /${ topic }`. Replace `${ topic }` with the topic for receiving device data. Only HTTPS is supported.
Host	The endpoint address: `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 when using the `auth` interface to authenticate the device.
body	The data content sent to ${topic}, which is in binary byte[] array format and encoded with UTF-8.

Response example:

```
 body :
{
  " code ":  0 , //  the   status   code
  " message ": " success ", //  the   result   message
  " info ": {
    " messageId ":  8926876279  16247040 ,
    " Data ":  byte  []/ UTF - 8   encoded   data ,  and   possibly
   empty
  }
}
```

Table 3-11: Error codes

Code	Message	Description
10000	common error	Unknown error.
10001	param error	A parameter exception occurred during the request.
20001	token is expired	The token is invalid. Call `auth` to authenticate the device again to obtain a new token.
20002	token is null	The request header contains no token information.
20003	check token error	Failed to identify the device based on the token. Call `auth` to authenticate the device again and obtain a new token.
30001	publish message error	An error occurred while publishing data.

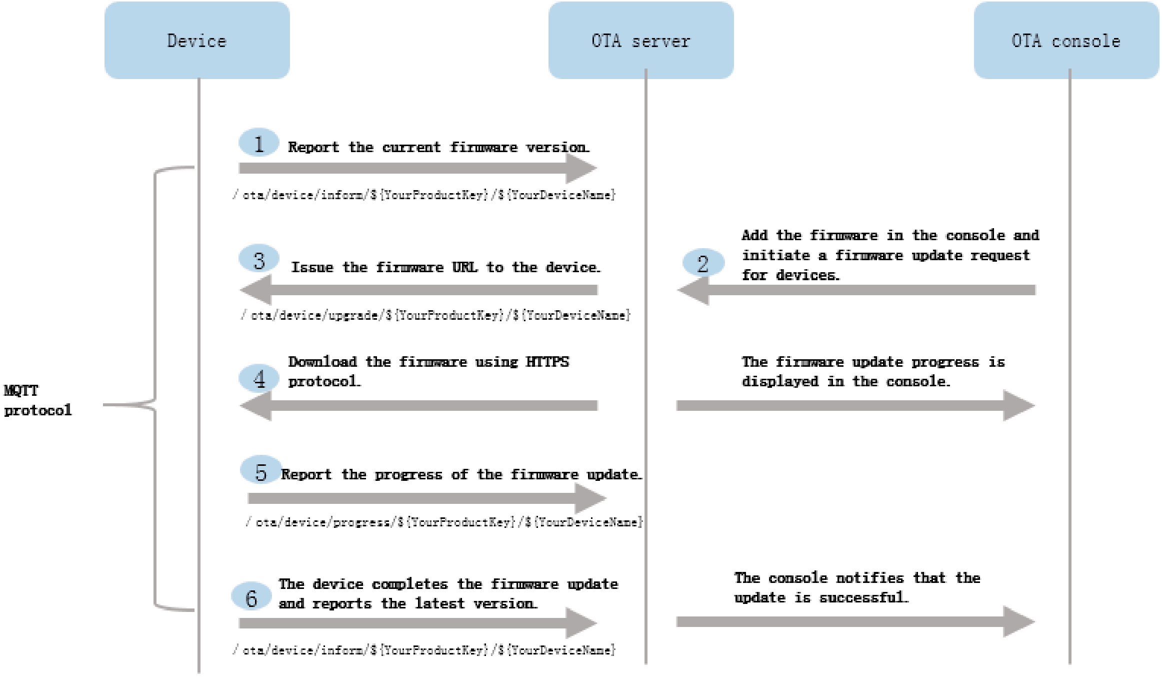
Code	Message	Description
40000	request too many	Too many requests. The throttling policy limits the number of requests.

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:


Device
OTA server
OTA console
1 Report the current firmware version.
/ota/device/inform/${YourProductKey}/${YourDeviceName}
2 Add the firmware in the console and initiate a firmware update request for devices.
3 Issue the firmware URL to the device.
/ota/device/upgrade/${YourProductKey}/${YourDeviceName}
4 Download the firmware using HTTPS protocol.
The firmware update progress is displayed in the console.
MQTT protocol
5 Report the progress of the firmware update.
/ota/device/progress/${YourProductKey}/${YourDeviceName}
6 The device completes the firmware update and reports the latest version.
/ota/device/inform/${YourProductKey}/${YourDeviceName}
The console notifies that the update is successful.

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`

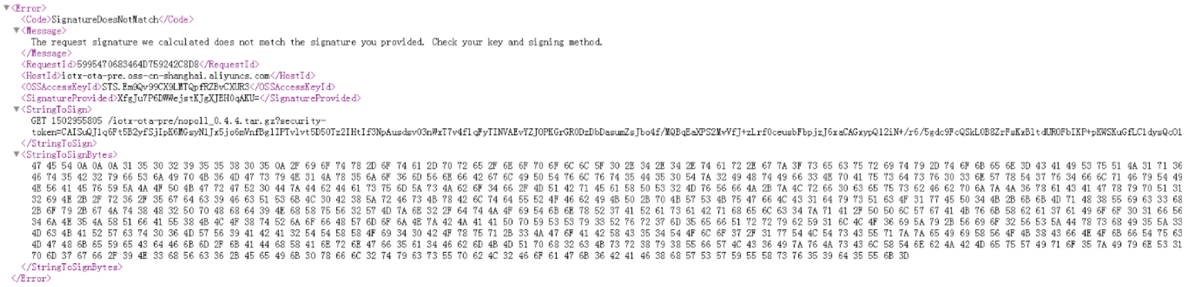
`Name` }. If the reported version is the same as the version defined in the firmware update file, then the update is successful.

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.

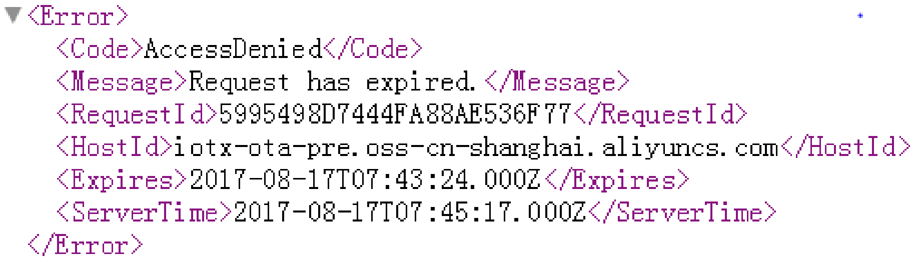
Errors

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



```
▼<Error>
   <Code>SignatureDoesNotMatch</Code>
 ▼<Message>
     The request signature we calculated does not match the signature you provided. Check your key and signing method.
   </Message>
   <RequestId>5995470683464D759242C8D8</RequestId>
   <HostId>iotx-ota-pre.oss-cn-shanghai.aliyuncs.com</HostId>
   <OSSAccessKeyId>STS.Em9Qv99CX9LMTQpfRZBvCXUR3</OSSAccessKeyId>
   <SignatureProvided>XfgJu7P6DWWejstKJgXJEH0qAKU=</SignatureProvided>
 ▼<StringToSign>
     GET 1502955805 /iotx-ota-pre/nopoll_0.4.4.tar.gz?security-
     token=CAISuQJ1q6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBglIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f/MQBqEaXPS2MvVfJ+zLrf0ceusbFbpjzJ6xaCAGxypQ12iN+/r6/5gdc9FcQSkL0B8ZrFsKxBltdUROFbIKP+pKWSKuGfLC1dysQcO1
   </StringToSign>
 ▼<StringToSignBytes>
     47 45 54 0A 0A 0A 31 35 30 32 39 35 35 38 30 35 0A 2F 69 6F 74 78 2D 6F 74 61 2D 70 72 65 2F 6E 6F 70 6F 6C 6C 5F 30 2E 34 2E 34 2E 74 61 72 2E 67 7A 3F 73 65 63 75 72 69 74 79 2D 74 6F 6B 65 6E 3D 43 41 49 53 75 51 4A 31 71 36
     46 74 35 42 32 79 66 53 6A 49 70 4B 36 4D 47 73 79 4E 31 4A 78 35 6A 6F 36 6D 56 6E 66 42 67 6C 49 50 54 76 6C 76 74 35 44 35 30 54 7A 32 49 48 74 49 66 33 4E 70 41 75 73 64 73 76 30 33 6E 57 78 54 37 76 34 66 6C 71 46 79 54 49
     4E 56 41 45 76 59 5A 4A 4F 50 4B 47 72 47 52 30 44 7A 44 62 44 61 73 75 6D 5A 73 4A 62 6F 34 66 2F 4D 51 42 71 45 61 58 50 53 32 4D 76 56 66 4A 2B 7A 4C 72 66 30 63 65 75 73 62 46 62 70 6A 7A 4A 36 78 61 43 41 47 78 79 70 51 31
     32 69 4E 2B 2F 72 36 2F 35 67 64 63 39 46 63 51 53 6B 4C 30 42 38 5A 72 46 73 4B 78 42 6C 74 64 55 52 4F 46 62 49 4B 50 2B 70 4B 57 53 4B 75 47 66 4C 43 31 64 79 73 51 63 4F 31 77 45 50 34 4B 2B 6B 6B 4D 71 48 38 55 69 63 33 68
     2B 6F 79 2B 67 4A 74 38 48 32 50 70 48 68 64 39 4E 68 58 75 56 32 57 4D 7A 6E 32 2F 64 74 4A 4F 69 54 6B 6E 78 52 37 41 52 61 73 61 42 71 68 65 6C 63 34 7A 71 41 2F 50 50 6C 57 67 41 4B 76 6B 58 62 61 37 61 49 6F 6F 30 31 66 56
     34 6A 4E 35 4A 58 51 66 41 55 38 4B 4C 4F 38 74 52 6A 6F 66 48 57 6D 6F 6A 4E 7A 42 4A 41 41 50 70 59 53 53 79 33 52 76 72 37 6D 35 65 66 51 72 72 79 62 59 31 6C 4C 4F 36 69 5A 79 2B 56 69 6F 32 56 53 5A 44 78 73 68 49 35 5A 33
     4D 63 4B 41 52 57 63 74 30 36 4D 57 56 39 41 42 41 32 54 54 58 58 4F 69 34 30 42 4F 78 75 71 2B 33 4A 47 6F 41 42 58 43 35 34 54 4F 6C 6F 37 2F 31 77 54 4C 54 73 43 55 71 7A 7A 65 49 69 58 56 4F 4B 38 43 66 4E 4F 6B 66 54 75 63
     4D 47 48 6B 65 59 65 43 64 46 6B 6D 2F 6B 41 44 68 58 41 6E 72 6E 47 66 35 61 34 46 62 6D 4B 4D 51 70 68 32 63 4B 73 72 38 79 38 55 66 57 4C 43 36 49 7A 76 4A 73 43 6C 58 54 6E 62 4A 42 4D 65 75 57 49 71 6F 35 7A 49 79 6E 53 31
     70 6D 37 67 66 2F 39 4E 33 68 56 63 36 2B 45 65 49 6B 30 78 66 6C 32 74 79 63 73 55 70 62 4C 32 46 6F 61 47 6B 36 42 41 46 38 68 57 53 57 59 55 58 73 76 35 39 64 35 55 6B 3D
   </StringToSignBytes>
 </Error>
```

- 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 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 user client 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 user client 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_productKey}/{gw_deviceName}/combine/login_reply.

Table 5-1: 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_productKey}/{gw_deviceName}/combine/logout_reply.

Table 5-2: 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/session/{gw_productKey}/{gw_deviceName}/combine/logout_reply.

Table 5-3: 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.

Sub-device failed to send message

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

Table 5-4: 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.

6 Develop devices based on Alink Protocol

6.1 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

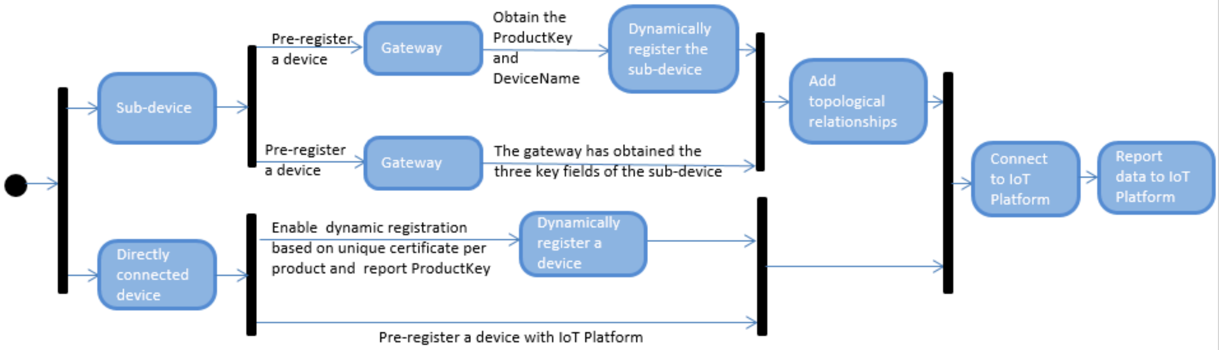
As shown in the following figure, devices can be connected to IoT Platform as directly connected devices or sub-devices. The connection process involves the following key steps: authenticate the device, establish a connection, and the device reports data to IoT Platform.

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.

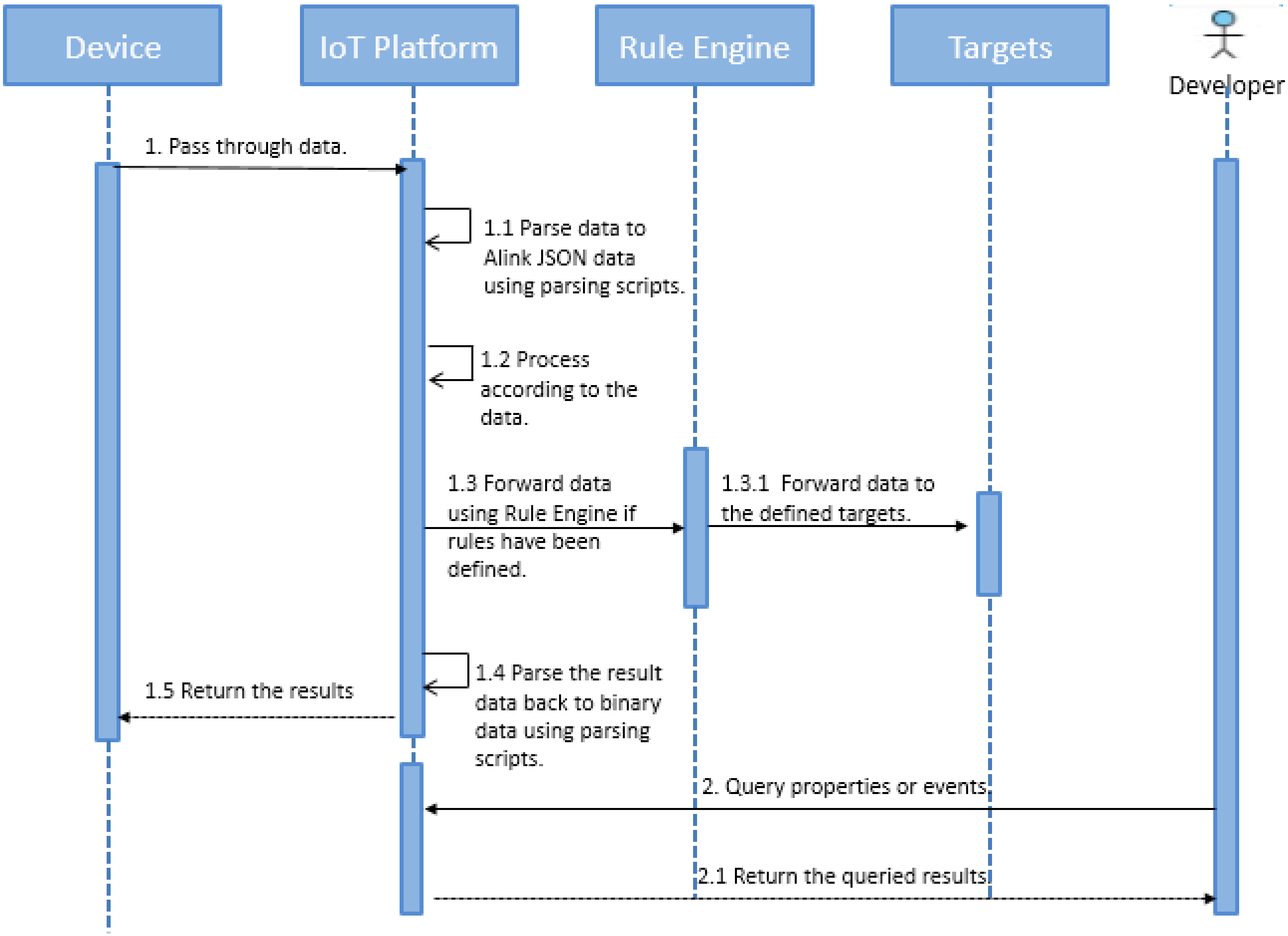
Sub-devices connect to IoT Platform through their gateways. Sub-devices can be connected to IoT Platform by using the following methods:

- 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.


Sub-device
Directly connected device
Pre-register a device
Gateway
Obtain the ProductKey and DeviceName
Dynamically register the sub-device
Pre-register a device
Gateway
The gateway has obtained the three key fields of the sub-device
Enable dynamic registration based on unique certificate per product and report ProductKey
Dynamically register a device
Pre-register a device with IoT Platform
Add topological relationships
Connect to IoT Platform
Report data to IoT Platform

Devices report properties or events

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


Device
IoT Platform
Rule Engine
Targets
Developer
1. Pass through data.
1.1 Parse data to Alink JSON data using parsing scripts.
1.2 Process according to the data.
1.3 Forward data using Rule Engine if rules have been defined.
1.3.1 Forward data to the defined targets.
1.4 Parse the result data back to binary data using parsing scripts.
1.5 Return the results
2. Query properties or events
2.1 Return the queried results

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:

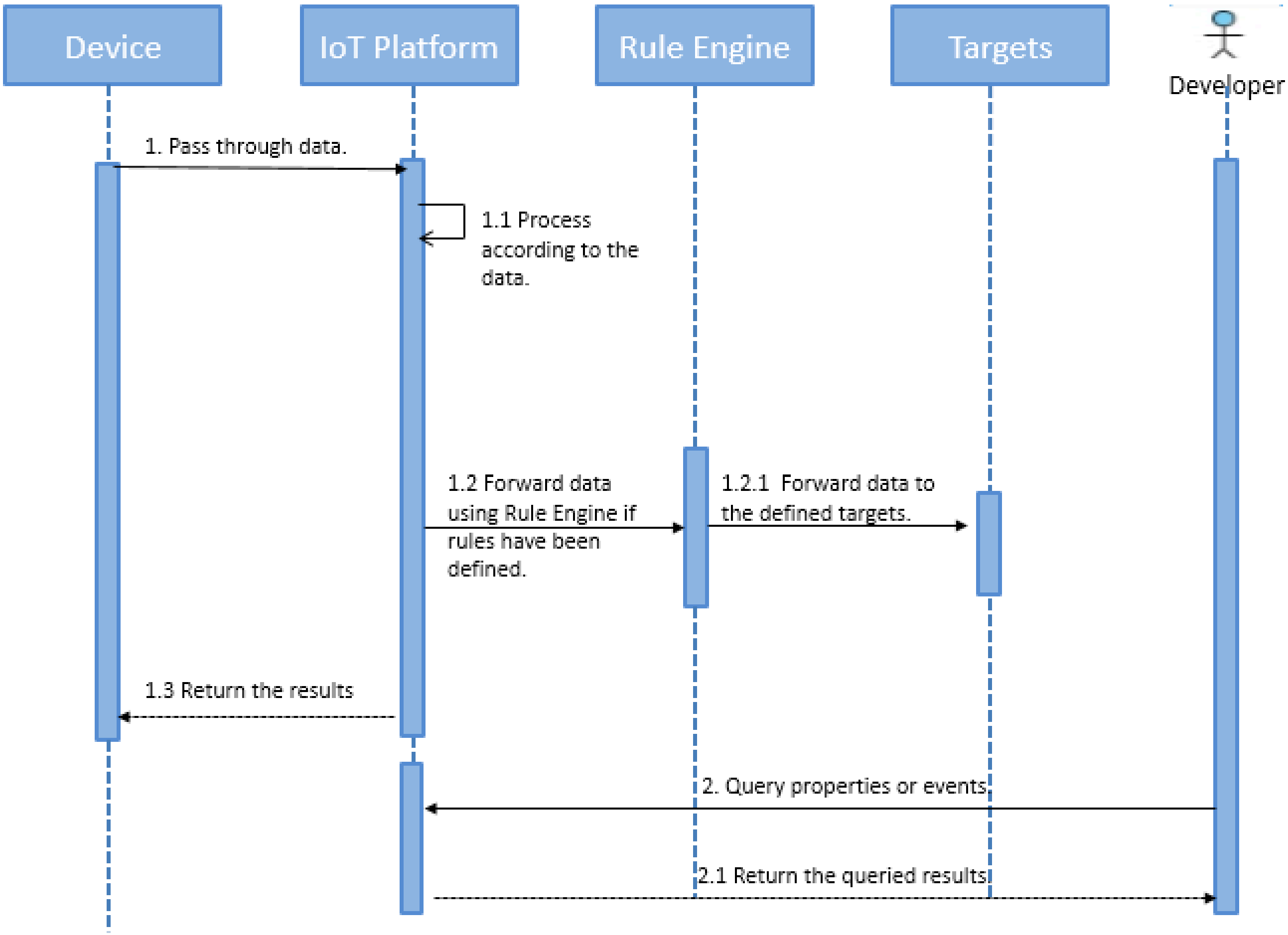
If you have configured rules for data forwarding, the Alink JSON data will be forwarded to the targets according to the rules.

- 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 `protocolTo RawData` 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


Device
IoT Platform
Rule Engine
Targets
Developer
1. Pass through data.
1.1 Process according to the data.
1.2 Forward data using Rule Engine if rules have been defined.
1.2.1 Forward data to the defined targets.
1.3 Return the results
2. Query properties or events
2.1 Return the queried results

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.

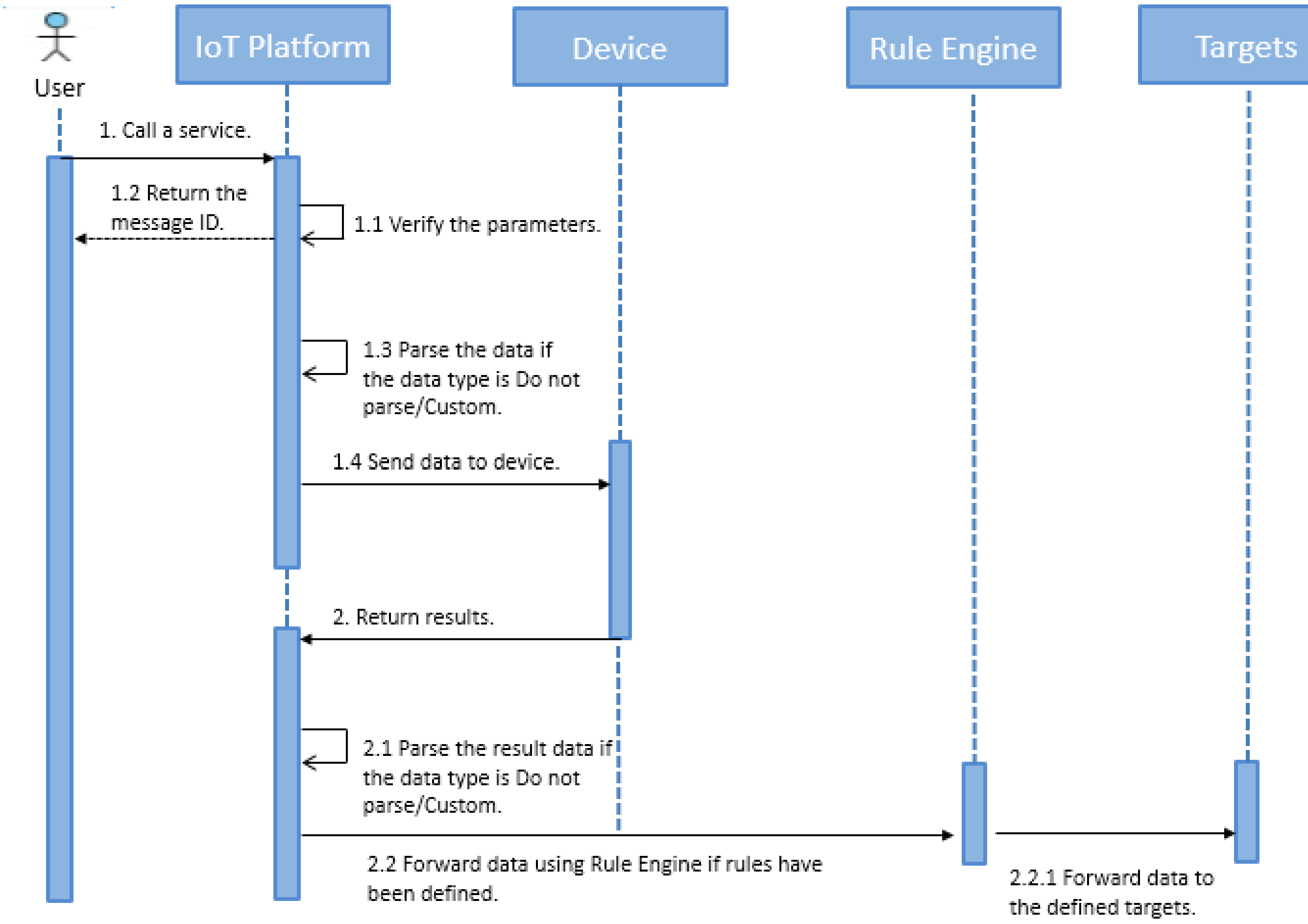
> Note:
>
> If you have configured rules for data forwarding, the data will be forwarded to the targets according to the rules.
>
> - When you configure SQL statements for rules, to obtain the device properties, specify the data topic to be `/ sys /{ productKey }/{ deviceName }/ thing / event / property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


User
IoT Platform
Device
Rule Engine
Targets
1. Call a service.
1.2 Return the message ID.
1.1 Verify the parameters.
1.3 Parse the data if the data type is Do not parse/Custom.
1.4 Send data to device.
2. Return results.
2.1 Parse the result data if the data type is Do not parse/Custom.
2.2 Forward data using Rule Engine if rules have been defined.
2.2.1 Forward data to the defined targets.

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.

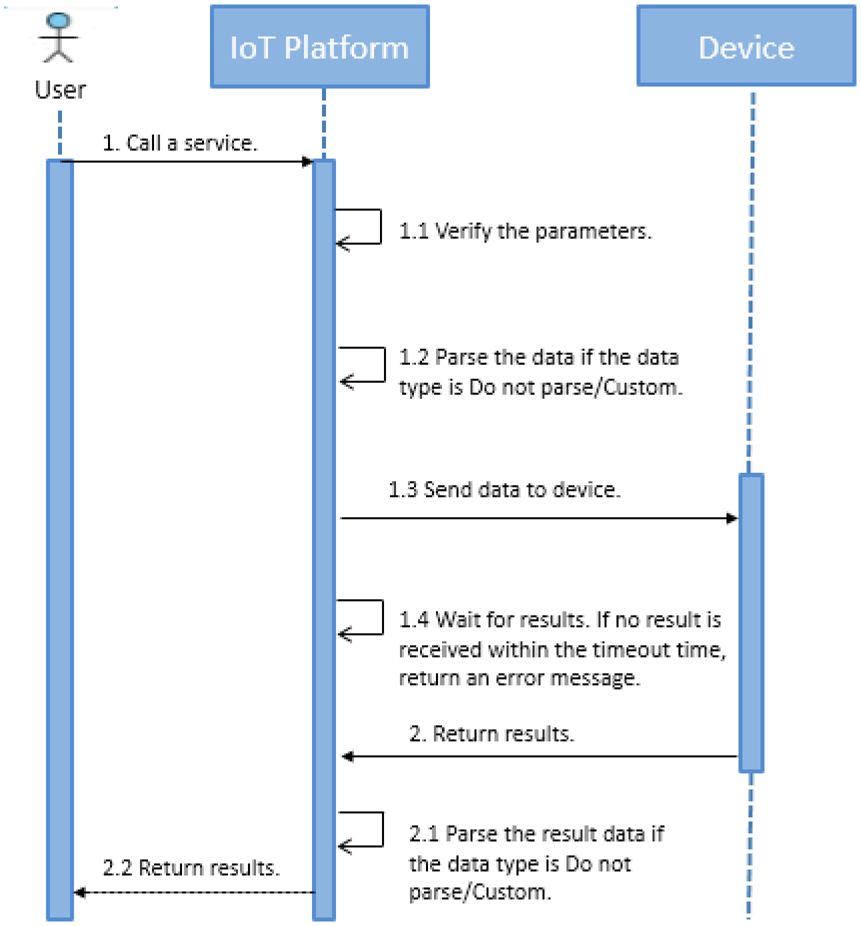
Note:

- 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.


User
IoT Platform
Device
1. Call a service.
1.1 Verify the parameters.
1.2 Parse the data if the data type is Do not parse/Custom.
1.3 Send data to device.
1.4 Wait for results. If no result is received within the timeout time, return an error message.
2. Return results.
2.1 Parse the result data if the data type is Do not parse/Custom.
2.2 Return results.

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:

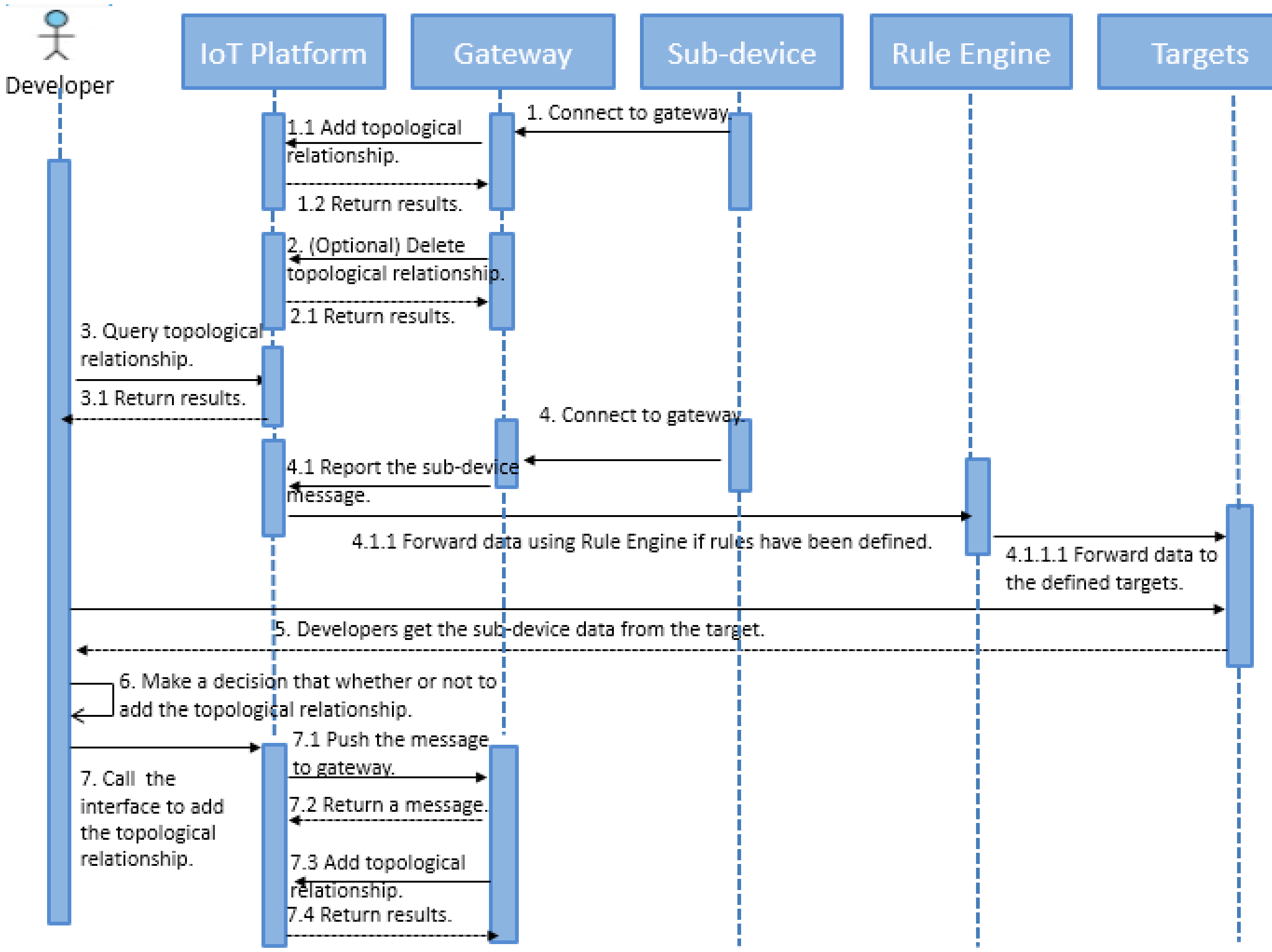
If the data type of the device is Do not parse/Custom, IoT Platform will call the `protocolTo RawData` method in the data parsing script to convert the data before sending the data to the device.

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.


Developer
IoT Platform
Gateway
Sub-device
Rule Engine
Targets
1. Connect to gateway.
1.1 Add topological relationship.
1.2 Return results.
2. (Optional) Delete topological relationship.
2.1 Return results.
3. Query topological relationship.
3.1 Return results.
4. Connect to gateway.
4.1 Report the sub-device message.
4.1.1 Forward data using Rule Engine if rules have been defined.
4.1.1.1 Forward data to the defined targets.
5. Developers get the sub-device data from the target.
6. Make a decision that whether or not to add the topological relationship.
7. Call the interface to add the topological relationship.
7.1 Push the message to gateway.
7.2 Return a message.
7.3 Add topological relationship.
7.4 Return results.

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.

6.2 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

- Topic: /sys/{productKey}/{deviceName}/thing/sub/register
- Reply topic: /sys/{productKey}/{deviceName}/thing/sub/register_reply

Request message

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

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. Reserve the value of the parameter for future use.
version	String	Protocol version. Currently, the value can only be 1.0.
params	List	Parameters used for dynamic registration.
deviceName	String	Name of the sub-device.
productKey	String	ID of the product to which the sub-device belongs.
iotId	String	Unique identifier of the sub-device.

Parameter	Type	Description
deviceSecret	String	DeviceSecret key.
method	String	Request method.
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 )
```

6.3 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

- 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 "
    }
  ],
  " method ": " thing . topo . add "
}
```

Response data format when using the Alink protocol

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the parameter value for future use.
version	String	Protocol version. Currently, the value can only be 1.0.
params	List	Input parameters of the request.
deviceName	String	Device name. The value is the name of the sub-device.
productKey	String	Product ID. The value is the ID of the product to which the sub-device belongs.

Parameter	Type	Description

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 ( deviceSecr`
		`et ,  clientId12  3deviceNam`
		`etestprodu  ctKey123ti`
		`mestamp152  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

- Topic: /sys/{productKey}/{deviceName}/thing/topo/delete
- Reply topic: /sys/{productKey}/{deviceName}/thing/topo/delete_reply

Request data format when using the Alink protocol

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

Response data format when using the Alink protocol

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the parameter value for future use.
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.
method	String	Request method.
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

- 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 ": {},
  " method ": " thing . topo . get "
}
```

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. Reserve the value of the parameter for future use.
version	String	Protocol version. Currently, the value can only be 1.0.
params	Object	Request parameters. This can be left empty.

Parameter	Type	Description
method	String	Request method.
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

- Topic: /sys/{productKey}/{deviceName}/thing/list/found
- Reply topic: /sys/{productKey}/{deviceName}/thing/list/found_reply

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

Request data format when using the Alink protocol

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

Response data format when using the Alink protocol

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the value of the parameter for future use.
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.
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

- Topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify
- Reply topic: /sys/{productKey}/{deviceName}/thing/topo/add/notify_reply

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

Request data format when using the Alink protocol

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " deviceName ": " 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. Reserve the value of the parameter for future use.
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.
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.

6.4 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.

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	The message ID.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	The unique identifier of the product to which the device belongs.
sign	String	Signature of the sub-device. Sub-devices use the same signature rules as gateways. Sign algorithm: 1. Sort all the parameters (except `sign` and `signMethod` and `cleanSession`) to be submitted to the server in alphabetical order, and then concatenate the parameters and values in turn (without any delimiters). 2. Then, sign the parameters by using the algorithm specified by signMethod and the DeviceSecret of the sub-device. Example: `sign = hmac_md5 ( deviceSecret , clientId123deviceNametestproductKey123timestamp123 )`
signMethod	String	Sign method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.

Parameter	Type	Description
clientId	String	Identifier of the device client. 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.

Error code	Error message	Description
521	device deleted	The sub-device has been deleted .
522	device forbidden	The sub-device has been disabled.
6287	invalid sign	The password or signature of the sub-device is incorrect.

Disconnect a sub-device from IoT Platform

Upstream

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

Note:

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

Request message:

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

Note:

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

Response message:

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

Request Parameters

Parameter	Type	Description
id	String	The message ID.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	The unique identifier of the product to which the device belongs.

Response parameters

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

Error messages

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

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

6.5 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. This topic describes how data is reported based on the TSL. For information about the data format of TSL, see *The TSL format* .

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 .

- 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. We recommend that you select this data type. Note that the following examples in this topic use the Alink JSON 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. For information about how to edit a data parsing script, see *Data parsing*.

Devices report properties

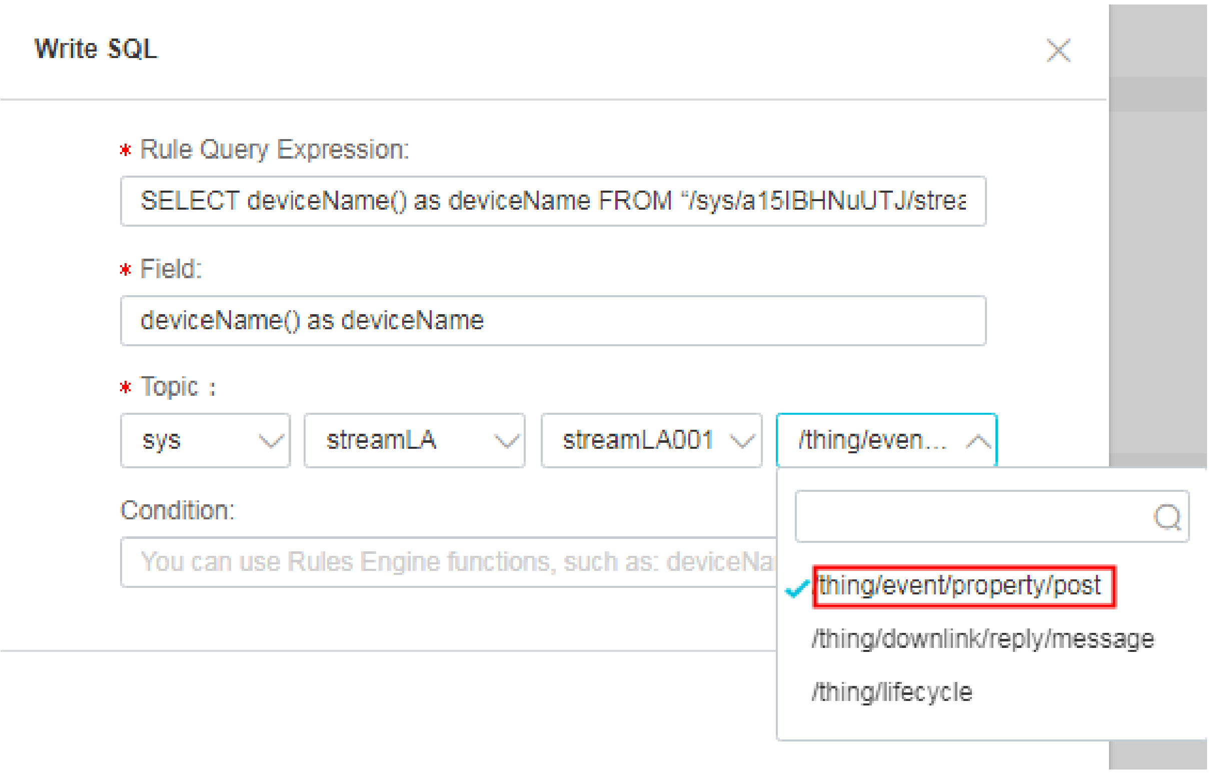
Report data (Do not parse/Custom)

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

Report Data (Alink JSON)

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

You can configure *Rules Engine* to forward the received property data to another Alibaba Cloud product instance. The following figure is an example of rule action configuration.



SELECT deviceName() as deviceName FROM "/sys/a15IBHNuUTJ/strea

deviceName() as deviceName

Request message

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

Response data

- If the data type of the product is Do not parse/Custom, then the response data is similar to the following output, and will be parsed by IoT Platform using the data parsing script before being sent to the device. For information about data processing procedures, see *Devices report properties and events*.

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

- Alink response message

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

Table 6-1: Request Parameters

Parameter	Type	Description
id	String	The message ID.
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.
method	String	The request method.

Table 6-2: 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.

Error messages

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. This error is reported when a system exception occurs. For the TSL definition, see *Overview*. Note: 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.

Set device properties

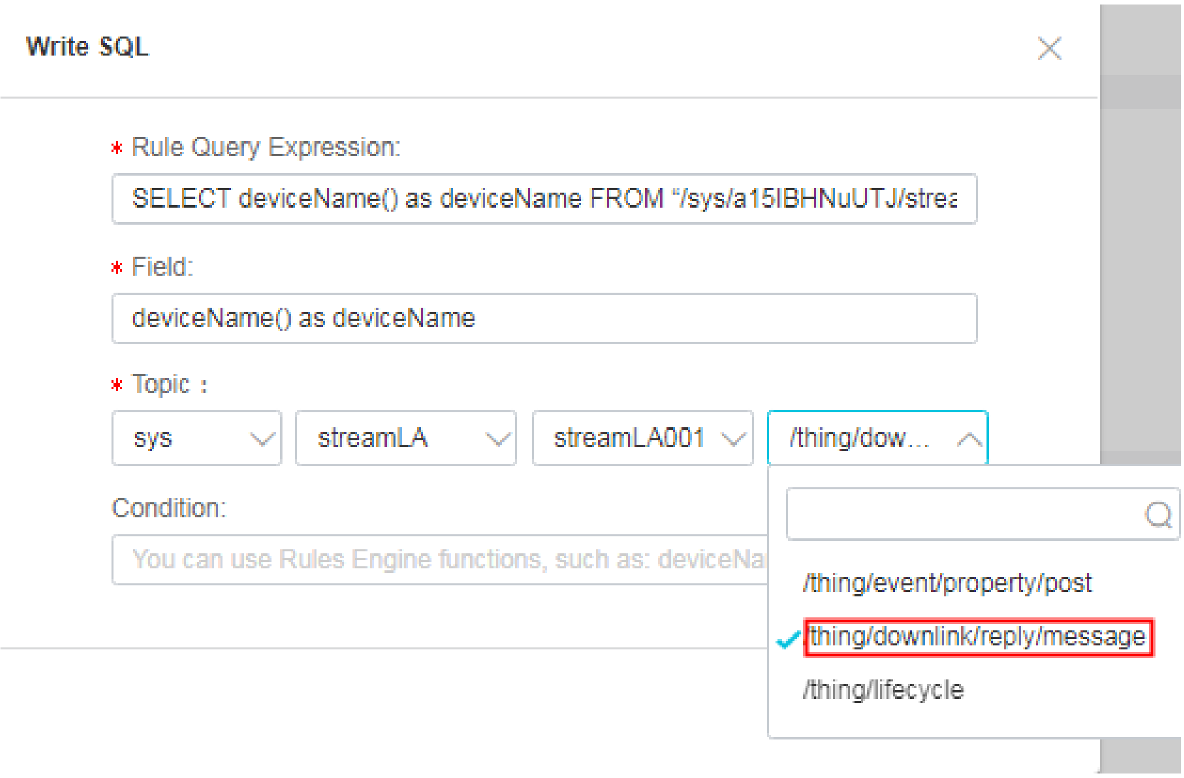
Push data to devices (Do not parse/Custom)

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

Push data to devices (Alink JSON)

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

You can get property setting results from the topic of data exchange as follows: `/ sys /{ productKey }/{ deviceName }/ thing / downlink / reply / message `. You can also configure *Rules Engine* to forward property setting results to another Alibaba Cloud product instance. The following figure is an example of rule action configuration.



Write SQL

* Rule Query Expression:

SELECT deviceName() as deviceName FROM "/sys/a15IBHNuUTJ/strea

* Field:

deviceName() as deviceName

* Topic :

sys | streamLA | streamLA001 | /thing/dow...

Condition:

You can use Rules Engine functions, such as: deviceNa

Request message

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

Response message

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

Table 6-3: Request Parameters

Parameter	Type	Description
id	String	The message ID.
version	String	The protocol version. Currently, the value is 1.0.
params	Object	The property parameters. In the preceding request example, the property to be set is `{ "temperature": "30.5" }` .
method	String	The request method.

Table 6-4: 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.

Devices report events

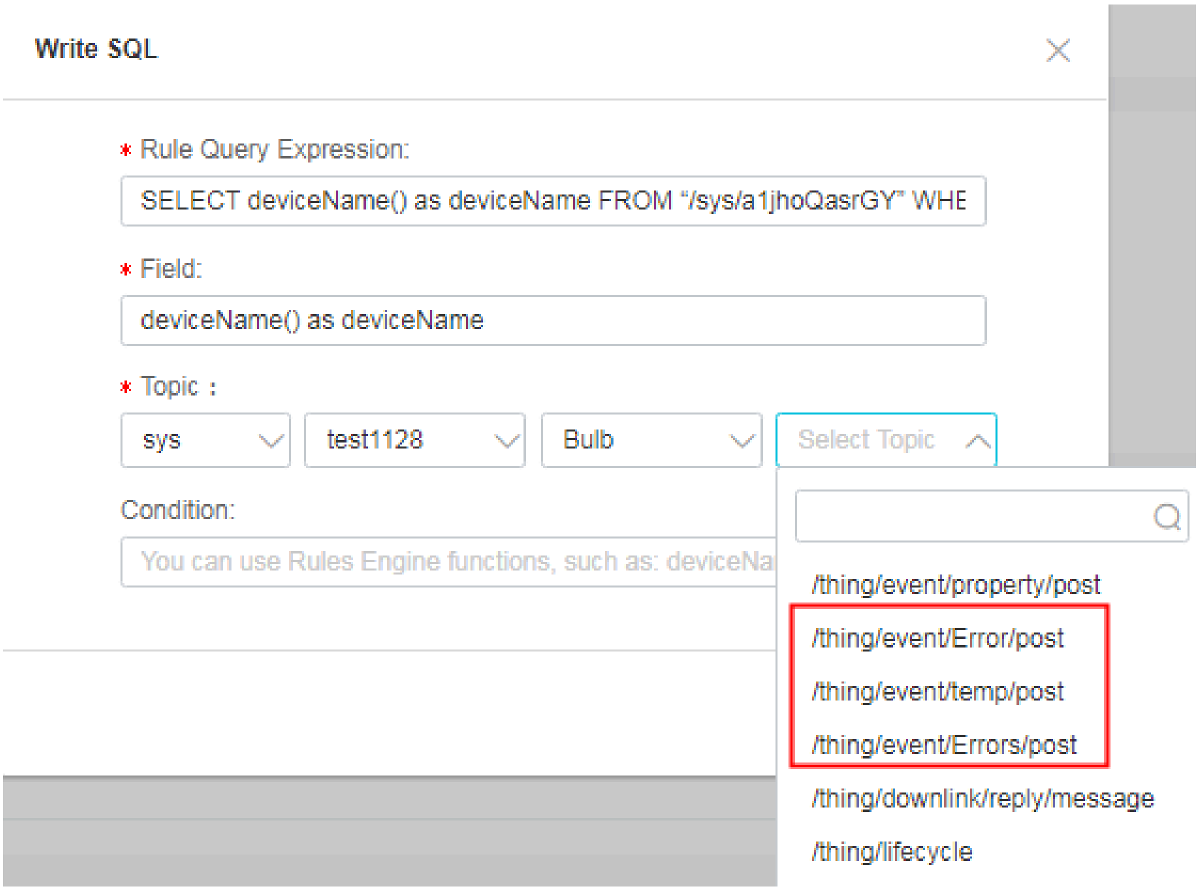
Report data (Do not parse/Custom)

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

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

You can configure *Rules Engine* to forward the received event data to another Alibaba Cloud product instance. The following figure is an example of rule action configuration.



Write SQL

* Rule Query Expression:

SELECT deviceName() as deviceName FROM "/sys/a1jhoQasrGY" WHE

* Field:

deviceName() as deviceName

* Topic :

sys | test1128 | Bulb | Select Topic

Condition:

You can use Rules Engine functions, such as: deviceNa

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " value ": {
      " Power ": " on ",
      " WF ": " 2 "
    },
    " time ": 1524448722 000
  },
  " method ": " thing . event .{ tsl . event . identifier }. post "
}
```

Response message

- If the data type of the product is Do not parse/Custom, then the response message is similar to the following output, and will be parsed by IoT Platform using the data parsing script before being sent to the device. For information about data processing procedures, see *Devices report properties and events*.

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

- Alink response message

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

Table 6-5: Request Parameters

Parameter	Type	Description
id	String	The message ID.
version	String	The protocol version. Currently, the value is 1.0.
params	List	The parameters of the reported events.
value	Object	The event information. In the preceding request example, the events are: `{ " Power ": " on ", " WF ": " 2 " }`
time	Long	The UTC timestamp when the event occurs.
method	String	The request method.

Table 6-6: Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	The result code. See *Common codes on devices*.

Parameter	Type	Description
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

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {
    " value ": {
      " errorCode ": " error "
    },
    " time ":  1524448722  000
  },
  " method ": " thing . event . alarm . post "
}
```

Note:

- `tsl . event . identifier` indicates the event identifier in the TSL. For information about TSL template, see *Overview*.
- 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.

Call device services

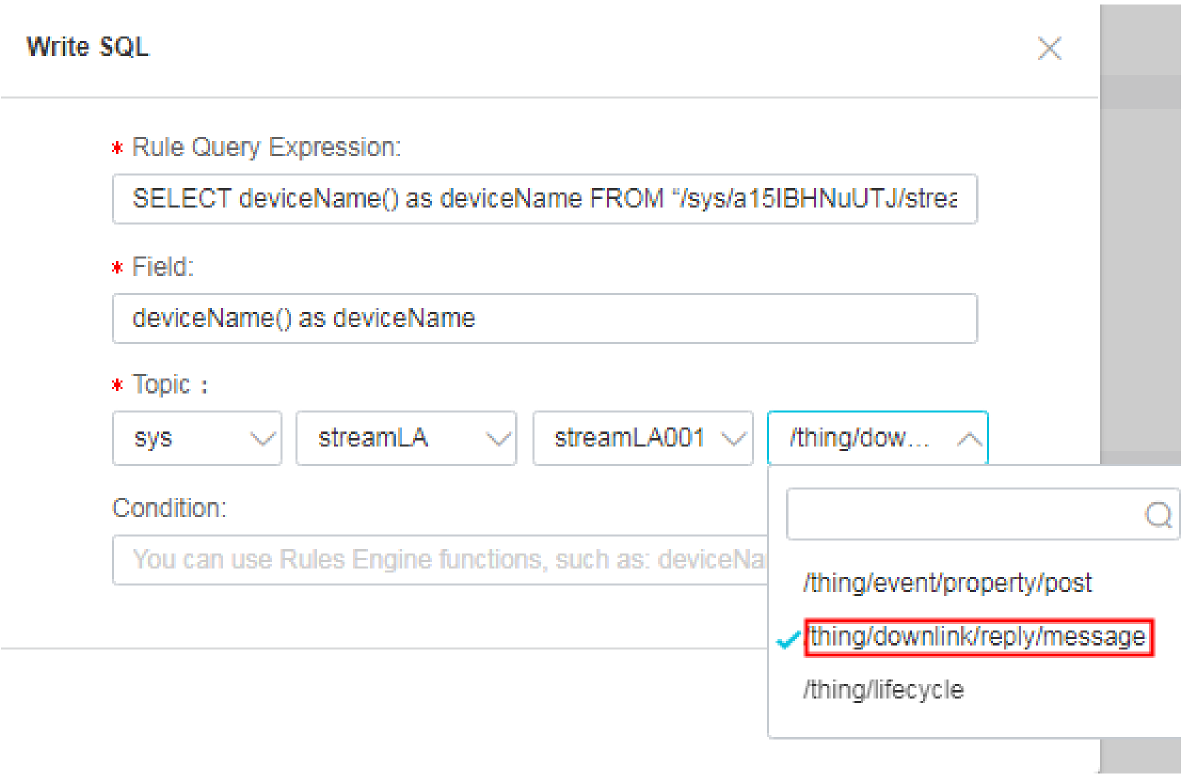
- Push data to devices (Do not parse/Custom)
  - Request topic: /sys/{productKey}/{deviceName}/thing/model/down_raw
  - Response topic: /sys/{productKey}/{deviceName}/thing/model/down_raw_reply
- Push data to devices (Alink JSON)
  - Request topic: /sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}
  - Response topic: /sys/{productKey}/{deviceName}/thing/service/{tsl.service.identifier}_reply

When you *define a service*, you must select a method for the service. The method for a service can be either synchronous method or asynchronous method .

- 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 get the operation results from the topic of data exchange as follows: `/ sys /{ productKey }/{ deviceName }/ thing / downlink / reply / message` .

You can configure *Rules Engine* to forward service call results returned by devices to another Alibaba Cloud product instance. The following figure is an example of rule action configuration.



Write SQL

* Rule Query Expression: SELECT deviceName() as deviceName FROM "/sys/a15IBHNuUTJ/strea

* Field: deviceName() as deviceName

* Topic : sys | streamLA | streamLA001 | /thing/dow...

Condition: You can use Rules Engine functions, such as: deviceNa

Request message

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

Response message

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

Table 6-7: Request Parameters

Parameter	Type	Description
id	String	The message ID.
version	String	The protocol version. Currently, the value is 1.0.
params	Map	The parameters used to call a service, including the identifier and value of the service. Example: `{ " Power ": " on ", " WF ": " 2 " }`
method	String	The request method.

Table 6-8: 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 for 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
  }
}
```

Note:

`tsl . service . identifier` indicates the identifier of the service in TSL. For information about how to define a TSL, see *Overview*.

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 at one time.
- A gateway can report up to 20 events at one time.
- A gateway can report the properties and events of up to 20 sub-devices.
- IoT Platform then verifies the devices, topological relationships, and property and event definitions in the TSL. If any one of the verifications fail, the data report also fails.

Report data (Do not parse/Custom)

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

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
          }
        }
      }
    ]
  },
  " method ": " thing . event . property . pack . post "
}
```

Response message

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

Request Parameters

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

Response parameters

Parameter	Type	Description
id	String	The message ID.
code	Integer	The result code. 200 indicates that the request is successful.

Parameter	Type	Description
data	Object	The data that is returned when the request is successful.

6.6 Send configuration data to gateway devices

Send extended configuration information of the TSL model and sub-device connection channel configuration that you configured on the cloud to the gateway device.

- Topic: /sys/{productKey}/{deviceName}/thing/model/config/push

Request message

```
{
  " id ":  123 ,
  " version ": " 1 . 0 ",
  " method ": " thing . model . config . push ",
  " data ": {
    " digest ":"",
    " digestMeth  od ":"",
    " url ": ""
  }
}
```

Parameter description

Parameter	Type	Description
id	String	The message ID.
version	String	The protocol version number. Default value: 1.0.
method	String	The method is `thing . model . config . push` .
data	Object	Data
digest	String	The signature that is used to verify the integrity of the data obtained from url.
digestMethod	String	The signature method. The default method is sha256.
url	String	The data url that you get from OSS.

Response message

```
{
  " id ": 123 ,
  " code ": 200 ,
  " message ":" success ",
  " data ":{
    " digest ":"",
    " digestMeth  od ":"",
    " url ":""
  }
}
```

url data

```
{
  " modelList ": [
    {
      " profile ": {
        " productKey ": " test01 "
      },
      " services ": [
        {
          " outputData ": "",
          " identifier ": " AngleSelfA  daption ",
          " inputData ": [
            {
              " identifier ": " test01 ",
              " index ":  0
            }
          ],
          " displayNam  e ": " test01 "
        }
      ],
      " properties ": [
        {
          " identifier ": " identifier ",
          " displayNam  e ": " test02 "
        },
        {
          " identifier ": " identifier  _01 ",
          " displayNam  e ": " identifier  _01 "
        }
      ],
      " events ": [
        {
          " outputData ": [
            {
              " identifier ": " test01 ",
              " index ":  0
            }
          ],
          " identifier ": " event1 ",
          " displayNam  e ": " abc "
        }
      ]
    },
    {
      " profile ": {
        " productKey ": " test02 "
      },
      " properties ": [
        {
          " originalDa  taType ": {
            " specs ": {
              " registerCo  unt ":  1 ,
              " reverseReg  ister ":  0 ,
              " swap16 ":  0
            },
            " type ": " bool "
          },
          " identifier ": " test01 ",
          " registerAd  dress ": " 0x03 ",
          " scaling ":  1 ,
          " operateTyp  e ": " inputStatu  s ",
          " pollingTim  e ":  1000 ,
          " trigger ":  1
        },
        {
          " originalDa  taType ": {
            " specs ": {
              " registerCo  unt ":  1 ,
              " reverseReg  ister ":  0 ,
              " swap16 ":  0
            },
            " type ": " bool "
          },
          " identifier ": " test02 ",
          " registerAd  dress ": " 0x05 ",
          " scaling ":  1 ,
          " operateTyp  e ": " coilStatus ",
          " pollingTim  e ":  1000 ,
          " trigger ":  2
        }
      ]
    }
  ],
  " serverList ": [
    {
      " baudRate ":  1200 ,
      " protocol ": " RTU ",
      " byteSize ":  8 ,
      " stopBits ":  2 ,
      " parity ":  1 ,
      " name ": " modbus01 ",
      " serialPort ": " 0 ",
      " serverId ": " D73251B427  7742 "
    },
    {
      " protocol ": " TCP ",
      " port ":  8000 ,
      " ip ": " 192 . 168 . 0 . 1 ",
      " name ": " modbus02 ",
      " serverId ": " 586CB066D6  A34 "
    },
    {
      " password ": " XIJTginONo  hPEUAyZxLB  7Q ==",
      " secPolicy ": " Basic128Rs  a15 ",
      " name ": " server_01 ",
      " secMode ": " Sign ",
      " userName ": " 123 ",
      " serverId ": " 55A9D276A7  ED470 ",
      " url ": " tcp : 00 ",
      " timeout ":  10
    },
    {
      " password ": " hAaX5s13gw  X2JwyvUkOA  fQ ==",
      " name ": " service_09 ",
      " secMode ": " None ",
      " userName ": " 1234 ",
      " serverId ": " 44895C63E3  FF401 ",
      " url ": " tcp : 00 ",
      " timeout ":  10
    }
  ],
  " deviceList ": [
    {
      " deviceConf  ig ": {
        " displayNam  ePath ": " 123 ",
        " serverId ": " 44895C63E3  FF4013924C  EF31519ABE  7B "
      },
      " productKey ": " test01 ",
      " deviceName ": " test_02 "
    },
    {
      " deviceConf  ig ": {
        " displayNam  ePath ": " 1 ",
        " serverId ": " 55A9D276A7  ED47 "
      },
      " productKey ": " test01 ",
      " deviceName ": " test_03 "
    },
    {
      " deviceConf  ig ": {
        " slaveId ":  1 ,
        " serverId ": " D73251B427  7742D "
      },
      " productKey ": " test02 ",
      " deviceName ": " test01 "
    },
    {
      " deviceConf  ig ": {
        " slaveId ":  2 ,
        " serverId ": " 586CB066D6  A34E "
      },
      " productKey ": " test02 ",
      " deviceName ": " test02 "
    }
  ],
  " tslList ": [
    {
      " schema ": " https :// iotx - tsl . oss - ap - southeast - 1 .
 aliyuncs . com / schema . json ",
      " profile ": {
        " productKey ": " test02 "
      },
      " services ": [
        {
          " outputData ": [],
          " identifier ": " set ",
          " inputData ": [
            {
              " identifier ": " test02 ",
              " dataType ": {
                " specs ": {
                  " unit ": " mm ",
                  " min ": " 0 ",
                  " max ": " 1 "
                },
                " type ": " int "
```

```
                },
                " name ": " FeatureTes  t02 "
              }
            ],
            " method ": " thing . service . property . set ",
            " name ": " set ",
            " required ":  true ,
            " callType ": " async ",
            " desc ": " Set   properties "
          },
          {
            " outputData ": [
              {
                " identifier ": " test01 ",
                " dataType ": {
                  " specs ": {
                    " unit ": " m ",
                    " min ": " 0 ",
                    " max ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " FeatureTes  t01 "
              },
              {
                " identifier ": " test02 ",
                " dataType ": {
                  " specs ": {
                    " unit ": " mm ",
                    " min ": " 0 ",
                    " max ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " FeatureTes  t02 "
              }
            ],
            " identifier ": " get ",
            " inputData ": [
              " test01 ",
              " test02 "
            ],
            " method ": " thing . service . property . get ",
            " name ": " get ",
            " required ":  true ,
            " callType ": " async ",
            " desc ": " Get   properties "
          }
        ],
        " properties ": [
          {
            " identifier ": " test01 ",
            " dataType ": {
              " specs ": {
                " unit ": " m ",
                " min ": " 0 ",
                " max ": " 1 "
              },
              " type ": " int "
            },
            " name ": " FeatureTes  t01 ",
            " accessMode ": " r ",
            " required ":  false
          },
          {
            " identifier ": " test02 ",
            " dataType ": {
              " specs ": {
                " unit ": " mm ",
                " min ": " 0 ",
                " max ": " 1 "
              },
              " type ": " int "
            },
            " name ": " FeatureTes  t02 ",
            " accessMode ": " rw ",
            " required ":  false
          }
        ],
        " events ": [
          {
            " outputData ": [
              {
                " identifier ": " test01 ",
                " dataType ": {
                  " specs ": {
                    " unit ": " m ",
                    " min ": " 0 ",
                    " max ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " FeatureTes  t01 "
              },
              {
                " identifier ": " test02 ",
                " dataType ": {
                  " specs ": {
                    " unit ": " mm ",
                    " min ": " 0 ",
                    " max ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " FeatureTes  t02 "
              }
            ],
            " identifier ": " post ",
            " method ": " thing . event . property . post ",
            " name ": " post ",
            " type ": " info ",
            " required ":  true ,
            " desc ": " Report   properties "
          }
        ]
      },
      {
        " schema ": " https :// iotx - tsl . oss - ap - southeast - 1 .
   aliyuncs . com / schema . json ",
        " profile ": {
          " productKey ": " test01 "
        },
        " services ": [
          {
            " outputData ": [],
            " identifier ": " set ",
            " inputData ": [
              {
                " identifier ": " identifier ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " 7614 "
              },
              {
                " identifier ": " identifier  _01 ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " FeatureTes  t01 "
              }
            ],
            " method ": " thing . service . property . set ",
            " name ": " set ",
            " required ":  true ,
            " callType ": " async ",
            " desc ": " Set   properties ",
          },
          {
            " outputData ": [
              {
                " identifier ": " identifier ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " 7614 "
              },
              {
                " identifier ": " identifier  _01 ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " FeatureTes  t01 "
              }
            ],
            " identifier ": " get ",
            " inputData ": [
              " identifier ",
              " identifier  _01 "
            ],
            " method ": " thing . service . property . get ",
            " name ": " get ",
            " required ":  true ,
            " callType ": " async ",
            " desc ": " Get   properties ",
          },
          {
            " outputData ": [],
            " identifier ": " AngleSelfA  daption ",
            " inputData ": [
              {
                " identifier ": " test01 ",
                " dataType ": {
                  " specs ": {
                    " min ": " 1 ",
                    " max ": " 10 ",
                    " step ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " Parameter1 ",
              }
            ],
            " method ": " thing . service . AngleSelfA  daption ",
            " name ": " adaptive   angle   calibratio  n ",
            " required ":  false ,
            " callType ": " async "
          }
        ],
        " properties ": [
          {
            " identifier ": " identifier ",
            " dataType ": {
              " specs ": {
                " length ": " 2048 "
              },
              " type ": " text "
            },
            " name ": " 7614 ",
            " accessMode ": " rw ",
            " required ":  true
          },
          {
            " identifier ": " identifier  _01 ",
            " dataType ": {
              " specs ": {
                " length ": " 2048 "
              },
              " type ": " text "
            },
            " name ": " FeatureTes  t01 ",
            " accessMode ": " rw ",
            " required ":  false
          }
        ],
        " events ": [
          {
            " outputData ": [
              {
                " identifier ": " identifier ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " 7614 "
              },
              {
                " identifier ": " identifier  _01 ",
                " dataType ": {
                  " specs ": {
                    " length ": " 2048 "
                  },
                  " type ": " text "
                },
                " name ": " FeatureTes  t01 "
              }
            ],
            " identifier ": " post ",
            " method ": " thing . event . property . post ",
            " name ": " post ",
            " type ": " info ",
            " required ":  true ,
            " desc ": " Report   properties ."
          },
          {
            " outputData ": [
              {
                " identifier ": " test01 ",
                " dataType ": {
                  " specs ": {
                    " min ": " 1 ",
                    " max ": " 20 ",
                    " step ": " 1 "
                  },
                  " type ": " int "
                },
                " name ": " ParameterT  est1 "
              }
            ],
            " identifier ": " event1 ",
            " method ": " thing . event . event1 . post ",
            " name ": " event1 ",
            " type ": " info ",
            " required ":  false
          }
        ]
      }
    ]
  }
```

Parameter description

Parameter	Type	Description
modelList	Object	The extended product information of all sub-devices that are mounted to the gateway.
serverList	Object	The sub-device channels of the gateway.
deviceList	Object	The connection configurations of all sub-devices that are mounted to the gateway.
tslList	Object	The TSL of all sub-devices that are mounted to the gateway.

modelList description

Currently, the communication protocols Modbus and OPC UA are supported, but the extended information of the two protocols are different.

- **Modbus**

```
{
  " profile ": {
    " productKey ": " test02 "
  },
  " properties ": [
    {
      " originalDa  taType ": {
        " specs ": {
          " registerCo  unt ":  1 ,
          " reverseReg  ister ":  0 ,
          " swap16 ":  0
        },
        " type ": " bool "
      },
      " identifier ": " test01 ",
      " registerAd  dress ": " 0x03 ",
      " scaling ":  1 ,
      " operateTyp  e ": " inputStatu  s ",
      " pollingTim  e ":  1000 ,
      " trigger ":  1
    },
    {
      " originalDa  taType ": {
        " specs ": {
          " registerCo  unt ":  1 ,
          " reverseReg  ister ":  0 ,
          " swap16 ":  0
        },
        " type ": " bool "
      },
      " identifier ": " test02 ",
      " registerAd  dress ": " 0x05 ",
      " scaling ":  1 ,
      " operateTyp  e ": " coilStatus ",
      " pollingTim  e ":  1000 ,
      " trigger ":  2
    }
  ]
}
```

Parameter description

Parameter	Type	Description
identifier	String	The identifier of a property, event, or service.
operateType	String	The operation type. Supported values include: - coilStatus - inputStatus - holdingRegister - inputRegister
registerAddress	String	The register address.

Parameter	Type	Description
originalDataType	Object	The original data type.
type	String	Supported values include: int16, uint16, int32, uint32, int64, uint64 , float, double, string, and customized data.
specs	Object	The description.
registerCount	Integer	The number of data in the register.
swap16	Integer	Swaps the first 8 bits and the last 8 bits of the 16-bit data in the register. 0: false ; 1: true.
reverseRegister	Integer	Swaps the bits of the original 32-bit data . 0: false; 1: true.
scaling	Integer	The zoom factor.
pollingTime	Integer	The collection interval.
trigger	Integer	The data report method. 1: report at a specific time; 2: report when changes are detected.

- OPC UA

```
{
  " profile ": {
    " productKey ": " test01 "
  },
  " services ": [
    {
      " outputData ": "",
      " identifier ": " AngleSelfA  daption ",
      " inputData ": [
        {
          " identifier ": " test01 ",
          " index ":  0
        }
      ],
      " displayNam  e ": " test01 "
    }
  ],
  " properties ": [
    {
      " identifier ": " identifier ",
      " displayNam  e ": " test02 "
    },
    {
      " identifier ": " identifier  _01 ",
      " displayNam  e ": " identifier  _01 "
    }
  ],
  " events ": [
    {
```

```
   " outputData ": [
    {
     " identifier ": " test01 ",
     " index ":  0
    }
   ],
   " identifier ": " event1 ",
   " displayNam  e ": " abc "
  }
 ]
}
```

Parameter description

Parameter	Type	Description
services	Object	The service.
properties	The object.	The property.
The events.	Object	The event.
outputData	Object	The output parameter, such as event reporting data and returned result of a service call.
identifier	String	The identifier.
inputData	Object	The input parameter.
index	Integer	The index information.
displayName	String	The name that is displayed.

serverList description

Two protocols (Modbus and OPC UA) are supported for channels.

- Modbus protocol

```
[
 {
  " baudRate ":  1200 ,
  " protocol ": " RTU ",
  " byteSize ":  8 ,
  " stopBits ":  2 ,
  " parity ":  1 ,
  " name ": " modbus01 ",
  " serialPort ": " 0 ",
  " serverId ": " D73251B427  7742 "
 },
 {
  " protocol ": " TCP ",
  " port ":  8000 ,
  " ip ": " 192 . 168 . 0 . 1 ",
  " name ": " modbus02 ",
  " serverId ": " 586CB066D6  A34 "
 }
]
```

Parameter	Type	Description
protocol	String	The protocol type. It can be TCP or RTU.
port	Integer	The port number.
ip	String	The IP address.
name	String	The channel name.
serverId	String	The channel ID.
baudRate	Integer	The baud rate.
byteSize	Integer	The number of bytes.
stopBits	Integer	The stop bit.
parity	Integer	The parity bit. Supported values include: - E: Even parity check. - O: Odd parity check. - N: No parity check.
serialPort	String	The serial port number.

- OPC UA protocol

```
{
  " password ": " XIJTginONo  hPEUAyZxLB  7Q ==",
  " secPolicy ": " Basic128Rs  a15 ",
  " name ": " server_01 ",
  " secMode ": " Sign ",
  " userName ": " 123 ",
  " serverId ": " 55A9D276A7  ED470 ",
  " url ": " tcp : 00 ",
  " timeout ":  10
}
```

Parameter description

Parameter	Type	Description
password	String	The password that has been encrypted by the AES encryption algorithm. For information about password encryption for OPC UA, see the information at the end of this table.
secPolicy	String	The encryption policy. Supported options include None, Basic128Rsa15, and Basic256.

Parameter	Type	Description
secMode	String	The encryption mode. Supported options include None, Sign, and SignAndEncrypt.
name	String	The server name.
userName	String	The user name.
serverId	String	The server ID.
url	String	The server connection address.
timeout	Integer	The timeout value.

Password encryption method for OPC UA

Use the AES encryption algorithm and 128-bit (16-byte) grouping. The default mode is CBC and the default padding is PKCS5Padding. Use deviceSecret of the device as the secret. The encrypted result is encoded in Base64.

Code example:

```
private   static   String   instance  = " AES / CBC / PKCS5Paddi
ng ";

    private   static   String   algorithm  = " AES ";

    private   static   String   charsetNam  e  = " utf - 8 ";
   /**
    *  Encryption   algorithm
    *
    * @ param   data  ( Data   to   be   encrypted )
    * @ param   deviceSecr  et  ( The   deviceSecr  et   of   the
device )
    * @ return
    */
    public   static   String   aesEncrypt ( String   data ,  String
  deviceSecr  et ) {
        try  {
            Cipher   cipher  =  Cipher . getInstanc  e ( instance
);
            byte []  raw  =  deviceSecr  et . getBytes ();
            SecretKeyS  pec   key  =  new   SecretKeyS  pec ( raw
,  algorithm );
            IvParamete  rSpec   ivParamete  r  =  new   IvParamete
rSpec ( deviceSecr  et . substring ( 0 ,  16 ). getBytes ());
            cipher . init ( Cipher . ENCRYPT_MO  DE ,  key ,
ivParamete  r );
            byte []  encrypted  =  cipher . doFinal ( data .
getBytes ( charsetNam  e ));

            return   new   BASE64Enco  der (). encode ( encrypted
);
       }  catch  ( Exception   e ) {
            e . printStack  Trace ();
       }
```

```
        return   null ;
   }

    public   static   String   aesDecrypt ( String   data ,  String
  deviceSecr  et ) {
        try  {
            byte []  raw  =  deviceSecr  et . getBytes (
charsetNam  e );
            byte []  encrypted1  =  new   BASE64Deco  der ().
decodeBuff  er ( data );
            SecretKeyS  pec   key  =  new   SecretKeyS  pec ( raw
,  algorithm );
            Cipher   cipher  =  Cipher . getInstanc  e ( instance
);
            IvParamete  rSpec   ivParamete  r  =  new   IvParamete
rSpec ( deviceSecr  et . substring ( 0 ,  16 ). getBytes ());
            cipher . init ( Cipher . DECRYPT_MO  DE ,  key ,
ivParamete  r );
            byte []  originalBy  tes  =  cipher . doFinal (
encrypted1 );
            String   originalSt  ring  =  new   String (
originalBy  tes ,  charsetNam  e );
            return   originalSt  ring ;
       }  catch  ( Exception   ex ) {
            ex . printStack  Trace ();
       }

        return   null ;
   }

    public   static   void   main ( String []  args )  throws
Exception  {
        String   text  = " test123 ";
        String   secret  = " testTNmjyW  HQzniA8wEk  TNmjyWHQte  st
";
        String   data  =  null ;
        data  =  aesEncrypt ( text ,  secret );
        System . out . println ( data );
        System . out . println ( aesDecrypt ( data ,  secret ));
   }
```

deviceList description

- **Modbus protocol**

```
{
  " deviceConf  ig ": {
    " slaveId ":  1 ,
    " serverId ": " D73251B427  7742D "
  },
  " productKey ": " test02 ",
  " deviceName ": " test01 "
}
```

Parameter description

Parameter	Type	Description
deviceConfig	Object	The device information.

Parameter	Type	Description
slaveId	Integer	The slave station ID.
serverId	String	The channel ID.
productKey	String	The product ID.
deviceName	String	The name of the device.

- OPC UA protocol

```
{
  " deviceConf  ig ": {
    " displayNam  ePath ": " 123 ",
    " serverId ": " 44895C63E3  FF4013924C  EF31519ABE  7B "
  },
  " productKey ": " test01 ",
  " deviceName ": " test_02 "
}
```

Parameter description

Parameter	Type	Description
deviceConfig	Object	The device connection configuration information.
productKey	String	The product ID.
deviceName	String	The name of the device.
displayNamePath	String	The name that is displayed.
serverId	String	The associated channel ID.

6.7 Disable and delete devices

Gateways can disable and delete their sub-devices.

Disable devices

Downstream

- 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.
version	String	Protocol version. Currently , the value is 1.0.
params	Object	Request parameters. Leave empty.
method	String	Request method.
code	Integer	Results information. For more information, see*Common codes on devices*

Enable devices

Downstream

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

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

Request message

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

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID.
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

- Topic: /sys/{productKey}/{deviceName}/thing/delete
- Reply topic: /sys/{productKey}/{deviceName}/thing/delete_reply

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

Request message

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

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID.
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.

6.8 Device tags

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

Report tags

Upstream

- Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/update
- Reply topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/update_reply

Request message

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

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the value of the parameter for future use.
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.
method	String	Request method.
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

- Topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/delete
- Reply topic: /sys/{productKey}/{deviceName}/thing/deviceinfo/delete_reply

Request message

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": [
    {
      " attrKey ": " Temperatur  e "
    }
  ],
  " method ": " thing . deviceinfo . delete "
}
```

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the value of the parameter for future use.
version	String	Protocol version. Currently , the value can only be 1.0.
params	Object	Request parameters.
method	String	Request method.
attrKey	String	Tag name. · Length: Up to 100 bytes. · Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_). · The tag name must start with an English letter or underscore (_).
attrValue	String	Tag value.

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

Error messages

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

6.9 TSL model

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

- Topic: /sys/{productKey}/{deviceName}/thing/dsltemplate/get
- Reply topic: /sys/{productKey}/{deviceName}/thing/dsltemplate/get_reply

The Allink data format of a request

```
{
  " id ": " 123 ",
  " version ": " 1 . 0 ",
  " params ": {},
  " method ": " thing . dsltemplat  e . get "
}
```

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. Reserve the parameter value for future use.
version	String	Protocol version. Currently , the value is 1.0.
params	Object	Leave this parameter empty.
method	String	Request method.
productKey	String	ProductKey. In the example, the ProductKey is 1234556554.
deviceName	String	Device name. In the example, the device name is airCondition.
data	Object	TSL model of the device. For more information, see*Overview*

Error codes

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

6.10 Firmware update

For information about the firmware update, see *Develop OTA features* and *Firmware update*.

Report the firmware version

Upstream

- 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.
version	String	Version information of the firmware.

Push firmware information

Downstream

- 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.
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

- 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.
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

- Topic: /ota/device/request/{productKey}/{deviceName}

Request message

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

Parameter description

Parameter	Type	Description
id	String	Message ID.
version	String	Version information of the firmware.

6.11 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

- 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 "
  },
  " method ": " thing . config . get "
}
```

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.
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.

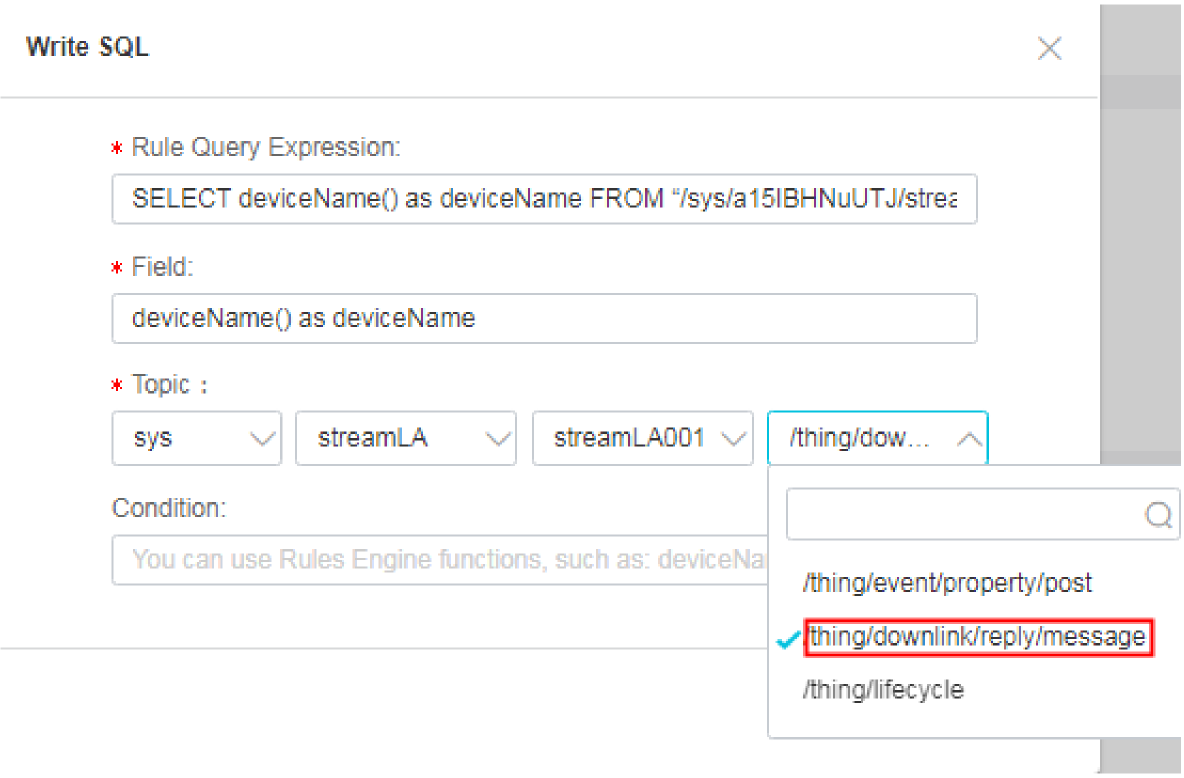
Push configurations in the IoT Platform console to devices.

Downstream

- 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.



Write SQL

* Rule Query Expression:

SELECT deviceName() as deviceName FROM "/sys/a15IBHNuUTJ/strea

* Field:

deviceName() as deviceName

* Topic :

sys | streamLA | streamLA001 | /thing/dow...

Condition:

You can use Rules Engine functions, such as: deviceNa

/thing/event/property/post

✓ /thing/downlink/reply/message

/thing/lifecycle

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.
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. For more information, see Common codes on devices.

6.12 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.