

# Alibaba Cloud IoT Platform

## Developer Guide (Devices)

Issue: 20190115

# 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 intellectu








al 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.	 <b>Danger:</b> 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.	 <b>Warning:</b> Restarting will cause business interruption. About 10 minutes are required to restore business.
	This indicates warning information, supplementary instructions, and other content that the user must understand.	 <b>Notice:</b> 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.	 <b>Note:</b> You can use <b>Ctrl + A</b> to select all files.
>	Multi-level menu cascade.	<b>Settings &gt; Network &gt; Set network type</b>
<b>Bold</b>	It is used for buttons, menus, page names, and other UI elements.	Click <b>OK</b> .
Courier font	It is used for commands.	Run the <code>cd /d C:/windows</code> command to enter the Windows system folder.
<i>Italics</i>	It is used for parameters and variables.	<code>bae log list --instanceid Instance_ID</code>
[] or [a b]	It indicates that it is a optional value, and only one item can be selected.	<code>ipconfig [-all -t]</code>
{ } or {a b}	It indicates that it is a required value, and only one item can be selected.	<code>swich {stand / slave}</code>

# Contents

---

<b>Legal disclaimer.....</b>	<b>I</b>
<b>Generic conventions.....</b>	<b>I</b>
<b>1 Download device SDKs.....</b>	<b>1</b>
<b>2 Authenticate devices .....</b>	<b>5</b>
2.1 Authenticate devices .....	6
2.2 Unique-certificate-per-device authentication.....	8
2.3 Unique-certificate-per-product authentication.....	9
<b>3 Protocols for connecting devices.....</b>	<b>12</b>
3.1 Establish MQTT connections over TCP.....	12
3.2 Establish MQTT over WebSocket connections.....	16
3.3 Establish communication over the CoAP protocol.....	18
3.4 Establish communication over the HTTPS protocol.....	25
<b>4 Configure a TSL-based device.....</b>	<b>30</b>
<b>5 OTA Development.....</b>	<b>40</b>
<b>6 Error codes for sub-device development.....</b>	<b>46</b>
<b>7 Device shadows.....</b>	<b>49</b>
7.1 Device shadow JSON format.....	49
7.2 Device shadow data stream.....	52
7.3 Use device shadows.....	60
<b>8 Java SDK.....</b>	<b>63</b>
<b>9 Develop devices based on Alink Protocol.....</b>	<b>66</b>
9.1 Alink protocol.....	66
9.2 Device identity registration.....	71
9.3 Add a topological relationship.....	75
9.4 Connect devices to IoT Platform.....	83
9.5 Device properties, events, and services.....	86
9.6 Send configuration data to gateway devices.....	98
9.7 Disable and delete devices.....	113
9.8 Device tags.....	116
9.9 TSL model.....	119
9.10 Firmware update.....	122
9.11 Remote configuration.....	126
9.12 Common codes on devices.....	130

# 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 [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 [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	√					
Connecting sub-devices to IoT Platform	√					
Device shadow	√	√	√			

	C SDK	Java SDK	Android SDK	iOS SDK	HTTP/2 SDK	General protocol
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	Development environment	Download link	Updates
V2.2.1	2018/09/03	GNU make on 64-bit Linux	<a href="#">RELEASED_V2.2.1</a>	<ul style="list-style-type: none"> <li>- Added supports for connecting devices to WiFi and using open-source applications to locally control devices.</li> <li>- Added supports for countdown routine before devices go offline.</li> <li>- Added supports for OTA using iTIs to download firmware files.</li> </ul>



Version number	Release date	Development environment	Download link	Updates
V2.1.0	2018/03/31	GNU make on 64-bit Linux	<a href="#">RELEASED_V2_10_20180331.zip</a>	<p>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.</p> <ul style="list-style-type: none"> <li>- Added support for TSL model definition on IoT Platform: You can set <code>FEATURE_CMP_ENABLED = y</code> and <code>FEATURE_DM_ENABLED = y</code> to define TSL models to provide API operations for properties, events, and services.</li> <li>- Added support for unique-certificate-per-product: You can set <code>FEATURE_SUPPORT_PRODUCT_SECRET = y</code> to enable unique-certificate-per-product authentication and streamline the production queuing process.</li> <li>- Added support for iTLS: You can set <code>FEATURE_MQTT_DIRECT_NOTLS = y</code> and <code>FEATURE_MQTT_DIRECT_NOITLS = n</code> to enable ID<sup>2</sup> encryption. You can use iTLS to establish data connections to enhance security and reduce memory consumption.</li> <li>- Added support for remote configuration: You can set <code>FEATURE_SERVICE_OTA_ENABLED = y</code> and <code>FEATURE_SERVICE_COTA_ENABLED = y</code> to enable the cloud to push configuration information to devices.</li> <li>- Optimized sub-device management of gateways : Added some features.</li> </ul>

- 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.	<a href="#">iotx-sdk-mqtt-java</a> : 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.
Other external reliance	None	None	Security of the gateway.

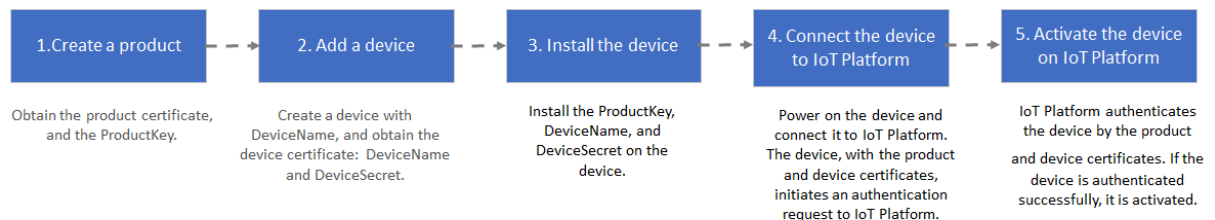
## 2.2 Unique-certificate-per-device authentication

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

### Context

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

Workflow of unique-certificate-per-device authentication:



### Procedure

1. In the [IoT Platform console](#), create a product. For more information, see [Create a product](#) in the User Guide.
2. Register a device to the product you have created and obtain the device certificate.

The screenshot shows the 'Device Details' page for a device named 'test11'. The left sidebar contains navigation links: IoT Platform, Data Overview, Quick Start, Devices (selected), Product, Device, Group, Edge Management, Rules, Applications, Data Analysis, Extended Services, and Documentation. The main content area shows the device's status as 'Inactive' and provides links to view product details, copy the ProductKey, and show the DeviceSecret. Below this is a 'Device Information' table.

Device Information					
Product Name	test11	ProductKey	atqHUCkUdL Copy	Region	China East 2 (Shanghai)
Node Type	Device	DeviceName	5b5qjzVPH2P2matgHLA Copy	DeviceSecret	***** Show
Current Status	Inactive	IP Address	-	Firmware Version	-
Created At	11/20/2018, 09:38:26	Activated At		Last Online	

Below the table is a 'Tag Information' section showing 'Device Tag: No tags' and a link to 'Add' tags.

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.

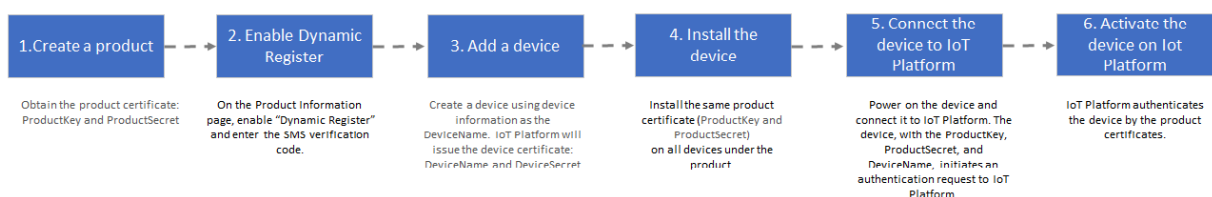
### Context



#### 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:



## 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.

The screenshot shows the 'Product Details' page for a product named 'test11'. The 'Dynamic Registration' toggle is highlighted with a red box and is set to 'Enabled'. The 'ProductKey' and 'ProductSecret' fields are also highlighted with red boxes. The 'Status' is 'Developing'.

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 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	<code>\${YourProductKey}.iot-as-mqtt. \${YourRegionId}.aliyuncs.com:1883</code> Replace <code>\${YourProductKey}</code> with your ProductKey. Replace <code>\${YourRegionId}</code> with the region ID of your device. For information about regions and zones, see <a href="#">Regions and zones</a> .
------------------------	---

Variable header: Keep Alive	<p>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.</p>
Parameters in an MQTT CONNECT packet	<pre>mqttClientId: clientId+" securemode=3,signmethod= hmacsha1,timestamp=132323232 " mqttUsername: deviceName+"&amp;"+productKey mqttPassword: sign_hmac(deviceSecret,content)</pre> <p><b>mqttPassword:</b> Sort the parameters to be submitted to the server alphabetically and then encrypt the parameters based on the specified sign method.</p> <p>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.</p> <ul style="list-style-type: none"> <li>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.</li> <li>timestamp: The 13-digit timestamp of the current time. This parameter is optional.</li> <li>mqttClientId: Extended parameters are placed between vertical bars ( ).</li> <li>signmethod: The signature algorithm. Valid values: hmacmd5, hmacsha1, and hmacsha256. Default value: hmacmd5.</li> <li>securemode: The current security mode. Value options: 2 (TLS connection) and 3 (TCP connection).</li> </ul> <p><b>Example:</b> Suppose that clientId=12345, deviceName=device, productKey=pk, timestamp=789, signmethod=hmacsha1, deviceSecret=secret. The MQTT CONNECT packet sent over TCP is as follows:</p> <pre>mqttclientId=12345 securemode=3,signmethod= hmacsha1,timestamp=789  mqttUsername=device&amp;pk mqttPassword=hmacsha1("secret","clientId12 345deviceName=deviceproductKeypktimestamp789"). toHexString(); //The toHexString() function converts a binary string to a hexadecimal string . The string is case-insensitive.</pre> <p>The encrypted password is as follows:</p> <pre>FAFD82A3D602B37FB0FA8B7892F24A477F851A14</pre>

## Connect the MQTT client to IoT Platform through HTTPS

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

- Request parameters

Parameter	Required	Description
productKey	Yes	The unique identifier of the product. You can view it in the IoT Platform console.
deviceName	Yes	The device name. You can view it in the IoT Platform console.
sign	Yes	The signature. The format is <code>hmacmd5(deviceSecret, content)</code> . 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: <code>hmacmd5</code> , <code>hmacsha1</code> , and <code>hmacsha256</code> . Default value: <code>hmacmd5</code> .
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.
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: application/x-www-form-urlencoded
Content-Length: 123
productKey=123&sign=123&timestamp=123&version=default&clientId=123
&resources=mqtt&deviceName=test
sign = hmac_md5(deviceSecret, clientId123deviceNameetestprodu
ctKey123timestamp123)
```

- Response example:

```
HTTP/1.1 200 OK
Server: Tengine
Date: Wed, 29 Mar 2017 13:08:36 GMT
Content-Type: application/json; charset=utf-8
Connection: close
{
  "code" : 200,
  "data" : {
    "iotId" : "42Ze0mk3556498a1AlTP",
    "iotToken" : "0d7fdeb9dc1f4344a2cc0d45edcb0bcb",
    "resources" : {
      "mqtt" : {
        "host" : "xxx.iot-as-mqtt.cn-shanghai.aliyuncs.com",
        "port" : 1883
      }
    },
    "message" : "success"
  }
}
```

## 2. Establish an MQTT connection.

- Download the [root.crt](#) file of IoT Platform. We recommend that you use TLS 1.2.
- Connect the device client to the Alibaba Cloud MQTT server using the returned MQTT host address and port of device authentication.
- 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.

## 3.2 Establish MQTT over WebSocket connections

### Context

IoT Platform supports MQTT over WebSocket. WebSocket is used to establish a connection. The MQTT protocol is used to communicate over the WebSocket connection.

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 Layer Security (TLS) is used in WebSocket Secure connections. Like a TLS connection, a WebSocket Secure connection requires a [root certificate](#).

#### 2. Client selection

Java clients can directly use the [Official client SDK](#) by replacing the connect URL in the SDK 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.

- Connect to the domain name of the China (Shanghai) region: `${productKey}.iot-as-mqtt.cn-shanghai.aliyuncs.com:443`

Replace `${productKey}` with your product key.

- An MQTT Connect packet contains the following parameters:

```
mqttClientId: clientId+"|securemode=3,signmethod=hmacsha1,
timestamp=132323232|"
mqttUsername: deviceName+"&"+productKey
mqttPassword: sign_hmac(deviceSecret,content)sign. Sort the
content parameters in alphabetical order and sign them according
to the signing method.
content=Parameters sent to the server (productKey,deviceName,
timestamp,clientId). Sort these parameters in alphabetical order
and splice the parameters and parameter values.
```

Where,

- `clientId`: Specifies the client ID up to 64 characters. We recommend that you use a MAC address or SN.
- `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 with predefined parameter values:

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

- For a WebSocket connection:

- Connection domain

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

- Connection parameters

```
mqttclientId=12345|securemode=3,signmethod=hmacsha1,timestamp=
789|
mqttUsername=device&pk
```

```
mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedevicep  
roductKeypktimestamp789").toHexString();
```

- For a WebSocket Secure connection:

- Connection domain

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

- Connection parameters

```
mqttclientId=12345|securemode=2,signmethod=hmacsha1,timestamp=  
789|  
mqttUsername=device&pk  
mqttPasswrod=hmacsha1("secret","clientId12345deviceNamedevicep  
roductKeypktimestamp789").toHexString();
```

### 3.3 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 <code>\${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com</code> . Replace the variable <code>\${YourProductKey}</code> 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 <a href="#">payload parameters</a> .

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	<p>The communication mode. Value options:</p> <ul style="list-style-type: none"> <li>0: After receiving a request from the device client, the server processes data and then returns the result with an acknowledgment (ACK).</li> <li>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.</li> </ul> <p>The default value is 0.</p>
sign	Yes	<p>The signature.</p> <p>The signature algorithm is <code>hmacmd5(DeviceSecret, content)</code>.</p> <p>The value of <b>content</b> is a string that is built by sorting and concatenating all the parameters (except <b>version</b>, <b>sign</b>, <b>resources</b>, and <b>signmethod</b>) that need to be submitted to the server in alphabetical order, without any delimiters.</p>
signmethod	No	The algorithm type. The supported types are <code>hmacmd5</code> and <code>hmacsha1</code> .
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": "f13102810756432e85dfd351eeb41c04"}
```

**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 <code>\${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com</code> . Replace <code>\${YourProductKey}</code> with the ProductKey value of the device.
Port	Yes	Set the value to 5684.
Accept	Yes	The encoding format of the data that is to be received by the device. Currently, application/json and application/cbor are supported.
Content-Format	Yes	The encoding format of the data that the device sends to IoT Platform. The server does not verify this parameter. Currently, application/json and application/cbor are supported.
CustomOptions	Yes	<ul style="list-style-type: none"> <li>Number: 2088.</li> <li>The value of token is the token information returned after <code>auth</code> is called to authenticate the device.</li> </ul> <div>  <b>Note:</b>  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. </div>

**Use the symmetric encryption method**

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

**Note:**

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

**2. Authenticate the device.****Request message:**

```
POST /auth
Host: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: {"productKey":"a1NUjcVkhZS","deviceName":"ff1a11e7c0
8d4b3db2b1500d8e0e55","clientId":"a1NUjcVkhZS&ff1a11e7c08d4b3db2b1
500d8e0e55","sign":"F9FD53EE0CD010FCA40D14A9FEAB81E0","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":"ad2b3a5eb51d64f7","seqOffset":1,"token":"MZ8m37hp01
w1SSqoDFzo0010500d00.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: ${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com
Port: 5682
Accept: application/json or application/cbor
Content-Format: application/json or application/cbor
payload: ${your_data}
```

```
CustomOptions: number:2088(token), 2089(seq)
```

**Table 3-6: Request parameters**

Parameter	Required	Description
Method	Yes	The request method. The supported method is POST.
URL	Yes	The format is <code>/topic/\${topic}</code> . Replace the variable <code>\${topic}</code> with the device topic used by the device to publish data.
Host	Yes	The endpoint address. The format is <code>\${YourProductKey}.coap.cn-shanghai.link.aliyuncs.com</code> . Replace the variable <code>\${YourProductKey}</code> 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	<p>The option value can be 2088 and 2089, which are described as follows:</p> <ul style="list-style-type: none"> <li>2088: Indicates the token. The value is the token returned after the device is authenticated.</li> </ul> <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;">  <b>Note:</b> 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. </div> <ul style="list-style-type: none"> <li>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.</li> </ul> <p>Response message for option</p> <div style="background-color: #f0f0f0; padding: 10px; margin: 10px 0;"> <pre>number:2090 (IoT Platform message ID)</pre> </div>

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

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

**Table 3-7: Parameter description**

Parameter	Description
Method	The request method. The supported method is POST.
URL	/auth URL address. Only HTTPS is supported.
Host	The endpoint address: <code>iot-as-http.cn-shanghai.aliyuncs.com</code>
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 <a href="#">Parameters in body</a> .

**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 information 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	<p>Signature.</p> <p>The signature algorithm is in the format of <code>hmacmd5 ( DeviceSecret , content )</code>.</p> <p>The value of <b>content</b> is a string that is built by sorting and concatenating all the parameters (except <b>version</b> , <b>sign</b>, and <b>signmethod</b>) that need to be submitted to the server in alphabetical order, without any delimiters.</p> <p>Signature example:</p> <p>If <code>clientId = 12345</code>, <code>deviceName = device</code> , <code>productKey = pk</code>, <code>timestamp = 789</code>, <code>signmethod = hmacsha1</code>, and <code>deviceSecret = secret</code>, then the signature algorithm is <code>hmacsha1 ( " secret " , "clientId12345deviceNamedeviceproductKeypktimestamp789" ) .toHexString ( ) ;</code>. In this example, binary data will be converted to hexadecimal data.</p>
signmethod	No	<p>The algorithm type. The supported types are <code>hmacmd5</code> and <code>hmacsha1</code>.</p> <p>The default value is <code>hmacmd5</code>.</p>
version	No	<p>The version. If you leave this blank, the value is default.</p>

Response example:

```
body:
{
  "code": 0, // the status code
  "message": "success", // the result message
  "info": {
    "token": "6944e5bfb92e4d4ea3918d1eda3942f6"
  }
}
```



**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 `/${YourProductKey}/${YourDeviceName}/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/a1GFjLP3xxC/device123/pub`.

Request message format:

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

**Table 3-10: Parameter description**

Parameter	Description
Method	The request method. The supported request method is POST.
URL	<code>/topic/\${topic}</code> . Replace <code>\${topic}</code> with the topic for receiving device data. Only HTTPS is supported.
Host	The endpoint address: <code>iot-as-http.cn-shanghai.aliyuncs.com</code>
password	This parameter is included in the request header. The value of this parameter is the token returned when using the <code>auth</code> interface to authenticate the device.

Parameter	Description
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": 892687627916247040,
    "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 <code>auth</code> 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 <code>auth</code> to authenticate the device again and obtain a new token.
30001	publish message error	An error occurred while publishing data.
40000	request too many	Too many requests. The throttling policy limits the number of requests.

## 4 Configure a TSL-based device

This topic describes how to configure a device based on a TSL model.

**Note:**

Only IoT Platform Pro supports this feature.

### Prerequisites

Create a product, add a device, and define the TSL in the IoT Platform console. A TSL model describes the properties, services, and events of a device, as shown in figure [Figure 4-1: Create devices](#).

**Figure 4-1: Create devices**

### Establish a connection to IoT Platform

1. For more information about establishing an MQTT connection to connect a device and IoT Platform, see [Establish MQTT connections over TCP](#).
2. Call the `linkkit_start` operation in the device SDK to establish a connection to IoT Platform and subscribe to topics.

When you use the device SDK, save a shadow for the device. A shadow is an abstraction of a device, which is used to retrieve the status information of the device. The interaction process between a device and IoT Platform is a synchronization process between the device and shadow and between the shadow and IoT Platform.

Variable `get_tsl_from_cloud` is used to synchronize the TSL model from IoT Platform when the device comes online.

- `get_tsl_from_cloud = 0`: Indicates that a TSL model has been pre-defined. `TSL_STRING` is used as the standard TSL model.

The SDK copies the TSL model that is created in the console, uses the TSL model to define `TSL_STRING` in `linkkit_sample.c`, and then calls the `linkkit_set_tsl` operation to set the pre-defined TSL model.

**Note:**

Use the C escape character correctly.

- `get_tsl_from_cloud = 1`: Indicates that no TSL model has been pre-defined. The SDK must dynamically retrieve the TSL model from IoT Platform.

Dynamically retrieving a TSL model consumes a large amount of memory and bandwidth.

The specific consumption depends on the complexity of the TSL model. A TSL model of 10 KB consumes about 20 KB of memory and 10 KB of bandwidth.

### 3. Use the `linkkit_ops_t` parameter to register the callback.

```
linkkit_start(8, get_tsl_from_cloud, linkkit_loglevel_debug, &
alinkops, linkkit_cloud_domain_sh, sample_ctx);
if (! get_tsl_from_cloud) {
    linkkit_set_tsl(TSL_STRING, strlen(TSL_STRING));
}
```

Function implementation:

```
typedef struct _linkkit_ops {
    int (*on_connect)(void *ctx);
    int (*on_disconnect)(void *ctx);
    int (*raw_data_arrived)(void *thing_id, void *data, int len, void
*ctx);
    int (*thing_create)(void *thing_id, void *ctx);
    int (*thing_enable)(void *thing_id, void *ctx);
    int (*thing_disable)(void *thing_id, void *ctx);
#ifdef RRPC_ENABLED
    int (*thing_call_service)(void *thing_id, char *service, int
request_id, int rrpc, void *ctx);
#else
    int (*thing_call_service)(void *thing_id, char *service, int
request_id, void *ctx);
#endif /* RRPC_ENABLED */
    int (*thing_prop_changed)(void *thing_id, char *property, void *
ctx);
} linkkit_ops_t;
/**
 * @brief start linkkit routines, and install callback functions(
async type for cloud connecting).
 *
 * @param max_buffered_msg, specify max buffered message size.
 * @param ops, callback function struct to be installed.
 * @param get_tsl_from_cloud, config if device need to get tsl from
cloud(! 0) or local(0), if local selected, must invoke linkkit_se
t_tsl to tell tsl to dm after start complete.
 * @param log_level, config log level.
 * @param user_context, user context pointer.
 * @param domain_type, specify the could server domain.
 *
 * @return int, 0 when success, -1 when fail.
 */
int linkkit_start(int max_buffered_msg, int get_tsl_from_cloud
, linkkit_loglevel_t log_level, linkkit_ops_t *ops, linkkit_cl
oud_domain_type_t domain_type, void *user_context);
/**
 * @brief install user tsl.
 */
```

```

* @param tsl, tsl string that contains json description for thing
object.
* @param tsl_len, tsl string length.
*
* @return pointer to thing object, NULL when fails.
*/
extern void* linkkit_set_tsl(const char* tsl, int tsl_len);

```

4. After you have connected the device to IoT Platform, log on to the IoT Platform console and verify whether the device has come online.

**Figure 4-2: Device comes online**

### Send property changes to IoT Platform

1. When the properties of a device change, the device automatically sends the changes to IoT Platform by publishing to topic `/sys/{productKey}/{deviceName}/thing/event/property/post`.

Request:

```

TOPIC: /sys/{productKey}/{deviceName}/thing/event/property/post
REPLY TOPIC: /sys/{productKey}/{deviceName}/thing/event/property/post_reply
request
{
  "id" : "123",
  "version": "1.0",
  "params" : {
    "PowerSwitch" : 1
  },
  "method": "thing.event.property.post"
}
response
{
  "id": "123",
  "code": 200,
  "data": {}
}

```

2. The SDK calls the `linkkit_set_value` operation to modify the property of the shadow, and then calls the `linkkit_trigger_event` operation to synchronize the shadow to IoT Platform.



**Note:**

The device will automatically send the current property of the shadow to IoT Platform.

Function:

```

linkkit_set_value(linkkit_method_set_property_value, sample->thing,
EVENT_PROPERTY_POST_IDENTIFIER, value, value_str); // set value

```

```
return linkkit_trigger_event(sample->thing, EVENT_PROPERTY_POST_
IDENTIFIER, NULL); // update value to cloud
```

Function implementation:

```
/**
 * @brief set value to property, event output, service output items.
 * if identifier is struct type or service output type or event
 * output type, use '.' as delimiter like "identifier1.identifier2"
 * to point to specific item.
 * value and value_str could not be NULL at the same time;
 * if value and value_str both as not NULL, value shall be used and
 * value_str will be ignored.
 * if value is NULL, value_str not NULL, value_str will be used.
 * in brief, value will be used if not NULL, value_str will be used
 * only if value is NULL.
 *
 * @param method_set, specify set value type.
 * @param thing_id, pointer to thing object, specify which thing to
 * set.
 * @param identifier, property, event output, service output
 * identifier.
 * @param value, value to set.(input int* if target value is int type
 * or enum or bool, float* if float type,
 * long long* if date type, char* if text type).
 * @param value_str, value to set in string format if value is null.
 *
 * @return 0 when success, -1 when fail.
 */
extern int linkkit_set_value(linkkit_method_set_t method_set, const
void* thing_id, const char* identifier,
const void* value, const char* value_str);
/**
 * @brief trigger a event to post to cloud.
 *
 * @param thing_id, pointer to thing object.
 * @param event_identifier, event identifier to trigger.
 * @param property_identifier, used when trigger event with method "
 * event.property.post", if set, post specified property, if NULL, post
 * all.
 *
 * @return 0 when success, -1 when fail.
 */
extern int linkkit_trigger_event(const void* thing_id, const char*
event_identifier, const char* property_identifier);
```

## Get a device property on IoT Platform

1. You can log on to the IoT Platform console and use topic `/sys/{productKey}/{deviceName}/thing/service/property/get` to get a property of a device.

Request:

```
TOPIC: /sys/{productKey}/{deviceName}/thing/service/property/get
REPLY TOPIC: /sys/{productKey}/{deviceName}/thing/service/property/
get_reply
request
{
```

```

    "id" : "123",
    "version": "1.0",
    "params" : [
        "powerSwitch"
    ],
    "method": "thing.service.property.get"
  }
  response
  {
    "id": "123",
    "code": 200,
    "data": {
      "powerSwitch": 0
    }
  }
}

```

2. When the device receives the GET command from IoT Platform, the SDK executes the command to read the property value from the shadow and returns the value to IoT Platform.

### Set a device property on IoT Platform

1. You can log on to the IoT Platform console and use topic `/sys/{productKey}/{deviceName}/thing/service/property/set` to set a property of a device client.

Request:

```

TOPIC: /sys/{productKey}/{deviceName}/thing/service/property/set
REPLY TOPIC: /sys/{productKey}/{deviceName}/thing/service/property/set_reply
payload:
{
  "id" : "123",
  "version": "1.0",
  "params" : {
    "PowerSwitch" : 0,
  },
  "method": "thing.service.property.set"
}
response
{
  "id": "123",
  "code": 200,
  "data": {}
}

```

2. The SDK registers the `thing_prop_changed` callback function in the `linkkit_ops_t` parameter of the `linkkit_start` method to respond to the request sent from IoT Platform for setting device properties.
3. The `linkkit_get_value` parameter in the callback function is used to get the device property of the shadow, which is the same as the device property that is modified on IoT Platform.
4. After setting the new property value, you can implement the `linkkit_answer_service` function to return the result to IoT Platform. You can choose whether to perform this task based on your business needs.



**Function implementation:**

```
static int thing_prop_changed(void* thing_id, char* property, void*
ctx)
{
char* value_str = NULL;
...
linkkit_get_value(linkkit_method_get_property_value, thing_id,
property, NULL, &value_str);
LINKKIT_PRINTF("#### property(%s) new value set: %s ####\n",
property, value_str);
}
/* do user's process logical here. */
linkkit_trigger_event(thing_id, EVENT_PROPERTY_POST_IDENTIFIER,
property);
return 0;
}
```

**Callback function:**

```
int (*thing_prop_changed)(void *thing_id, char *property, void *ctx);
```

**Function implementation:**

```
/**
 * @brief get value from property, event output, service input/output
 items.
 * if identifier is struct type or service input/output type or event
 output type, use '.' as delimiter like "identifier1.identifier2"
 * to point to specific item.
 * value and value_str could not be NULL at the same time;
 * if value and value_str both as not NULL, value shall be used and
 value_str will be ignored.
 * if value is NULL, value_str not NULL, value_str will be used.
 * in brief, value will be used if not NULL, value_str will be used
 only if value is NULL.
 * @param method_get, specify get value type.
 * @param thing_id, pointer to thing object, specify which thing to get
 .
 * @param identifier, property, event output, service input/output
 identifier.
 * @param value, value to get(input int* if target value is int type or
 enum or bool, float* if float type,
 * long long* if date type, char* if text type).
 * @param value_str, value to get in string format. DO NOT modify this
 when function returns,
 * user should copy to user's own buffer for further process.
 * user should NOT free the memory.
 *
 * @return 0 when success, -1 when fail.
 */
extern int linkkit_get_value(linkkit_method_get_t method_get, const
void* thing_id, const char* identifier,
```

```
void* value, char** value_str);
```

#### Function:

```
linkkit_set_value(linkkit_method_set_service_output_value, thing,
identifier, &sample->service_custom_output_contrastratio, NULL);
linkkit_answer_service(thing, service_identifier, request_id, 200);
```

#### Function implementation:

```
/**
 * @brief set value to property, event output, service output items.
 * if identifier is struct type or service output type or event output
 * type, use '.' as delimiter like "identifier1.identifier2"
 * to point to specific item.
 * value and value_str could not be NULL at the same time;
 * if value and value_str both as not NULL, value shall be used and
 * value_str will be ignored.
 * if value is NULL, value_str not NULL, value_str will be used.
 * in brief, value will be used if not NULL, value_str will be used
 * only if value is NULL.
 *
 * @param method_set, specify set value type.
 * @param thing_id, pointer to thing object, specify which thing to set
 *
 * @param identifier, property, event output, service output identifier
 *
 * @param value, value to set.(input int* if target value is int type
 * or enum or bool, float* if float type,
 * long long* if date type, char* if text type).
 * @param value_str, value to set in string format if value is null.
 *
 * @return 0 when success, -1 when fail.
 */
extern int linkkit_set_value(linkkit_method_set_t method_set, const
void* thing_id, const char* identifier,
const void* value, const char* value_str);
/**
 * @brief answer to a service when a service requested by cloud.
 *
 * @param thing_id, pointer to thing object.
 * @param service_identifier, service identifier to answer, user should
 * get this identifier from handle_dm_callback_fp_t type callback
 * report that "dm_callback_type_service_requested" happened, use this
 * function to generate response to the service sender.
 * @param response_id, id value in response payload. its value is from
 * "dm_callback_type_service_requested" type callback function.
 * use the same id as the request to send response as the same
 * communication session.
 * @param code, code value in response payload. for example, 200 when
 * service is successfully executed, 400 when not successfully executed.
 * @param rrpc, specify rrpc service call or not.
 *
 * @return 0 when success, -1 when fail.
 */
```

```
extern int linkkit_answer_service(const void* thing_id, const char*
service_identifer, int response_id, int code);
```

### IoT Platform requests a service from the device.

1. IoT Platform uses topic `/sys/{productKey}/{deviceName}/thing/service/{dsl.service.identifer}` to invoke a service from the device. The service is defined in `dsl.service.identifer` of the standard TSL model.

```
TOPIC: /sys/{productKey}/{deviceName}/thing/service/{dsl.service.
identifer}
REPLY TOPIC:
/sys/{productKey}/{deviceName}/thing/service/{dsl.service.identifer}
_reply
request
{
  "id" : "123",
  "version": "1.0",
  "params" : {
    "SprinkleTime" : 50,
    "SprinkleVolume" : 600
  },
  "method": "thing.service.AutoSprinkle"
}
response
{
  "id": "123",
  "code": 200,
  "data": {}
}
```

2. The SDK registers the `thing_call_service` callback function in the `linkkit_ops_t` parameter of the `linkkit_start` method, to send a response to the service request.
3. After setting the new property value, you must call the `linkkit_answer_service` function to send a response to IoT Platform.

Function:

```
int (*thing_call_service)(void *thing_id, char *service, int
request_id, void *ctx);
```

Function implementation:

```
static int handle_service_custom(sample_context_t* sample, void*
thing, char* service_identifer, int request_id)
{
  char identifier[128] = {0};
  /*
   * get iutput value.
   */
  snprintf(identifier, sizeof(identifier), "%s.%s", service_identifer
, "SprinkleTime");
  linkkit_get_value(linkkit_method_get_service_input_value, thing,
identifier, &sample->service_custom_input_transparency, NULL);
```

```

/*
 * set output value according to user's process result.
 */
snprintf(identifier, sizeof(identifier), "%s.%s", service_identifier
, "SprinkleVolume");
sample->service_custom_output_contrastratio = sample->service_cu
stom_input_transparency >= 0 ? sample->service_custom_input
_transparency : sample->service_custom_input_transparency * -1;
linkkit_set_value(linkkit_method_set_service_output_value, thing,
identifier, &sample->service_custom_output_contrastratio, NULL);
linkkit_answer_service(thing, service_identifier, request_id, 200);
return 0;
}

```

## Send events to IoT Platform

1. A device subscribes to topic `/sys/{productKey}/{deviceName}/thing/event/{dsl.event.identifer}/post` to send an event to IoT Platform. The event is defined in `dsl.event.identifer` of the standard TSL model.

Request:

```

TOPIC: /sys/{productKey}/{deviceName}/thing/event/{dsl.event.
identifer}/post
REPLY TOPIC: /sys/{productKey}/{deviceName}/thing/event/{dsl.event.
identifer}/post_reply
request
{
  "id" : "123",
  "version": "1.0",
  "params" : {
    "ErrorCode" : 0
  },
  "method": "thing.event.Error.post"
}
response:
{
  "id" : "123",
  "code": 200,
  "data" : {}
}

```

2. The SDK calls the `linkkit_trigger_event` method to send an event to IoT Platform.

Function:

```

static int post_event_error(sample_context_t* sample)
{
  char event_output_identifier[64];
  snprintf(event_output_identifier, sizeof(event_output_identifier),
    "%s.%s", EVENT_ERROR_IDENTIFIER, EVENT_ERROR_OUTPUT_INFO_IDENTIFIER
  );
  int errorCode = 0;
  linkkit_set_value(linkkit_method_set_event_output_value,
sample->thing,
event_output_identifier,
&errorCode, NULL);
}

```

```
return linkkit_trigger_event(sample->thing, EVENT_ERROR_IDENTIFIER,
NULL);
}
```

Function implementation:

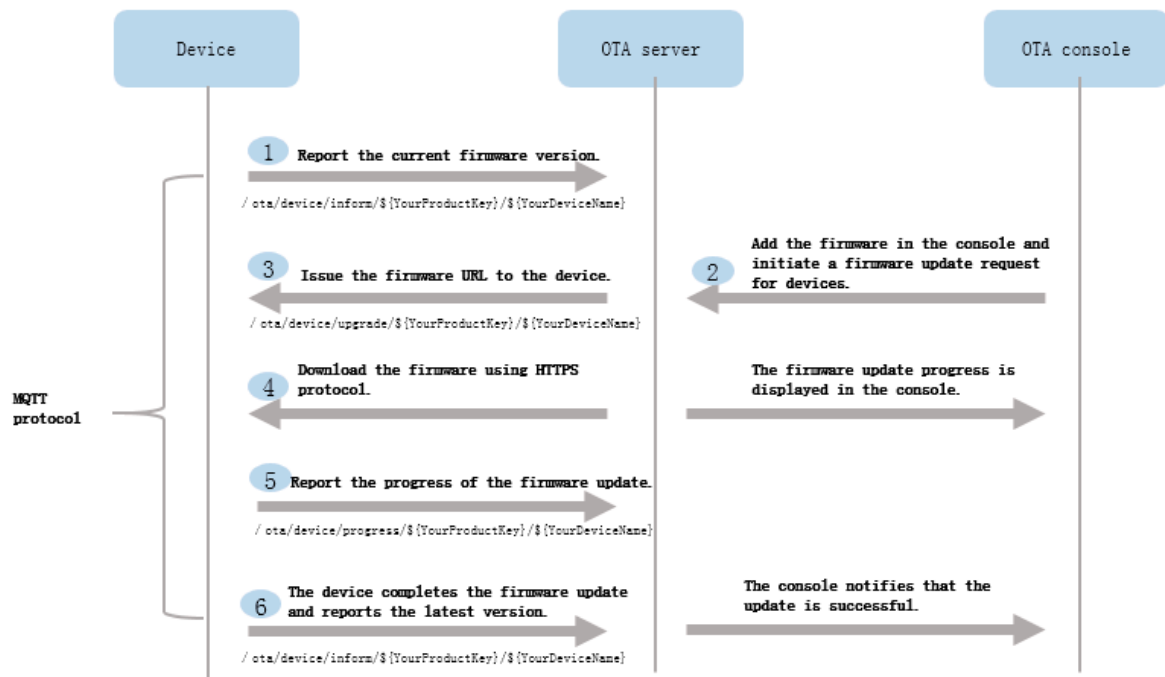
```
/**
 * @brief trigger a event to post to cloud.
 *
 * @param thing_id, pointer to thing object.
 * @param event_identifier, event identifier to trigger.
 * @param property_identifier, used when trigger event with method "
event.property.post", if set, post specified property, if NULL, post
all.
 *
 * @return 0 when success, -1 when fail.
 */
extern int linkkit_trigger_event(const void* thing_id, const char*
event_identifier, const char* property_identifier);
```

## 5 OTA Development

### Update firmware

In this example, IoT Platform uses the MQTT protocol to update the firmware. [Figure 5-1: Firmware update](#) shows the update process as follows:

Figure 5-1: Firmware update



### Topics for firmware update

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

```
/ota/device/inform/${productKey}/${deviceName}
```

- The device subscribes to this topic to receive a notification of the firmware update from IoT Platform.

```
/ota/device/upgrade/${productKey}/${deviceName}
```

- The device publishes a message to this topic to report the progress of the firmware update to IoT Platform.

```
/ota/device/progress/${productKey}/${deviceName}
```



#### Note:

- The device does not periodically send the firmware version to IoT Platform. Instead, the device sends the firmware version to IoT Platform only when the device starts.
- You can view the firmware version to check if the OTA update is successful.
- After you have configured the firmware update for multiple devices in the console of an OTA server, the update status of each device becomes Pending.

When the OTA system receives the update progress from the device, the update status of the device changes to Updating.

- An offline device cannot receive any update notifications from the OTA server.

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

## OTA code description

1. Install the firmware on a device, and start the device.

The initialization code for OTA is as follows:

```
h_ota = IOT_OTA_Init(PRODUCT_KEY, DEVICE_NAME, pclient);
if (NULL == h_ota) {
    rc = -1;
    printf("initialize OTA failed\n");
}
```



### Note:

The MQTT connection (the obtained MQTT client handle pclient) is used to initialize the OTA module.

The function is declared as follows:

```
/**
 * @brief Initialize OTA module, and return handle.
 * You must construct the MQTT client before you call this interface
 *
 * @param [in] product_key: specify the product key.
 * @param [in] device_name: specify the device name.
 * @param [in] ch_signal: specify the signal channel.
 *
 * @retval 0 : Successful.
 * @retval -1 : Failed.
 * @see None.
 */
void *IOT_OTA_Init(const char *product_key, const char *device_name
, void *ch_signal);
/**
```

```

* @brief Report firmware version information to OTA server (optional
*).
* NOTE: please
*
* @param [in] handle: specify the OTA module.
* @param [in] version: specify the firmware version in string format
.
*
* @retval 0 : Successful.
* @retval < 0 : Failed, the value is error code.
* @see None.
*/
int IOT_OTA_ReportVersion(void *handle, const char *version);

```

## 2. The device downloads the firmware from the received URL.

- IOT\_OTA\_IsFetching(): Identifies whether firmware is available for download.
- IOT\_OTA\_FetchYield(): Downloads a firmware package.
- IOT\_OTA\_IsFetchFinish(): Identifies whether the download has completed or not.

An example code is as follows:

```

// Identifies whether firmware is available for download.
if (IOT_OTA_IsFetching(h_ota)) {
    unsigned char buf_ota[OTA_BUF_LEN];
    uint32_t len, size_downloaded, size_file;
    do {
        //Iteratively downloads firmware.
        len = IOT_OTA_FetchYield(h_ota, buf_ota, OTA_BUF_LEN, 1);
        if (len > 0) {
            //Writes the firmware into the storage such as the flash.
        }
    } while (! IOT_OTA_IsFetchFinish(h_ota)); //Identifies whether the
    firmware download has completed or not.
}
exit: Ctrl ↵
/**
* @brief Check whether is on fetching state
*
* @param [in] handle: specify the OTA module.
*
* @retval 1 : Yes.
* @retval 0 : No.
* @see None.
*/
int IOT_OTA_IsFetching(void *handle);
/**
* @ Brief fetch firmware from remote server with specific timeout
value.
* NOTE: If you want to download more faster, the bigger 'buf' should
be given.
*
* @param [in] handle: specify the OTA module.
* @param [out] buf: specify the space for storing firmware data.
* @param [in] buf_len: specify the length of 'buf' in bytes.
* @param [in] timeout_s: specify the timeout value in second.
*
* @retval < 0 : Error occur..
* @retval 0 : No any data be downloaded in 'timeout_s' timeout
period.

```



```

* @retval (0, len] : The length of data be downloaded in 'timeout_s'
* timeout period in bytes.
* @see None.
*/
int IOT_OTA_FetchYield(void *handle, char *buf, uint32_t buf_len,
uint32_t timeout_s);
/**
* @brief Check whether is on end-of-fetch state.
*
* @param [in] handle: specify the OTA module.
*
* @retval 1 : Yes.
* @retval 0 : False.
* @see None.
*/
int IOT_OTA_IsFetchFinish(void *handle);

```

**Note:**

If you have insufficient device memory, you need to write the firmware into the system OTA partition while downloading the firmware.

3. Call `IOT_OTA_ReportProgress()` to report the download status.

Example code:

```

if (percent - last_percent > 0) {
    IOT_OTA_ReportProgress(h_ota, percent, NULL);
}
IOT_MQTT_Yield(pclient, 100); //

```

You can upload the update progress to IoT Platform. The update progress (1% to 100%) is displayed in real time in the progress column of the updating list in the console.

You can also upload the following error codes:

- -1: Failed to update the firmware.
- -2: Failed to download the firmware.
- -3: Failed to verify the firmware.
- -4: Failed to write the firmware into flash.

4. Call `IOT_OTA_Ioctl()` to identify whether the downloaded firmware is valid. If the firmware is valid, the device will run with the new firmware at the next startup.

Example code:

```

int32_t firmware_valid;
IOT_OTA_Ioctl(h_ota, IOT_OTAG_CHECK_FIRMWARE, &firmware_valid, 4);
if (0 == firmware_valid) {
    printf("The firmware is invalid\n");
} else {
    printf("The firmware is valid\n");
}

```

```
}
```

If the firmware is valid, modify the system boot parameters to make the hardware system run with the new firmware at the next startup. The modification method varies by hardware system.

```
/**
 * @brief Get OTA information specified by 'type'.
 * By this interface, you can get information like state, size of
 * file, md5 of file, etc.
 *
 * @param [in] handle: handle of the specific OTA
 * @param [in] type: specify what information you want, see detail '
 * IOT_OTA_CmdType_t'
 * @param [out] buf: specify buffer for data exchange
 * @param [in] buf_len: specify the length of 'buf' in byte.
 * @return
 * @verbatim
 * NOTE:
 * 1) When type is IOT_OTAG_FETCHED_SIZE, 'buf' should be pointer of
 * uint32_t, and 'buf_len' should be 4.
 * 2) When type is IOT_OTAG_FILE_SIZE, 'buf' should be pointer of
 * uint32_t, and 'buf_len' should be 4.
 * 3) When type is IOT_OTAG_MD5SUM, 'buf' should be a buffer, and '
 * buf_len' should be 33.
 * 4) When type is IOT_OTAG_VERSION, 'buf' should be a buffer, and '
 * buf_len' should be OTA_VERSION_LEN_MAX.
 * 5) When type is IOT_OTAG_CHECK_FIRMWARE, 'buf' should be pointer of
 * uint32_t, and 'buf_len' should be 4.
 * 0, firmware is invalid; 1, firmware is valid.
 * @endverbatim
 *
 * @retval 0 : Successful.
 * @retval < 0 : Failed, the value is error code.
 * @see None.
 */
int IOT_OTA_Ioctl(void *handle, IOT_OTA_CmdType_t type, void *buf,
size_t buf_len);
```

##### 5. Call IOT\_OTA\_Deinit to terminate a connection and release the memory.

```
/**
 * @brief Deinitialize OTA module specified by the 'handle', and
 * release the related resource.
 * You must call this operation to release resource if reboot is not
 * invoked after downloading.
 *
 * @param [in] handle: specify the OTA module.
 *
 * @retval 0 : Successful.
 * @retval < 0 : Failed, the value is error code.
 * @see None.
 */
```

```
int IOT_OTA_Deinit(void *handle);
```

6. After the device restarts, the device runs with the new firmware and reports the new firmware version to IoT Platform. After the OTA module is initialized, call `IOT_OTA_ReportVersion()` to report the current firmware version. The code is as follows:

```
if (0 != IOT_OTA_ReportVersion(h_ota, "version2.0")) {  
    rc = -1;  
    printf("report OTA version failed\n");  
}
```

## 6 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 6-1: Error codes**

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

Code	Message	Description
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 6-2: 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 6-3: 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 6-4: Error codes**

Code	Message	Description
520	device session error	<p>Sub-device session error.</p> <ul style="list-style-type: none"><li>The sub-device session does not exist. The sub-device is not connected to IoT Platform or has gone offline.</li><li>The sub-device session exists, however, the session is not established through the current gateway.</li></ul>

## 7 Device shadows

---

### 7.1 Device shadow JSON format

#### Format of the device shadow JSON file

The format is as follows:

```
{
  "state": {
    "desired": {
      "attribute1": integer2,
      "attribute2": "string2",
      ...
      "attributeN": boolean2
    },
    "reported": {
      "attribute1": integer1,
      "attribute2": "string1",
      ...
      "attributeN": boolean1
    }
  },
  "metadata": {
    "desired": {
      "attribute1": {
        "timestamp": timestamp
      },
      "attribute2": {
        "timestamp": timestamp
      },
      ...
      "attributeN": {
        "timestamp": timestamp
      }
    },
    "reported": {
      "attribute1": {
        "timestamp": timestamp
      },
      "attribute2": {
        "timestamp": timestamp
      },
      ...
      "attributeN": {
        "timestamp": timestamp
      }
    }
  },
  "timestamp": timestamp,
  "version": version
}
```

The JSON properties are described in [Table 7-1: JSON property](#).

**Table 7-1: JSON property**

Property	Description
desired	The desired status of the device. The application writes the desired property of the device, without accessing the device.
reported	The status that the device has reported. The device writes data to the reported property to report its latest status. The application obtains the status of the device by reading this property.
metadata	The device shadow service automatically updates metadata according to the updates in the device shadow JSON file. State metadata in the device shadow JSON file contains the timestamp of each property. The timestamp is represented as epoch time to obtain exact update time.
timestamp	The latest update time of the device shadow JSON file.
version	When you request updating the version of the device shadow, the device shadow checks whether the requested version is later than the current version. If the requested version is later than the current one, the device shadow updates to the requested version. If not, the device shadow rejects the request. The version number is increased according to the version update to ensure the latest device shadow JSON file version.

Example of the device shadow JSON file:

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



```
}  
},  
"timestamp" : 1469564492,  
"version" : 1  
}
```

### Empty properties

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

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

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

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

### Array

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

- Initial status:

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

- Update:

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

- Final status:

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

## 7.2 Device shadow data stream

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

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

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

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

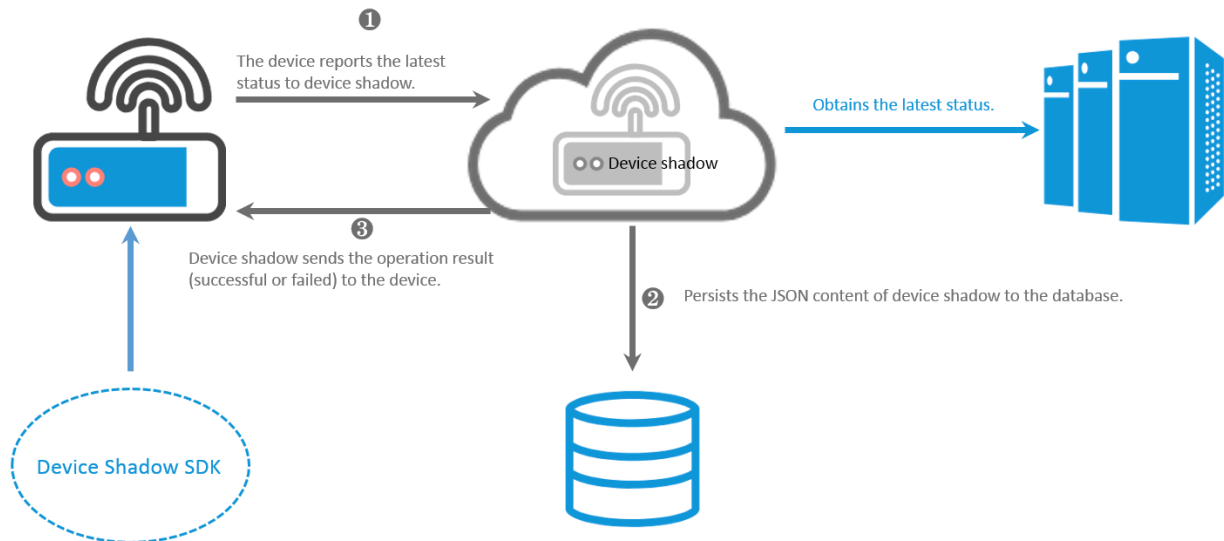
The device shadow updates the status to this topic, and the device subscribes to the messages from this topic.

Take a lightbulb device of a product `bulb_1` as an example to introduce the communication among devices, device shadows, and applications. In the following example, the ProductKey is 10000 and the DeviceName is lightbulb. The device publishes messages to and subscribes to messages of the two custom topics using the method of QoS 1.

## Device reports status automatically

The flow chart is shown in [Figure 7-1: Device reports status automatically](#).

**Figure 7-1: Device reports status automatically**



1. When the lightbulb is online, the device uses topic `/shadow/update/10000/lightbulb` to report the latest status to the device shadow.

Format of the JSON message:

```

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

The JSON parameters are described in [Table 7-2: Parameter description](#).

**Table 7-2: Parameter description**

Parameter	Description
<b>method</b>	<p>The operation type when a device or application requests the device shadow.</p> <p>When you update the status, This parameter <b>method</b> is required and must be set to <code>update</code>.</p>

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

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

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

3. After the device shadow has been updated, it will return the result to the device (lightbulb) by sending a message to the topic `/shadow/get/10000/lightbulb`.

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

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

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

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

```

},
"timestamp": 1469564576
}

```

Error codes are described in [Table 7-3: Error codes](#).

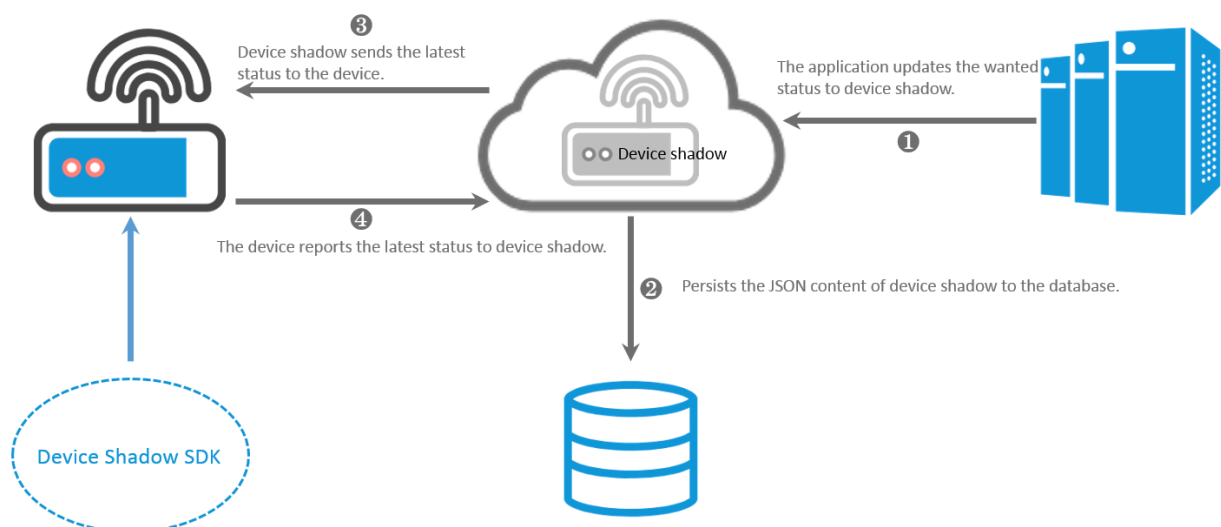
**Table 7-3: Error codes**

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

### Application changes device status

The flow chart is shown in [Figure 7-2: Application changes device status](#).

**Figure 7-2: Application changes device status**



1. The application sends a command to the device shadow to change the status of the lightbulb.

The application sends a message to topic `/shadow/update/10000/lightbulb/`. The message is as follows:

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

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

```
{
  "state" : {
    "reported" : {
      "color" : "red"
    },
    "desired" : {
      "color" : "green"
    }
  },
  "metadata" : {
    "reported" : {
      "color" : {
        "timestamp" : 1469564492
      }
    },
    "desired" : {
      "color" : {
        "timestamp" : 1469564576
      }
    }
  },
  "timestamp" : 1469564576,
  "version" : 2
}
```

3. After the update, the device shadow sends a message to the topic `/shadow/get/10000/lightbulb` and returns the result of update to the device. The result message is created by the device shadow.

```
{
  "method": "control",
  "payload": {
    "status": "success",
    "state": {
      "reported": {
        "color": "red"
      }
    }
  },
}
```

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

4. When the device lightbulb is online and has subscribed to the topic `/shadow/get/10000/lightbulb`, the device receives the message and changes its color to green according to the **desired** field in the request file. After the device has updated the status, it will report the latest status to the cloud.

```
{
  method": "update",
  "state": {
    "reported": {
      "color": "green"
    }
  },
  "version": 3
}
```

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

5. After the latest status has been reported successfully, the device client sends a message to the topic `/shadow/update/10000/lightbulb` to empty the property of desired field. The message is as follows:

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

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

```
{
  "state": {
```

```

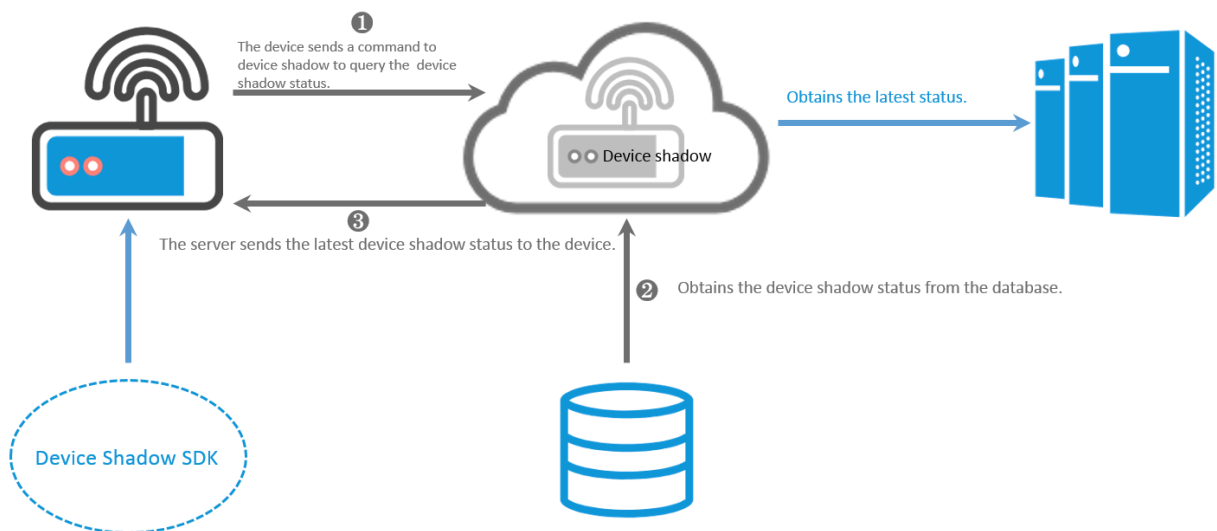
"reported" : {
  "color" : "green"
},
"metadata" : {
  "reported" : {
    "color" : {
      "timestamp" : 1469564577
    }
  },
  "desired" : {
    "timestamp" : 1469564576
  }
},
"version" : 4
}

```

### Devices request for device shadows

The flow chart is shown in [Figure 7-3: The device requests for device shadow](#).

**Figure 7-3: The device requests for device shadow**



1. The device lightbulb sends a message to the topic `/shadow/update/10000/lightbulb` and obtains the latest status saved in the device shadow. The message is as follows:

```

{
  "method": "get"
}

```

2. When the device shadow receives above message, the device shadow sends a message to the topic `/shadow/get/10000/lightbulb`. The message is as follows:

```

{
  "method": "reply",
  "payload": {
    "status": "success",
  }
}

```



```

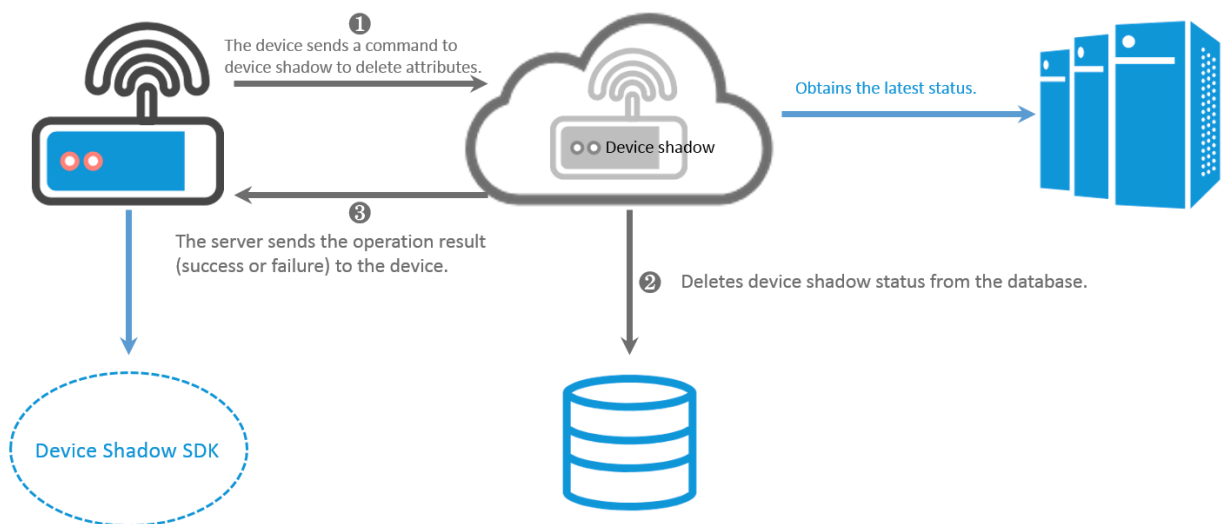
"state": {
  "reported": {
    "color": "red"
  },
  "desired": {
    "color": "green"
  }
},
"metadata": {
  "reported": {
    "color": {
      "timestamp": 1469564492
    }
  },
  "desired": {
    "color": {
      "timestamp": 1469564492
    }
  }
},
"version": 2,
"timestamp": 1469564576
}

```

### Devices delete device shadow attributes

The flow chart is shown in [Figure 7-4: Delete device shadow attributes](#).

**Figure 7-4: Delete device shadow attributes**



The device lightbulb is to delete the specified attributes saved in the device shadow. The device sends a JSON message to the topic `/shadow/update/10000/lightbulb`. See the message in the following example.

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

- Delete one attribute:

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

- Delete all the attributes:

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

## 7.3 Use device shadows

This topic describes the communication between devices, device shadows, and applications.

### Context

A device shadow is the shadow that is built on IoT Platform based on a special topic for the related device. This device synchronizes status to the cloud using this device shadow. The cloud can quickly obtain the device status from the device shadow even when the device is not connected to IoT Platform.

### Procedure

1. The C SDK provides the `IOT_Shadow_Construct` function to create the device shadow.

The function is declared as follows:

```
/**
 * @brief Construct the device shadow.
 * This function is used to initialize the data structures, establish
 * MQTT-based connections.
 * and subscribe to the topic: "/shadow/get/${product_key}/${
 * device_name}".
 *
 * @param [in] pparam: The specific initial parameter.
 * @retval NULL : The construction of the shadow failed.
 * @retval NOT_NULL : The construction is successful.
 * @see None.
 */
```

```
void *IOT_Shadow_Construct(iotx_shadow_para_pt pparam);
```

2. Use the `IOT_Shadow_RegisterAttribute` function to register the properties of the device shadow.

The function is declared as follows:

```
/**
 * @brief Create a data type registered to the server.
 *
 * @param [in] handle: The handle of the device shadow.
 * @param [in] pattr: The parameter registered to the server.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_RegisterAttribute(void *handle, iotx_shadow_attr_pt pattr);
```

3. You can use the `IOT_Shadow_Pull` function in the C SDK to synchronize device status to IoT Platform whenever the device shadow starts.

The function is declared as follows:

```
/**
 * @brief Synchronize device shadow data to the cloud.
 * It is a synchronization function.
 * @param [in] handle: The handle of the device shadow.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_Pull(void *handle);
```

4. When the device updates its status, you can use `IOT_Shadow_PushFormat_Init`, `IOT_Shadow_PushFormat_Add`, and `IOT_Shadow_PushFormat_Finalize` in the C SDK to update the device status, and use `IOT_Shadow_Push` in the C SDK to synchronize the status to the cloud.

The function is declared as follows:

```
/**
 * @brief Start processing the structure of the data type format.
 *
 * @param [in] handle: The handle of the device shadow.
 * @param [out] pformat: The format structure of the device shadow.
 * @param [in] buf: The buffer that stores the device shadow attribute.
 * @param [in] size: The maximum length of the device shadow attribute.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_PushFormat_Init(
```

```

        void *handle,
        format_data_pt pformat,
        char *buf,
        uint16_t size);

/**
 * @brief The format of the attribute name and value for the update.
 *
 * @param [in] handle: The handle of the device shadow.
 * @param [in] pformat: The format structure of the device shadow.
 * @param [in] pattr: The data type format created in the added
member attributes.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_PushFormat_Add(
        void *handle,
        format_data_pt pformat,
        iotx_shadow_attr_pt pattr);

/**
 * @ Brief Complete processing the structure of the data type format.
 *
 * @param [in] handle: The handle of the device shadow.
 * @ Param [in] pformat: The format structure of the device shadow.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_PushFormat_Finalize(void *handle, format_data_pt pformat);

```

5. To disconnect the device from IoT Platform, use `IOT_Shadow_DeleteAttribute` and `IOT_Shadow_Destroy` in the C SDK to delete all properties that have been created for this device on IoT Platform, and release the device shadow.

The function is declared as follows:

```

/**
 * @brief Deconstruct the specific device shadow.
 *
 * @param [in] handle: The handle of the device shadow.
 * @retval SUCCESS_RETURN : Success.
 * @retval other : See iotx_err_t.
 * @see None.
 */
iotx_err_t IOT_Shadow_Destroy(void *handle);

```

## 8 Java SDK

---

This topic describes how to connect devices to Alibaba Cloud IoT Platform over the MQTT protocol. The Java SDK is used as an example.

### Prerequisites

In this demo, a Maven project is used. Install Maven first.

### Context

This demo is not made for the Android operating system. If you are using Android, see open-source library <https://github.com/eclipse/paho.mqtt.android>.

### Procedure

1. Download the mqttClient SDK at [iotx-sdk-mqtt-java](#).
2. Use IntelliJ IDEA or Eclipse to import the demo into a Maven project.
3. Log on to the Alibaba Cloud IoT Platform console, and select **Devices**. Click **View** next to the device to obtain the ProductKey, DeviceName, and DeviceSecret.
4. Modify and run the SimpleClient4IOT.java configuration file.
  - a) Configure the parameters.

```
/** Obtain ProductKey, DeviceName, and DeviceSecret from the
console */
private static String productKey = "";
private static String deviceName = "";
private static String deviceSecret = "";
/** The topics used for testing */
private static String subTopic = "/" + productKey + "/" + deviceName + "/"
get";
private static String pubTopic = "/" + productKey + "/" + deviceName + "/"
pub";
```

- b) Connect to MQTT server.

```
// The client device ID. It can be specified using either MAC
address or device serial number. It cannot be empty and must
contain no more than 32 characters
String clientId = InetAddress.getLocalHost().getHostAddress();
// Authenticate the device
Map params = new HashMap();
params.put("productKey", productKey); // Specifies the product key
that the user registered in the console
params.put("deviceName", deviceName); // Specifies the device name
that the user registered in the console
params.put("clientId", clientId);
String t = System.currentTimeMillis() + "";
params.put("timestamp", t);
```

```
// Specifies the MQTT server. If using the TLS protocol, begin the
// URL with SSL. If using the TCP protocol, begin the URL with TCP
String targetServer = "ssl://" + productKey + ".iot-as-mqtt.cn-
shanghai.aliyuncs.com:1883";
// Client ID format:
String mqttClientId = clientId + "|securemode=2,signmethod=
hmacsha1,timestamp="+t+"|"; // Specifies the custom device
// identifier. Valid characters include letters and numbers. For more
// information, see Establish MQTT over TCP connections (https://
help.aliyun.com/document\_detail/30539.html?spm=a2c4g.11186623.6.
592.R3LqNT)
String mqttUsername = deviceName+"&" + productKey; // Specifies
// username format
String mqttPassword = SignUtil.sign(params, deviceSecret, "
hmacsha1");// Signature
// Code excerpt for connecting over MQTT
MqttClient sampleClient = new MqttClient(url, mqttClientId,
persistence);
MqttConnectOptions connOpts = new MqttConnectOptions();
connOpts.setMqttVersion(4); // MQTT 3.1.1
connOpts.setSocketFactory(socketFactory);
// Configure automatic reconnection
connOpts.setAutomaticReconnect(true);
// If set to true, then all offline messages are cleared. These
// messages include all QoS 1 or QoS 2 messages that are not received
connOpts.setCleanSession(false);
connOpts.setUserName(mqttUsername);
connOpts.setPassword(mqttPassword.toCharArray());
connOpts.setKeepAliveInterval(80); // Specifies the heartbeat
// interval. We recommend that you set it to 60 seconds or longer
sampleClient.connect(connOpts);
```

#### c) Send data.

```
String content = "The content of the data to be sent. It can be in
any format";
MqttMessage message = new MqttMessage(content.getBytes("utf-8"));
message.setQos(0); // Message QoS. 0: At most once. 1: At least
// once
sampleClient.publish(topic, message); // Send data to a specified
// topic
```

#### d) Receive data.

```
// Subscribe to a specified topic. When new data is sent to the
// topic, the specified callback method is called.
sampleClient.subscribe(topic, new IMqttMessageListener() {
@Override
public void messageArrived(String topic, MqttMessage message)
throws Exception {
// After the device successfully subscribes to a topic, when new
// data is sent to the topic, the specified callback method is called
.
// If you subscribe to the same topic again, only the initial
// subscription takes effect.
}
```

```
} );
```

**Note:**

For more information about MQTT connection parameters, see [Establish MQTT connections over TCP](#).

## 9 Develop devices based on Alink Protocol

---

### 9.1 Alink protocol

This article describes how to encapsulate Alink protocol data and establish connections from devices to IoT Platform using the Alink protocol.

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 introduce the device connection procedures and data pass through 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 report data.

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

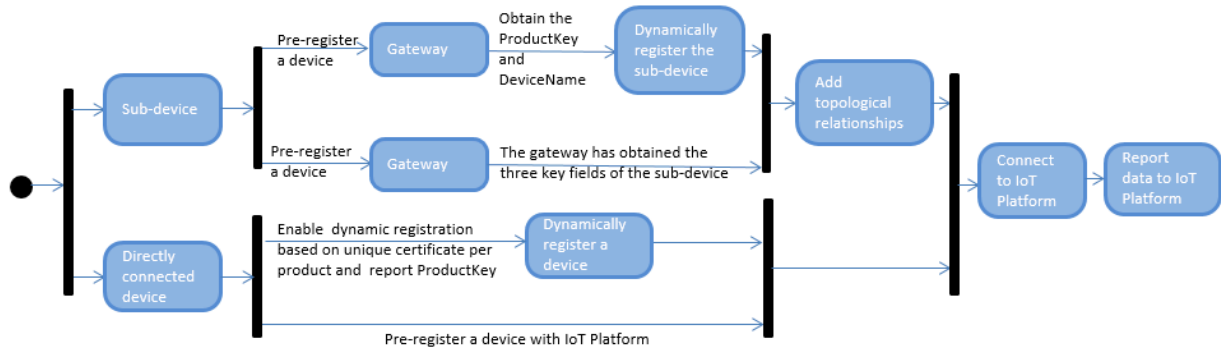
- If [#unique\\_34](#) is enabled, install the three key fields (ProductKey, DeviceName, and DeviceSecret) to the physical device for authentication, connect the device to IoT Platform, and report data to IoT Platform.
- If dynamic registration based on [#unique\\_35](#) is enabled, install the product certificate (ProductKey and ProductSecret) to the physical device for authentication, connect the device to IoT Platform, and report data to IoT Platform.

The gateway starts the connection process for sub-devices. Sub-devices can be connected to IoT Platform by using the following methods:

- If [#unique\\_34](#) is enabled, install the ProductKey, DeviceName, and DeviceSecret to the physical sub-device for authentication. The sub-device then sends these three key fields to the gateway, and the gateway adds the topological relationship and sends the data of the sub-device through the gateway connection 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 the gateway forwards the ProductKey and DeviceName to IoT Platform. IoT Platform then verifies the received DeviceName and sends a DeviceSecret to the sub-device

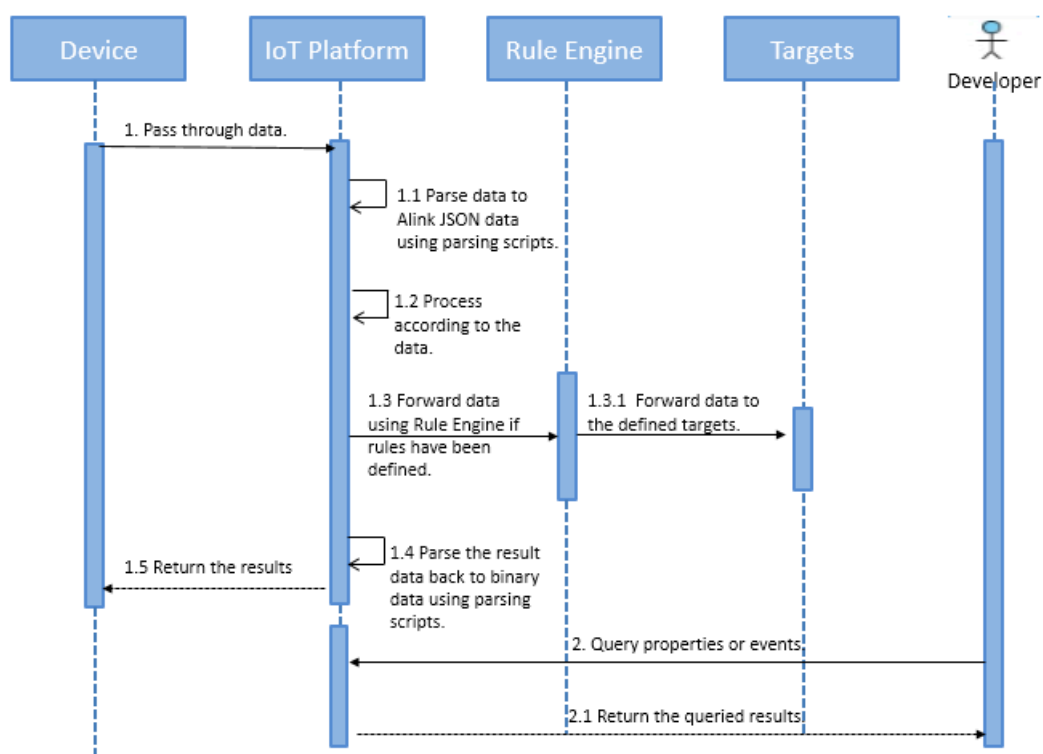


. The sub-device sends the obtained ProductKey, DeviceName, and DeviceSecret to the gateway, and the gateway adds the topological relationship and sends data to IoT Platform through the gateway connection channel.



### Devices report properties or events

- Pass through data (the data type is Do not parse/Custom.)



1. The device reports binary data to IoT Platform using the topic for pass through data.
2. IoT Platform parses the received data using the data parsing script that you have submitted on the IoT Platform console. The `rawDataToProtocol` method in the script is called to convert the binary data reported by the device to Alink JSON data that is used for processing.
3. that is used for processing.

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

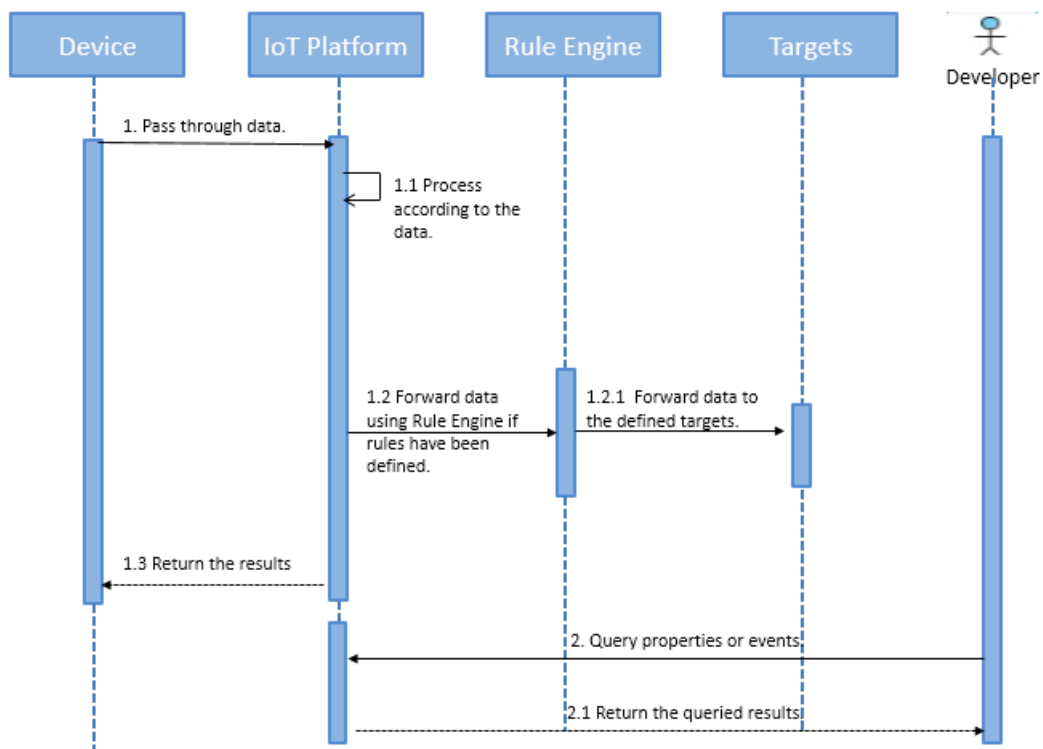
4. IoT Platform parses the returned data to binary data using the data parsing script that you have submitted on the IoT Platform console.
5. IoT Platform pushes the converted binary data to the device.



**Note:**

- The data forwarded by the rules engine is the data that has been parsed by the data parsing script.
- When you configure data forwarding using the rules engine, to obtain the device properties, use the topic: `/sys/{productKey}/{deviceName}/thing/event/property/post`.
- When you configure data forwarding using the rules engine, to obtain the device events, use the topic: `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`.

- Non-pass through (Alink JSON) data



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

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

### 3. IoT Platform returns the results to the device.

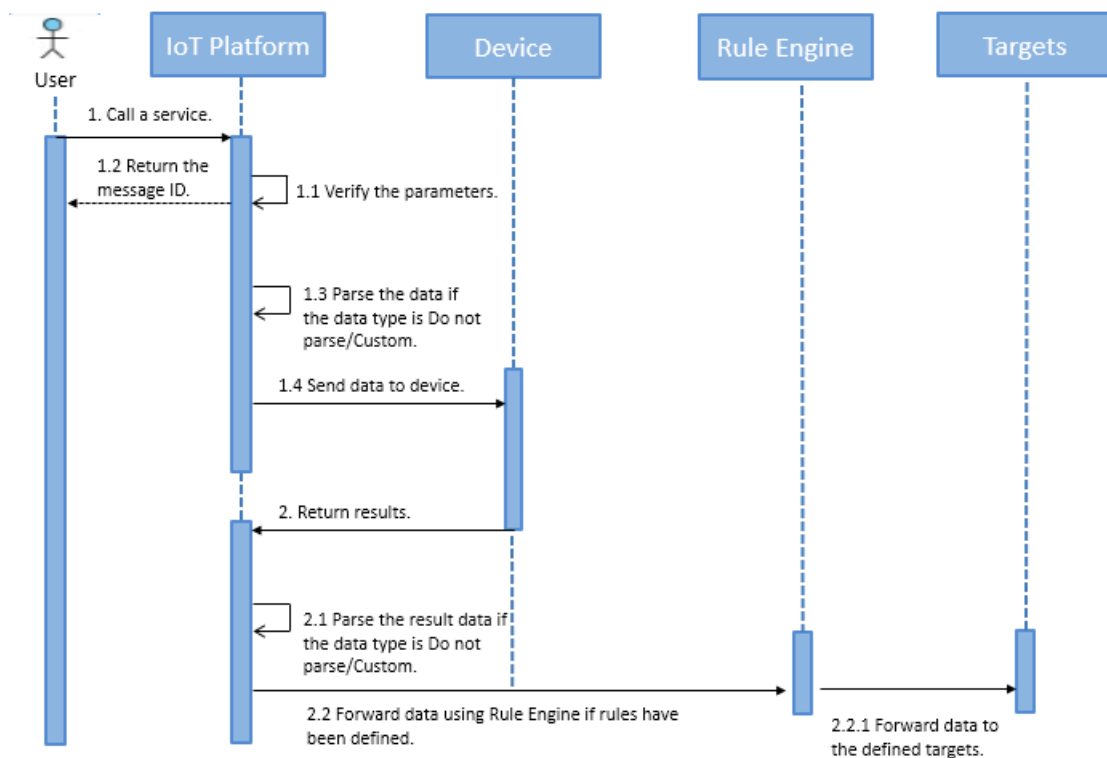


#### Note:

- When you configure data forwarding using the rules engine, to obtain the device properties, use the topic: `/sys/{productKey}/{deviceName}/thing/event/property/post`.
- When you configure data forwarding using the rules engine, to obtain the device events, use the topic: `/sys/{productKey}/{deviceName}/thing/event/{tsl.event.identifier}/post`.

## Call device services or set device properties

- Call device services or set device properties asynchronously



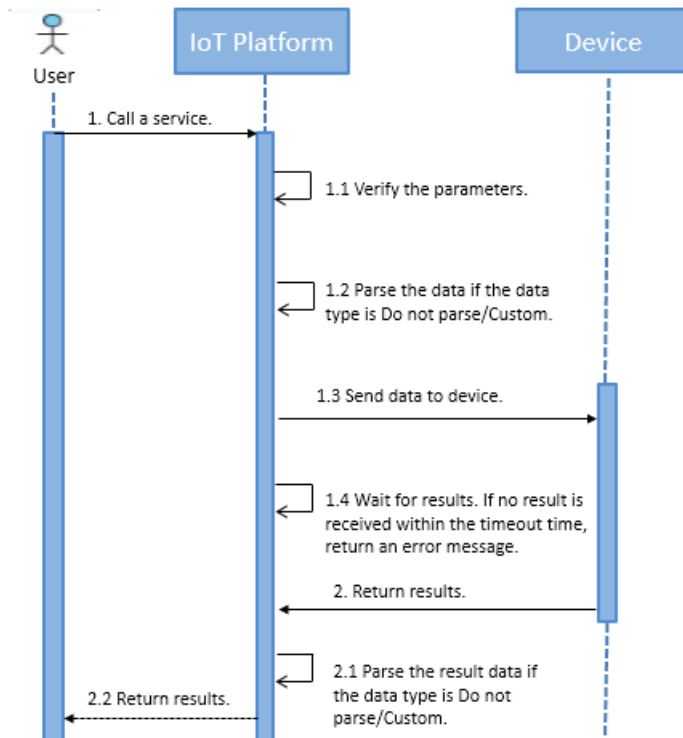
1. The user sets a device property or calls a device service using the asynchronous call method.
2. IoT Platform verifies the parameters.
3. IoT Platform uses the asynchronous call method to handle the request and return the results. If the call is successful, a message ID, which is to be sent to the device, is included in the response.

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

4. IoT Platform sends the data to the device, and the device handles the request asynchronously.
  - If the data is pass through (Do not parse/Custom) data, use the topic for pass through data.
  - If the data is non-pass through (Alink JSON) data, use the topic for non-pass through data.
5. After the device has completed the requested operation, it will return the results.
  - If the data type is pass through (Do not parse/Custom), IoT Platform will call the `rawDataToProtocol` method in the data parsing script to convert the data returned by the device.
  - If you have configured rules for data forwarding, the data will be forwarded to the targets according to the rules.

**Note:**

- When you configure data forwarding using the rules engine, use the topic: `/sys/{productKey}/{deviceName}/thing/downlink/reply/message` to obtain the calling results.
  - 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 device services and set device properties synchronously



1. The user calls a device service using the synchronous call method.

2. IoT Platform verifies the parameters.

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

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.

4. After the device has completed the requested operation, it will return the results. If a result is not received within the timeout period, a timeout error will be returned.

If the data type is pass through (Do not parse/Custom), IoT Platform will call the `rawDataToProtocol` method in the data parsing script to convert the data returned by the device.

5. IoT Platform returns the results to the user.

## 9.2 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": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.sub.register"
}
```

### Response message

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

### 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.
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: application/x-www-form-urlencoded
Content-Length: 123
productKey=1234556554&deviceName=deviceName1234&random=567345&sign=
adfvl23hdfdh&signMethod=HmacMD5
```

#### Response message

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

#### Parameter description

Parameter	Type	Description
productKey	String	ID of the product to which the device belongs.
deviceName	String	Name of the device
random	String	Random number.



Parameter	Type	Description
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_shal(productSecret, deviceNamedeviceName1234productKey1234556554random123)
```

## 9.3 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": "deviceName1234",
      "productKey": "123456554",
      "sign": "xxxxxx",
      "signmethod": "hmacSha1",
      "timestamp": "1524448722000",
      "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.
sign	String	Signature.
signmethod	String	Signing method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.

Parameter	Type	Description
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.

Signature algorithm

**Notice:**

IoT Platform supports common signature algorithms.

Sort all the parameters (except for **sign** and **signMethod**) that will be submitted to the server in lexicographical order, and then connect the parameters and values in turn (no connect symbols ).

Sign the signing parameters by using the algorithm specified by the signing method.

For example, in the following request, sort the parameters in **params** in alphabetic order and then sign the parameters.

```
sign= hmac_md5(deviceSecret, clientId123deviceNametestproductKey123timestamp1524448722000)
```

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": "deviceName1234",
      "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": "deviceName1234",
      "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.
method	String	Request method.
deviceName	String	Name of the sub-device.
productKey	String	Product ID of the sub-device.

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.

### 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": "deviceName1234",
      "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": "deviceName1234",
      "productKey": "1234556554"
    }
  ],
  "method": "thing.topo.add.notify"
}
```

Response data format when using the Alink protocol

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

Parameter description

Parameter	Type	Description
id	String	Message ID. 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.



## 9.4 Connect devices to IoT Platform

Make sure that a directly connected device has been registered with IoT Platform before connecting to IoT Platform.

Make sure that a sub-device has been registered with IoT Platform before connecting to IoT Platform. In addition, you also need to make sure that the topological relationship with the gateway has been added to the gateway. IoT Platform will verify the identity of the sub-device according to the topological relationship to identify whether the sub-device can use the gateway connection.

### Connect sub-devices to IoT Platform

Upstream

- Topic: /ext/session/{productKey}/{deviceName}/combine/login
- Reply topic: /ext/session/{productKey}/{deviceName}/combine/login\_reply

Request message

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

Response message

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

Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the value of the parameter for future use.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.

Parameter	Type	Description
productKey	String	ID of the product to which the sub-device belongs.
sign	String	Signature of a sub-device. Sub-devices use the same signature rules as the gateway.
signmethod	String	Sign method. The supported methods are hmacSha1, hmacSha256, hmacMd5, and Sha256.
timestamp	String	Timestamp.
clientId	String	Identifier of a device client. This parameter can have the same value as the ProductKey or DeviceName parameter.
cleanSession	String	A value of true indicates that when the device is offline, messages sent based on QoS =1 method will be cleared.
code	Integer	Result code. A value of 200 indicates that the request is successful.
message	String	Result message.
data	String	Additional information in the response, in JSON format.

**Notice:**

A gateway can accommodate a maximum of 200 concurrent online sub-devices. When the maximum number is reached, the gateway rejects any connection requests.

Sign the parameters

Sort all the parameters (except **sign** and **signmethod**) to be submitted to the server in alphabetical order, and then splice the parameters and values in turn (without splice symbols ).

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

Example:

```
sign= hmac_md5(deviceSecret, cleanSessiontrueclientId123deviceNameproductKey123timestamp123)
```

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 throttled because the device requests authentication to IoT Platform too frequently.
428	too many subdevices under gateway	Too many sub-devices connect to the gateway at the same time.
6401	topo relation not exist	The topological relationship between the gateway and the sub-device does not exist.
6100	device not found	The sub-device does not exist.
521	device deleted	The sub-device has been deleted.
522	device forbidden	The sub-device has been disabled.
6287	invalid sign	The password or signature of the sub-device is incorrect.

## Disconnect sub-devices from IoT Platform

Upstream

- Topic: /ext/session/{productKey}/{deviceName}/combine/logout
- Reply topic: /ext/session/{productKey}/{deviceName}/combine/logout\_reply

Request message

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

Response message

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

```
"code": 200,
"message": "success",
"data": ""
}
```

#### Parameter description

Parameter	Type	Description
id	String	Message ID. Reserve the parameter for future use.
params	Object	Request parameters.
deviceName	String	Name of the sub-device.
productKey	String	ID of the product to which the sub-device belongs.
code	Integer	Result code. A value of 200 indicates that the request is successful.
message	String	Result code.
data	String	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 information about sub-device connections, see [Connect sub-devices to IoT Platform](#). For information about error codes, see [Error codes](#).

## 9.5 Device properties, events, and services

If you have defined [TSL](#) mode for a product, the devices of this product can separately report data of the properties, events, and services that you have defined. For the data format of TSL, see [The TSL format](#). This topic describes how data is reported based on the TSL.

IoT Platform supports two data types: ICA Standard Data Format (Alink JSON) and Do not parse/Custom. When you are creating a product, you are required to select a data type for devices of this product. We recommend that you select the Alink JSON type.

- 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. For the data format, see the request examples and response examples in this topic.
- 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 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.

Write SQL

\* Rule Query Expression:

SELECT deviceName() as deviceName FROM "/sys/a15IBHNUUTJ/stre

\* Field:

deviceName() as deviceName

\* Topic :

sys

streamLA

streamLA001

/thing/event...

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": {
    "Power": {
      "value": "on",
      "time": 1524448722000
    },
    "WF": {
      "value": 23.6,
      "time": 1524448722000
    }
  },
  "method": "thing.event.property.post"
}
```

#### Response message

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

**Table 9-1: Request parameters**


Parameter	Type	Description
id	String	Message ID.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Request parameters. In the request example above, 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	Request method.

**Table 9-2: Response parameters**

Parameter	Type	Description
id	String	Message ID.

Parameter	Type	Description
code	Integer	Result code. See <a href="#">Common codes on devices</a> .
data	String	Data 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	<p>The TSL verification service is not available.</p> <p>IoT Platform verifies all the received properties according to the TSLs of products. This error is reported when a system exception occurs. For TSL definition, see <a href="#">What is Thing Specification Language (TSL)?</a>.</p> <div> <b>Note:</b> 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.</div>

#### 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: `/sys/{productKey}/{deviceName}/thing/downlink/reply/message`. You can 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:

\* Field:

\* Topic :

Condition:

Dropdown menu options:  


- /thing/event/property/post
- ✓ /thing/downlink/reply/message
- /thing/lifecycle

### Request message

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

### Response message

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



```
}
```

**Table 9-3: Request Parameters**

Parameter	Type	Description
id	String	Message ID.
version	String	Protocol version. Currently, the value is 1.0.
params	Object	Property parameters. In the request example above, the property to be set is <pre>{ "temperature": "30.5" }</pre>
method	String	Request method.

**Table 9-4: Response parameters**

Parameter	Type	Description
id	String	Message ID.
code	Integer	Result code. See <a href="#">Common codes on devices</a> .
data	String	Data 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

Q

/thing/event/property/post

/thing/event/Error/post

/thing/event/temp/post

/thing/event/Errors/post

/thing/downlink/reply/message

/thing/lifecycle

### Request message

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "Power": "on",
      "WF": "2"
    },
    "time": 1524448722000
  },
  "method": "thing.event.{tsl.event.identifier}.post"
}
```

### Response message

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

**Table 9-5: Request Parameters**

Parameter	Type	Description
id	String	Message ID.

Parameter	Type	Description
version	String	Protocol version. Currently, the value is 1.0.
params	List	Parameters of the reported events.
value	Object	The event information. In the request example above, the events are <pre>{  "Power": "on",  "WF": "2"}</pre>
time	Long	The UTC timestamp when the event occurs.
method	String	Request method.

**Table 9-6: Response parameters**

Parameter	Type	Description
id	String	Message ID.
code	Integer	Result code. See <a href="#">Common codes on devices</a> .
data	String	Data 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}/airCondition/thing/",  "profile": {    "productKey": "a123456789",    "deviceName": "airCondition"  },  "events": [    {      "identifier": "alarm",      "name": "alarm",      "desc": "Fan alarm",      "type": "alert",      "required": true,      "outputData": [        {
```

```

        "identifier": "errorCode",
        "name": "ErrorCode",
        "dataType": {
            "type": "text",
            "specs": {
                "length": "255"
            }
        }
    },
    ],
    "method": "thing.event.alarm.post"
}
]
}

```

The device reports this event:

```

{
  "id": "123",
  "version": "1.0",
  "params": {
    "value": {
      "errorCode": "error"
    },
    "time": 1524448722000
  },
  "method": "thing.event.alarm.post"
}

```



#### Note:

- `tsl.event.identifier` indicates the event identifier in the TSL. For TSL template, see [What is Thing Specification Language \(TSL\)?](#).
- 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`

Services can be called in two methods: synchronous method and asynchronous method. When you [define a service](#), you are required to select a method for the service.

- Synchronous method: IoT Platform uses the Revert-RPC method to push requests to devices. For the operation process of Revert-RPC method, see [What is RRPC](#).
- Asynchronous method: IoT Platform pushes requests to devices in an asynchronous manner, and the devices also 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: `/sys/{productKey}/{deviceName}/thing/downlink/reply/message`.

You can configure [Rules Engine](#) to forward service calling 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:

\* Field:

\* Topic :

Condition:

Dropdown menu options:  
   
☒

### 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 9-7: Request Parameters**

Parameter	Type	Description
id	String	Message ID.
version	String	Protocol version. Currently, the value is 1.0.
params	Map	Parameters used to call a service, including the identifier and value of the service. As in the example above: <pre>{   "Power": "on",   "WF": "2" }</pre>
method	String	Request method.

**Table 9-8: Response parameters**

Parameter	Type	Description
id	String	Message ID.
code	Integer	Result code. See <a href="#">Common codes on devices</a> .
data	String	Data 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 follow the definition of the service in the TSL.

## Examples

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

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

Request message for calling the service:

```
{
```

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

#### Response message

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



#### Note:

`tsl.service.identifier` indicates the identifier of the service in TSL. For how to define TSL, see [What is Thing Specification Language \(TSL\)?](#).

## 9.6 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": "",
    "digestMethod": "",
    "url": ""
  }
}
```

#### Parameter description

Parameter	Type	Description
id	String	The message ID.
version	String	The protocol version number. Default value: 1.0.



Parameter	Type	Description
method	String	The method is <code>thing.model.config.push</code> .
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":"",
    "digestMethod":"",
    "url":""
  }
}
```

### url data

```
{
  "modelList": [
    {
      "profile": {
        "productKey": "test01"
      },
      "services": [
        {
          "outputData": "",
          "identifier": "AngleSelfAdaption",
          "inputData": [
            {
              "identifier": "test01",
              "index": 0
            }
          ],
          "displayName": "test01"
        }
      ],
      "properties": [
        {
          "identifier": "identifier",
          "displayName": "test02"
        },
        {
          "identifier": "identifier_01",
```

```

        "displayName": "identifier_01"
      }
    ],
    "events": [
      {
        "outputData": [
          {
            "identifier": "test01",
            "index": 0
          }
        ],
        "identifier": "event1",
        "displayName": "abc"
      }
    ]
  },
  {
    "profile": {
      "productKey": "test02"
    },
    "properties": [
      {
        "originalDataType": {
          "specs": {
            "registerCount": 1,
            "reverseRegister": 0,
            "swap16": 0
          },
          "type": "bool"
        },
        "identifier": "test01",
        "registerAddress": "0x03",
        "scaling": 1,
        "operateType": "inputStatus",
        "pollingTime": 1000,
        "trigger": 1
      },
      {
        "originalDataType": {
          "specs": {
            "registerCount": 1,
            "reverseRegister": 0,
            "swap16": 0
          },
          "type": "bool"
        },
        "identifier": "test02",
        "registerAddress": "0x05",
        "scaling": 1,
        "operateType": "coilStatus",
        "pollingTime": 1000,
        "trigger": 2
      }
    ]
  }
],
"serverList": [
  {
    "baudRate": 1200,
    "protocol": "RTU",
    "byteSize": 8,
    "stopBits": 2,
    "parity": 1,
    "name": "modbus01",

```

```

        "serialPort": "0",
        "serverId": "D73251B4277742"
    },
    {
        "protocol": "TCP",
        "port": 8000,
        "ip": "192.168.0.1",
        "name": "modbus02",
        "serverId": "586CB066D6A34"
    },
    {
        "password": "XIJTginONohPEUAYZxLB7Q==",
        "secPolicy": "Basic128Rsa15",
        "name": "server_01",
        "secMode": "Sign",
        "userName": "123",
        "serverId": "55A9D276A7ED470",
        "url": "tcp:00",
        "timeout": 10
    },
    {
        "password": "hAaX5s13gwX2JwyvUkOafQ==",
        "name": "service_09",
        "secMode": "None",
        "userName": "1234",
        "serverId": "44895C63E3FF401",
        "url": "tcp:00",
        "timeout": 10
    }
],
"deviceList": [
    {
        "deviceConfig": {
            "displayNamePath": "123",
            "serverId": "44895C63E3FF4013924CEF31519ABE7B"
        },
        "productKey": "test01",
        "deviceName": "test_02"
    },
    {
        "deviceConfig": {
            "displayNamePath": "1",
            "serverId": "55A9D276A7ED47"
        },
        "productKey": "test01",
        "deviceName": "test_03"
    },
    {
        "deviceConfig": {
            "slaveId": 1,
            "serverId": "D73251B4277742D"
        },
        "productKey": "test02",
        "deviceName": "test01"
    },
    {
        "deviceConfig": {
            "slaveId": 2,
            "serverId": "586CB066D6A34E"
        },
        "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": "FeatureTest02"
              }
            ],
            "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": "FeatureTest01"
              },
              {
                "identifier": "test02",
                "dataType": {
                  "specs": {
                    "unit": "mm",
                    "min": "0",
                    "max": "1"
                  },
                  "type": "int"
                },
                "name": "FeatureTest02"
              }
            ],
            "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": "FeatureTest01",
        "accessMode": "r",
        "required": false
    },
    {
        "identifier": "test02",
        "dataType": {
            "specs": {
                "unit": "mm",
                "min": "0",
                "max": "1"
            },
            "type": "int"
        },
        "name": "FeatureTest02",
        "accessMode": "rw",
        "required": false
    }
],
"events": [
    {
        "outputData": [
            {
                "identifier": "test01",
                "dataType": {
                    "specs": {
                        "unit": "m",
                        "min": "0",
                        "max": "1"
                    },
                    "type": "int"
                },
                "name": "FeatureTest01"
            },
            {
                "identifier": "test02",
                "dataType": {
                    "specs": {
                        "unit": "mm",
                        "min": "0",
                        "max": "1"
                    },
                    "type": "int"
                },
                "name": "FeatureTest02"
            }
        ]
    }
],

```

```

        "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": "FeatureTest01"
          }
        ],
        "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": "FeatureTest01"
    }
],
"identifier": "get",
"inputData": [
    "identifier",
    "identifier_01"
],
"method": "thing.service.property.get",
"name": "get",
"required": true,
"callType": "async",
"desc": "Get properties",
},
{
    "outputData": [],
    "identifier": "AngleSelfAdaption",
    "inputData": [
        {
            "identifier": "test01",
            "dataType": {
                "specs": {
                    "min": "1",
                    "max": "10",
                    "step": "1"
                },
                "type": "int"
            },
            "name": "Parameter1",
        }
    ],
    "method": "thing.service.AngleSelfAdaption",
    "name": "adaptive angle calibration",
    "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": "FeatureTest01",
        "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": "FeatureTest01"
        }
      ],
      "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": "ParameterTest1"
        }
      ],
      "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": [
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      },
      "identifier": "test01",
      "registerAddress": "0x03",
      "scaling": 1,
      "operateType": "inputStatus",
      "pollingTime": 1000,
      "trigger": 1
    },
    {
      "originalDataType": {
        "specs": {
          "registerCount": 1,
          "reverseRegister": 0,
          "swap16": 0
        },
        "type": "bool"
      },
      "identifier": "test02",
      "registerAddress": "0x05",
      "scaling": 1,
      "operateType": "coilStatus",
      "pollingTime": 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: <ul style="list-style-type: none"> <li>- coilStatus</li> <li>- inputStatus</li> <li>- holdingRegister</li> <li>- inputRegister</li> </ul>
registerAddress	String	The register address.
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": "AngleSelfAdaption",
      "inputData": [
        {
          "identifier": "test01",
          "index": 0
        }
      ],
      "displayName": "test01"
    }
  ]
}
```

```
    },
    ],
    "properties": [
      {
        "identifier": "identifier",
        "displayName": "test02"
      },
      {
        "identifier": "identifier_01",
        "displayName": "identifier_01"
      }
    ],
    "events": [
      {
        "outputData": [
          {
            "identifier": "test01",
            "index": 0
          }
        ],
        "identifier": "event1",
        "displayName": "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": "D73251B4277742"
  },
  {
    "protocol": "TCP",
    "port": 8000,
    "ip": "192.168.0.1",
    "name": "modbus02",
    "serverId": "586CB066D6A34"
  }
]

```

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: <ul style="list-style-type: none"> <li>- E: Even parity check.</li> <li>- O: Odd parity check.</li> <li>- N: No parity check.</li> </ul>
serialPort	String	The serial port number.

- OPC UA protocol

```

{
  "password": "XIJTginONohPEUAYzxLB7Q==",
  "secPolicy": "Basic128Rsa15",
  "name": "server_01",
  "secMode": "Sign",
  "userName": "123",
  "serverId": "55A9D276A7ED470",
  "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.
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/PKCS5Padding";

private static String algorithm = "AES";

private static String charsetName = "utf-8";
/**
 * Encryption algorithm
 *
 * @param data (Data to be encrypted)
 * @param deviceSecret (The deviceSecret of the device)
 * @return
 */
public static String aesEncrypt(String data, String deviceSecret
) {
    try {
        Cipher cipher = Cipher.getInstance(instance);
        byte[] raw = deviceSecret.getBytes();
        SecretKeySpec key = new SecretKeySpec(raw, algorithm);
        IvParameterSpec ivParameter = new IvParameterSpec(
deviceSecret.substring(0, 16).getBytes());
        cipher.init(Cipher.ENCRYPT_MODE, key, ivParameter);
        byte[] encrypted = cipher.doFinal(data.getBytes(
charsetName));

        return new BASE64Encoder().encode(encrypted);
    }
```

```

        } catch (Exception e) {
            e.printStackTrace();
        }

        return null;
    }

    public static String aesDecrypt(String data, String deviceSecret
) {
        try {
            byte[] raw = deviceSecret.getBytes(charsetName);
            byte[] encrypted1 = new BASE64Decoder().decodeBuffer(
data);

            SecretKeySpec key = new SecretKeySpec(raw, algorithm);
            Cipher cipher = Cipher.getInstance(instance);
            IvParameterSpec ivParameter = new IvParameterSpec(
deviceSecret.substring(0, 16).getBytes());
            cipher.init(Cipher.DECRYPT_MODE, key, ivParameter);
            byte[] originalBytes = cipher.doFinal(encrypted1);
            String originalString = new String(originalBytes,
charsetName);
            return originalString;
        } catch (Exception ex) {
            ex.printStackTrace();
        }

        return null;
    }

    public static void main(String[] args) throws Exception {
        String text = "test123";
        String secret = "testTNmjyWHQzniA8wEkTNmjyWHQtest";
        String data = null;
        data = aesEncrypt(text, secret);
        System.out.println(data);
        System.out.println(aesDecrypt(data, secret));
    }

```

### deviceList description

- Modbus protocol

```

{
    "deviceConfig": {
        "slaveId": 1,
        "serverId": "D73251B4277742D"
    },
    "productKey": "test02",
    "deviceName": "test01"
}

```

### Parameter description

Parameter	Type	Description
deviceConfig	Object	The device information.
slaveId	Integer	The slave station ID.
serverId	String	The channel ID.

Parameter	Type	Description
productKey	String	The product ID.
deviceName	String	The name of the device.

- OPC UA protocol

```
{
  "deviceConfig": {
    "displayNamePath": "123",
    "serverId": "44895C63E3FF4013924CEF31519ABE7B"
  },
  "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.

## 9.7 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 <a href="#">Common codes on devices</a>

### 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.

## 9.8 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": "Temperature",
      "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. <ul style="list-style-type: none"><li>Length: Up to 100 bytes.</li><li>Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_).</li><li>The tag name must start with an English letter or underscore (_).</li></ul>
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": "Temperature"
    }
  ],
  "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. <ul style="list-style-type: none"><li>Length: Up to 100 bytes.</li><li>Valid characters: Lowercase letters a to z, uppercase letters A to Z, digits 0 to 9, and underscores (_).</li><li>The tag name must start with an English letter or underscore (_).</li></ul>
attrValue	String	Tag value.
code	Integer	Result code. A value of 200 indicates the request is successful.

### Error messages

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

## 9.9 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.dsltemplate.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/airCondition/thing/",
    "profile": {
      "productKey": "1234556554",
      "deviceName": "airCondition"
    },
    "properties": [
      {
        "identifier": "fan_array_property",
        "name": "Fan array property",
        "accessMode": "r",
        "required": true,
        "dataType": {
          "type": "array",
          "specs": {
            "size": "128",
            "item": {
              "type": "int"
            }
          }
        }
      }
    ]
  },
  "events": [
    {
      "identifier": "alarm",
      "name": "alarm",

```

```

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

```

```

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

```

Parameter descriptions:

Parameter	Type	Description
id	String	Message ID. 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.

Parameter	Type	Description
productKey	String	ProductKey. In the example, the ProductKey is 1234556554.
deviceName	String	Device name. In the example, the device name is airCondition.
data	Object	TSL model of the device. For more information, see <a href="#">What is Thing Specification Language (TSL)?</a>

#### 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.

## 9.10 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&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJlq6Ft5B2yfSjIpK6MGsyNlJx5jo6mVnfBgLIPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4flqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceuSbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltDUROFbIKP%2BpKWSKuGfLCLdysQcOlwEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTKnxR7ARAsaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fv4jN5JXQfAU8KLO8tRjofHWmojNzBJAAPPySSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54TOlo7%2F1wTLTsCUqzzeIiXVOK8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMqph2cKsr8y8UfWLC6IzvJsClXTnbJBMeuWIqo5zIynS1pm7gf%2F9N3hVc6%2BEeIk0xf12tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
    "md5": "93230c3bde425a9d7984a594ac55ea1e",
    "sign": "93230c3bde425a9d7984a594ac55ea1e",
    "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.

Parameter	Type	Description
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 information is available."
  }
}
```

### Parameter description

Parameter	Type	Description
id	String	Message ID.

Parameter	Type	Description
step	String	Firmware update progress information. Value range: <ul style="list-style-type: none"><li>• A value from 1 to 100 indicates the progress percentage.</li><li>• A value of -1 indicates the firmware update has failed.</li><li>• A value of -2 indicates that the firmware download has failed.</li><li>• A value of -3 indicates that firmware verification has failed.</li><li>• A value of -4 indicates that the firmware installation has failed.</li></ul>
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.

## 9.11 Remote configuration

This article introduces Topics and Alink JSON format requests and responses for remote configuration. 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": {
    "configScope": "product",
    "getType": "file"
  },
  "method": "thing.config.get"
}
```

Response message

```
{
  "id": "123",
  "version": "1.0",
  "code": 200,
  "data": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJlq6Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBg1IPTvlvt5D50Tz2IHtIf3NpAusdsv03nWxT7v4f1qFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu5bFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxB1tdUROfbIKP%2BpKWSKuGfLCldysQc0lwEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2%2FdtJOiTkxr7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fv4jN5JXQfAU8KLO8trjofHWmojNzBJAAPpySSy3Rvr7m5efQrryby1lLO6iZy%2BVio2VSZDxshI5Z3McKARWct06MWV9ABA2TTXXOi40BOxuq%2B3JGoABXC54Tolo7%2F1wTLTsCUqzzeIiXVOK8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMQph2cKsr8y8UfWLC6IzvJsClXTnbJBMeuWIqo5zIynSlpm7gf%2F9N3hVc6%2BEEIk0xfl2tycsUpbL2FoaGk6BAF8hWSWYUXsv59d5Uk%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 `/sys/{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:

\* Field:

\* Topic :

Condition:

Dropdown menu options:  


- /thing/event/property/post
- ☒ /thing/downlink/reply/message
- /thing/lifecycle

Request message:

```
{
  "id": "123",
  "version": "1.0",
  "params": {
    "configId": "123dagdah",
    "configSize": 1234565,
    "sign": "123214adfadgadg",
    "signMethod": "Sha256",
    "url": "https://iotx-config.oss-cn-shanghai.aliyuncs.com/nopoll_0.4.4.tar.gz?Expires=1502955804&OSSAccessKeyId=XXXXXXXXXXXXXXXXXXXX&Signature=XfgJu7P6DWWejstKJgXJEH0qAKU%3D&security-token=CAISuQJlq6"
```

```
Ft5B2yfSjIpK6MGsyN1Jx5jo6mVnfBgLIPTvlvt5D50Tz2IHtIf3NpAusdsV03nWxT7v4f
lqFyTINVAEvYZJOPKGrGR0DzDbDasumZsJbo4f%2FMQBqEaXPS2MvVfJ%2BzLrf0ceu
sbFbpjzJ6xaCAGxypQ12iN%2B%2Fr6%2F5gdc9FcQSkL0B8ZrFsKxBltDUROfbIKP%
2BpKWSKuGfLCldysQcOlwEP4K%2BkkMqH8Uic3h%2Boy%2BgJt8H2PpHhd9NhXuV2WMzn2
%2FdtJOiTkxr7ARasaBqhelc4zqA%2FPPlWgAKvkXba7aIoo01fv4jN5JXQfAU8KLO8tR
jofHWmojNzBJAAPpYSSy3Rvr7m5efQrrybY1lLO6iZy%2BVio2VSZDxshI5Z3McK
ARWct06MWV9ABA2TTXXOi40B0xuq%2B3JGoABXC54Tolo7%2F1wTLTsCUqzzeIiXVOK
8CfNOKfTucMGHkeYeCdFkm%2FkADhXAnrnGf5a4FbmKMqph2cKsr8y8UfWLC6Iz
vJsClXTnbJBMeuWIqo5zIynSlpm7gf%2F9N3hVc6%2BEeIk0xfl2tycsUpbL2
FoaGk6BAF8hWSWYUXsv59d5Uk%3D",
  "getType": "file"
},
"method": "thing.config.push"
}
```

### Response message

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

### Parameter description

Parameter	Type	Description
id	String	Message ID.
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.

Parameter	Type	Description
code	Integer	Result code. For more information, see Common codes on devices.

## 9.12 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.