Alibaba Cloud
物联网边缘计算

Developer Guide (Edge)

Issue: 20200506

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

Document conventions

Style	Description	Example
	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	 **Danger:** Resetting will result in the loss of user configuration data.
	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	 **Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	 **Notice:** If the weight is set to 0, the server no longer receives new requests.
	A note indicates supplemental instructions, best practices, tips, and other content.	 **Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings** > **Network** > **Set network type**.
Bold	Bold formatting is used for buttons, menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands.	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`

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

Contents

1 Device access SDK

1.1 SDK for C

This topic describes how to use the SDK for C and the related API. Link IoT Edge provides the SDK for C, which is named linkedge-thing-access-sdk-c.

For the source code of the SDK for C, visit Open-source SDK for C.

get_properties_callback

```
/*
    * The callback function that is used to obtain the device properties that are defined
 in the device TSL. Driver developers must implement the business logic to obtain the
properties.
    *
    * When Link IoT Edge needs to obtain the properties of a device, the SDK calls this
function to obtain the data, parse the data into a fixed format, and then sends it back to
Link IoT Edge.
    * Developers must identify the device by using device ID and property names. Then,
developers must assign the property values based on the @device_data_t format.
    *
    * @dev_handle: the device from which Link IoT Edge obtains the properties.
    * @properties: the property values. Driver developers must update the value of the
properties parameter based on the obtained property values.
    * @properties_count: the number of properties.
    * @usr_data: the private data specified by the user when the device is registered.
    * If all properties are obtained, LE_SUCCESS is returned. If the request fails, an error
code is returned. For more information, see the error code macros defined in the le_error
.h file.
    * */
typedef int (*get_properties_callback)(device_handle_t dev_handle,
                                       leda_device_data_t properties[],
                                       int properties_count,
                                       void *usr_data);
```

set_properties_callback

```
/*
    * The callback function that is used to set properties that are defined in the TSL. Driver
developers must implement the business logic.
    *
    * When Link IoT Edge needs to set the properties of a device, the SDK calls this function
 and sends the data to the edge application.
    * Developers must use this function to set device properties.
    *
    * @dev_handle: the device for which Link IoT Edge sets the properties.
    * @properties: the new property values to be set for the device.
    * @properties_count: the number of properties.
    * @usr_data: the private data specified by the user when the device is registered.
    *
    * If the request is successful, LE_SUCCESS is returned. If the request fails, an error code
is returned. For more information, see the error code macros defined in the le_error.h file.
    * */
typedef int (*set_properties_callback)(device_handle_t dev_handle,
```

```
                                const leda_device_data_t properties[],
                                int properties_count,
                                void *usr_data);
```

call_service_callback

```
/*
    * The callback function that is used to call a service that is based on the service
defined in the device TSL. Driver developers must implement the business logic for the
service.
    *
    * When Link IoT Edge needs to call a service of a device, the SDK calls this function to
pass specific service parameters to the application.
    * Developers must call the service in the function and assign the return value of the
service to the output_data parameter based on the @device_data_t format.
    *
    * @dev_handle: the specific device from which Link IoT Edge calls the service.
    * @service_name: the name of the service. The service name must be the same as that
 in the TSL.
    * @data: the parameters of the service. The parameters must be the same as the
parameters in the TSL of the device.
    * @data_count: the number of parameters of the service.
    * @output_data: the output data. Developers must set this parameter to the return
value of the service based on the format defined in the TSL of the device.
    * @usr_data: the private data specified by the user when the device is registered.
    *
    * If the request is successful, LE_SUCCESS is returned. If the request fails, an error code
is returned. For more information, see the error code macros defined in the le_error.h file.
    * */
typedef int (*call_service_callback)(device_handle_t dev_handle,
                                const char *service_name,
                                const leda_device_data_t data[],
                                int data_count,
                                leda_device_data_t output_data[],
                                void *usr_data);
```

leda_report_properties

```
/*
 * Reports properties. The property reporting capability is specified in the TSL model of
the product to which the device belongs.
 *
 * Reports properties. One or more properties can be reported together.
 *
 * dev_handle: the unique identifier of the device in Link IoT Edge.
 * properties: the @leda_device_data_t property array.
 * properties_count: the number of properties.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 *
 */
int leda_report_properties(device_handle_t dev_handle, const leda_device_data_t
properties[], int properties_count);
```

leda_report_event

```
/*
 * Reports device events defined in the device TSL.
 *
 *
```

```
 * dev_handle: the unique identifier of the device in Link IoT Edge.
 * event_name: the event name.
 * data: the event parameter array @leda_device_data_t.
 * data_count: the length of the event parameter array.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request fails, an error code is returned.
 *
 */
int leda_report_event(device_handle_t dev_handle, const char *event_name, const leda_device_data_t data[], int data_count);
```

leda_offline

```
/*
 * Disconnects a device. If a device is malfunctioning or before you shut down a device, you may want to disconnect the device. you can first disconnect the device. After the device is disconnected, Link IoT Edge will not continue to route messages to the device.
 *
 * dev_handle: the unique identifier of the device in Link IoT Edge.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request fails, an error code is returned.
 *
 */
int leda_offline(device_handle_t dev_handle);
```

leda_online

```
/*
 * Connects the device to IoT Platform. The device can be recognized by Link IoT Edge only after the device is connected to IoT Platform.
 *
 * dev_handle: the unique identifier of the device in Link IoT Edge.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request fails, an error code is returned.
 */
int leda_online(device_handle_t dev_handle);
```

leda_register_and_online_by_device_name

```
/*
 * Uses the device_name that has been created in the Alibaba Cloud IoT Platform to register, activate, and generate a unique identifier for the device.
 *
 * If you want to register multiple devices, you can call this function multiple times.
 *
 * product_key: the ProductKey of the product created in Alibaba Cloud IoT Platform.
 * device_name: the device name (DeviceName) that is created in the Alibaba Cloud IoT Platform.
 * device_cb: the device callback struct. For more information, see @leda_device_callback.
 * @usr_data: the private data specified by the user when the device is registered. The data is sent to the device during the callback.
 *
 * A blocking function. The unique identifier of the device in Link IoT Edge is returned. A value of 0 indicates that the device is valid and a value that is less than 0 indicates that it is invalid.
 *
 */
```

```
device_handle_t leda_register_and_online_by_device_name(const char *product_key,
const char *device_name, leda_device_callback_t *device_cb, void *usr_data);
```

leda_register_and_online_by_local_name

```
/*
 * Uses an on-premises custom device_name to register, activate, and generate a unique
 identifier for the device.
 *
 * If you want to register multiple devices, you can call this function multiple times.
 *
 * product_key: the ProductKey of the product created in Alibaba Cloud IoT Platform.
 * local_name: the name of the device. The names of products under the same product
must be unique.
 * device_cb: the device callback struct. For more information, see @leda_device_callback
.
 * @usr_data: the private data specified by the user when the device is registered. The
data is sent to the device during the callback.
 *
 * A blocking function. The unique identifier of the device in Link IoT Edge is returned. A
value of 0 indicates that the device is valid and a value that is less than 0 indicates that it
 is invalid.
 *
 * Note: You cannot use this function and the eda_register_and_online_by_device_name
function at the same time.
 * All devices of the same ProductKey must use the same function for device registration.
Otherwise, the registration may result in errors.
 */
device_handle_t leda_register_and_online_by_local_name(const char *product_key,
const char *local_name, leda_device_callback_t *device_cb, void *usr_data);
```

leda_init

```
/*
 * Initializes the driver module. During the initialization, a worker thread pool is created in
 the module to asynchronously process device operation requests from IoT Platform. You
can set the number of worker threads by specifying the worker_thread_nums parameter.
 *
 * worker_thread_nums: the number of threads in the thread pool. This value is set based
 on the number of registered devices.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 */
int leda_init(int worker_thread_nums);
```

leda_exit

```
/*
 * Terminates the driver module.
 *
 * Before the module exits, the resources are released.
 *
 * A blocking function.
 */
void leda_exit(void);
```

leda_get_driver_info_size

```
/*
```

```
 * Obtains the length of the driver information.
 *
 * A blocking function. If the length of the driver information is obtained, LE_SUCCESS is
returned. If the request fails, 0 is returned.
 */
int leda_get_driver_info_size(void);
```

leda_get_driver_info

```
/*
 * Obtains the information of the driver that is configured on IoT Platform.
 *
 * driver_info: the driver information. You must allocate the memory in advance.
 * size: the driver information length. If the size of the driver_info is larger than the size
parameter, LE_ERROR_INVAILD_PARAM is returned.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 *
 * The configuration format:
   {
      "json":{
         "ip":"127.0.0.1",
         "port":54321
      },
      "kv":[
         {
            "key":"ip",
            "value":"127.0.0.1",
            "note":"IP address"
         },
         {
            "key":"port",
            "value":"54321",
            "note":"Port number"
         }
      ],
      "fileList":[
         {
            "path":"device_config.json"
         }
      ]
   }
 */
int leda_get_driver_info(char *driver_info, int size);
```

leda_get_device_info_size

```
/*
 * Obtains the length of the device information.
 *
 * A blocking function. If the length of the device information is obtained, LE_SUCCESS is
returned. If the request fails, 0 is returned.
 */
int leda_get_device_info_size(void);
```

leda_get_device_info

```
/*
 * Obtains the information of the device that is configured on IoT Platform.
 *
```

```
 * driver_info: the device information. You must allocate the memory in advance.
 * size: the device information length. You can obtain the length by calling the
leda_get_device_info_size method. If the size of the device_info is larger than the size
parameter, LE_ERROR_INVAILD_PARAM is returned.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 *
 * The configuration format:
   [
      {
         "custom":{
            "port":12345,
            "ip":"127.0.0.1"
         },
         "deviceName":"device1",
         "productKey":"a1ccxxeypky"
      }
   ]
 */
int leda_get_device_info(char *device_info, int size);
```

leda_get_config_size

```
/*
 * Obtains the length of the driver configuration information.
 *
 * A blocking function. If the length of the driver configuration is obtained, LE_SUCCESS is
returned. If the request fails, 0 is returned.
 */
int leda_get_config_size(void);
```

leda_get_config

```
/*
 * Obtains all driver configurations.
 *
 * config: the driver configuration. You must allocate the memory in advance.
 * size: the device information length. You can obtain the length by calling the leda_get_c
onfig_size method. If the size of the device_info is larger than the size parameter,
LE_ERROR_INVAILD_PARAM is returned.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 *
 * The configuration format:
   {
      "config":{
         "json":{
            "ip":"127.0.0.1",
            "port":54321
         },
         "kv":[
            {
               "key":"ip",
               "value":"127.0.0.1",
               "note":"IP address"
            },
            {
               "key":"port",
               "value":"54321",
               "note":"Port number"
```

```
            }
        ],
        "fileList":[
            {
                "path":"device_config.json"
            }
        ]
    },
    "deviceList":[
        {
            "custom":"{"port":12345,"ip":"127.0.0.1"}",
            "deviceName":"device1",
            "productKey":"a1ccxxeypky"
        }
    ]
  }
 */
int leda_get_config(char *config, int size);
```

config_changed_callback

```
/*
 * The callback method that is used to modify driver configurations.
 *
 * config: the configuration information.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 */
typedef int (*config_changed_callback)(const char *config);
```

leda_register_config_changed_callback

```
/*
 * Configure the subscription-driven configuration change listener for callbacks.
 *
 * config_cb: the callback function that is used to notify configuration changes.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request
fails, an error code is returned.
 */
int leda_register_config_changed_callback(config_changed_callback config_cb);
```

leda_get_tsl_size

```
/*
 * Obtains the length of the TSL based on the ProductKey.
 *
 * product_key: the ProductKey of the product.
 *
 * A blocking function. If the length of the TSL is obtained, LE_SUCCESS is returned. If the
request fails, 0 is returned.
 */
int leda_get_tsl_size(const char *product_key);
```

leda_get_tsl

```
/*
 * Obtains the TSL information of a product based on the ProductKey.
 *
```

```
 * product_key: the ProductKey of the product.
 * tsl: the content of the TSL model. You need to apply for memory resources in advance to upload the data.
 * size: the TSL content size. You can obtain the size by calling the leda_get_tsl_size method. If the size of the tsl parameter is larger than the size parameter, LE_ERROR_INVAILD_PARAM is returned.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request fails, an error code is returned.
 */
int leda_get_tsl(const char *product_key, char *tsl, int size);
```

leda_get_tsl_ext_info_size

```
/*
 * Obtains the extended TSL information length of a product based on the ProductKey.
 *
 * product_key: the ProductKey of the product.
 *
 * A blocking function. If the request is successful, the length of the extended TSL information is returned. If the request fails, 0 is returned.
 */
int leda_get_tsl_ext_info_size(const char *product_key);
```

leda_get_tsl_ext_info

```
/*
 * Obtains the extended TSL information of a product based on the ProductKey.
 *
 * product_key: the ProductKey of the product.
 * tsl_ext_info: the extended TSL information. You need to request memory information in advance.
 * size: the length of the extended TSL information. You can obtain the length by calling the leda_get_tsl_ext_info_size method. If the size of the device_info is larger than the size parameter, LE_ERROR_INVAILD_PARAM is returned.
 *
 * A blocking function. If the request is successful, LE_SUCCESS is returned. If the request fails, an error code is returned.
 */
int leda_get_tsl_ext_info(const char *product_key, char *tsl_ext_info, int size);
```

leda_get_device_handle

```
/*
 * Obtains the device handle.
 *
 * product_key: the ProductKey of the product.
 * device_name: the device name (DeviceName).
 *
 * A blocking function. If the request is successful, a device handle is returned. If the request fails, a number that is less than 0 is returned.
 */
```

```
device_handle_t leda_get_device_handle(const char *product_key, const char *
device_name);
```

1.2 SDK for Node.js

You can develop a driver by using the SDK for Node.js to connect a device to a gateway. Then connect the device to the IoT Platform.

For the source code of the SDK for Node.js, visit Open-source SDK for Node.js.

Installation and usage

1. Run the following command to install the SDK:

```
npm install linkedge-thing-access-sdk
```

> **Note:**
> You can also use the development tools provided by Link IoT Edge together with the SDK to develop drivers.

2. After the SDK is installed, you can develop drivers based on the SDK interface.

> **Notice:**
> If you run a driver directly after the driver development is completed, an error will occur. You must deploy the driver to the gateway on the IoT Platform console before you run the driver. For more information about how to deploy the driver to the gateway, see Driver Development.

The following is an example of how to use the SDK to develop a driver:

```
const {
  Config,
  ThingAccessClient
} = require('linkedge-thing-access-sdk');

const callbacks = {
  setProperties: function (properties) {
    // Set properties to the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return {
      code: 0,
      message: 'success',
    };
  },
  getProperties: function (keys) {
    // Get properties from the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return {
      code: 0,
      message: 'success',
```

```
      params: {
        key1: 'value1',
        key2: 'value2',
      }
    };
  },
  callService: function (name, args) {
    // Call services on the physical thing and return the result.
    // Return an object representing the result or the promise wrapper of the object.
    return new Promise((resolve) => {
      resolve({
        code: 0,
        message: 'success',
      });
    });
  }
};
Config.get()
  .then(config => {
    const thingInfos = config.getThingInfos();
    thingInfos.forEach(thingInfo => {
      const client = new ThingAccessClient(thingInfo, callbacks);
      client.registerAndOnline()
        .then(() => {
          return new Promise(() => {
            setInterval(() => {
              client.reportEvent('high_temperature', { temperature: 41 });
              client.reportProperties({ 'temperature': 41 });
            }, 2000);
          });
        })
        .catch(err => {
          console.log(err);
          client.cleanup();
        });
        .catch(err => {
          console.log(err);
        });
    });
  });
```

Constants

Parameter	Type	Description
PRODUCT_KEY	String	A key in the configuration object. The key is passed to the **ThingAccessClient** constructor. Set the value to the ProductKey that was allocated to you by IoT Platform.
DEVICE_NAME	String	A key in the configuration object. The key is passed to the **ThingAccessClient** constructor. Set the value to the DeviceName that was allocated to you by IoT Platform.

Parameter	Type	Description
LOCAL_NAME	String	A key in the configuration object. The key is passed to the **ThingAccessClient** constructor. Set the value to the on-premises device name.
CALL_SERVICE	String	A key in the callback object. The key is passed to the **ThingAccessClient** constructor. Set the value to the callback function that is used to call device services. For more information about the function definition, see **callbacks.callService()**.
GET_PROPERTIES	String	A key in the callback object. The key is passed to the **ThingAccessClient** constructor. Set the value to the callback function that is used to obtain device properties. For more information about the function definition, see **callbacks.getProperties()**.
SET_PROPERTIES	String	A key in the callback object. The key is passed to the **ThingAccessClient** constructor. Set the value to the callback function that is used to set device properties. For more information about the function definition, see **callbacks.setProperties()**.
RESULT_SUCCESS	Number	The operation was successful. It is a status code used by the callback functions.
RESULT_FAILURE	Number	The operation failed. It is a status code used by the callback functions.
ERROR_CLEANUP	String	The error code for the **cleanup()** function.
ERROR_CONNECT	String	The error code for the **registerAndOnline()** function.
ERROR_DISCONNECT	String	The error code for the **offline()** function.
ERROR_GET_CONFIG	String	The error code for the **getConfig()** function.
ERROR_GET_TSL	String	The error code for the **getTsl()** function.
ERROR_GET_TSL_EXT_INFO	String	The error code for the **getTslExtInfo()** function.
ERROR_UNREGISTER	String	The error code for the **unregister()** function.

Config

The information related to driver configurations.

- **static get()**

  Returns the global driver configuration object, which is generated by the system when the device is associated with the driver.

  > **Note:**
  > The difference between the **Config.get()** function and the **getConfig()** function is that **getConfig()** returns a configuration string whereas **Config.get()** returns a configuration object.

  Response:

  ```
  Promise<Config>
  ```

- **static registerChangedCallback(callback)**

  The callback function that is used to make configuration changes

  Table 1-1: Request parameters

Parameter	Type	Description
callback	Function	The callback function that is called when a configuration change occurs.

- **static unregisterChangedCallback(callback)**

  The callback function that is used to revert configuration changes.

  Table 1-2: Request parameters

Parameter	Type	Description
callback	Function	The callback function that is called when a configuration change occurs.

- **Config(string)**

  Constructs a new Config object based on a configuration string.

  Table 1-3: Request parameters

Parameter	Type	Description
string	String	The JSON configuration string.

- **getThingInfos()**

  Returns all device-related information.

  Response:

  ```
  ThingInfo[]
  ```

- **getDriverInfo()**

  Returns the driver information.

  Response:

  ```
  Object
  ```

ThingInfo

This function can be used to identify the information of devices that are connected to Link IoT Edge. After a device is connected to Link IoT Edge, information such as the ProductKey, DeviceName, and custom configurations of the device can be identified by the function.

ThingInfo(productKey, deviceName, custom)

Constructs a new ThingInfo object.

Table 1-4: Request parameters

Parameter	Type	Description
productKey	String	The unique identifier of the product.
deviceName	String	The name of the device.
custom	Object	Custom device configurations.

ThingAccessClient

The client that allows devices to connect to IoT Platform. You can use the client to report properties and events. You can also use the client to send commands from the cloud to the devices.

- **ThingAccessClient(config, callbacks)**

  The constructor, which is initialized by specifying the config and callbacks parameters.

  Table 1-5: Request parameters

Parameter	Type	Description
config	Object	The metadata that is used to configure the client. Format: `{ "productKey": "Your Product Key", "deviceName": "Your Device Name" }`
callbacks	Object	The callback function object. Format: `callbacks: { setProperties: function(properties) {}, getProperties: function(keys) {}, callService: function(name, args) {} }` - For callback parameters of setting device properties, see the **callbacks.setProperties** section. - For callback parameters of obtaining device properties, see the **callbacks.getProperties** section. - For callback parameters of calling device services, see the **callbacks.callService** section.

- **callbacks.setProperties(properties)**

  The callback function that is used to set device properties. You can use the callback function to set device properties.

  Table 1-6: Request parameters

Parameter	Type	Description
properties	Object	The property object. The format of the property value is as follows: `{ "key1": "value1", "key2": "value2" }`

  Response:

```
{
  "code": 0,
  "message": "string",
  "params": {}
}
```

  Table 1-7: Response Parameters

Parameter	Type	Description
code	Number	The status code. - 0: indicates that the call was successful. - Non-zero: indicates that the call failed and an error corresponding to the non-zero value was returned.
message	String	Optional. The status description information.
params	Object	Optional. This parameter can be used to return the configuration result of each property. The parameter contains the actual values of the properties.

- **callbacks.getProperties(keys)**

  The callback function that is used to obtain specified device properties. You can use the callback function to obtain the properties of the device.

  Table 1-8: Request parameters

Parameter	Type	Description
keys	String[]	The names of the properties. The format is as follows: `['key1', 'key2']`

  Response:

```
{
  "code": 0,
  "message": "string",
  "params": {}
}
```

  Table 1-9: Response Parameters

Parameter	Type	Description
code	Number	The status code. - 0: indicates that the call was successful. - Non-zero: indicates that the call failed and an error corresponding to the non-zero value was returned.
message	String	Optional. The status description information.
params	Object	Optional. This parameter can be used to return the value of the specified properties if the request is successful.

- **callbacks.callService(name, args)**

  The callback function that is used to call device services. You can use a callback function to call the device service.

  Table 1-10: Request parameters

Parameter	Type	Description
name	String	The name of the device service.
args	Object	The list of service input parameters. The format is as follows: `{ "key1": "value1", "key2": "value2" }`

  Table 1-11: Response Parameters

Parameter	Type	Description
code	Number	The status code. - 0: indicates that the call was successful. - Non-zero: indicates that the call failed and an error corresponding to the non-zero value was returned.
message	String	Optional. The status description information.
params	Object	Optional. This parameter can be used to return additional information when the service is called.

- **registerAndOnline()**

  Registers the device on the gateway and notifies the gateway to connect the device to IoT Platform. You must register and connect the devices to IoT platform before they can receive commands from or send data to IoT platform.

  Response:

  ```
  Promise<Void>
  ```

- **online()**

  Notifies the gateway that the device is online. The function is used when the device is offline and then is online again.

  Response:

  ```
  Promise<Void>
  ```

- **offline()**

  Notifies the gateway that the device is offline.

  Response:

  ```
  Promise<Void>
  ```

- **reportEvent(eventName, args)**

  Reports device events to IoT Platform.

  Table 1-12: Request parameters

Parameter	Type	Description
eventName	String	The event name, which is the same as the name of the event you specified in the product definition.

Parameter	Type	Description
args	Object	The keys and values in the event. The format is as follows: `{ "key1": "value1", "key2": "value2" }`

- **reportProperties(properties)**

  Reports device properties to IoT Platform.

  Table 1-13: Request parameters

Parameter	Type	Description
properties	Object	The keys and values in the properties. The format is as follows: `{ "key1": "value1", "key2": "value2" }`

- **getTsl()**

  Returns the TSL string of the device. The data format is the same as in IoT Platform.

  Response:

  ```
  Promise<Void>
  ```

- **getTslExtInfo()**

  Returns the extended information of the TSL.

  Response:

  ```
  Promise<String>
  ```

- **cleanup()**

  Recycles resources. You can use this function to free up resources.

  Response:

  ```
  Promise<Void>
  ```

- **unregister()**

  Removes a device from the gateway. Use this function with caution.

Response:

```
Promise<Void>
```

getConfig()

Obtains the driver configuration string. which is generated by the system when the device is associated with the driver.

> **Note:**
> The difference between the **static get()** function (**Config.get()**) is that **getConfig()** returns a configuration string whereas **Config.get()** returns a configuration object.

Response:

```
Promise<String>
```

destroy()

Destroys all resources used by the SDK. You can call the **destroy()** function when you no longer want to use the SDK.

Response:

```
Promise<Void>
```

1.3 SDK for Python

Link IoT Edge provides the SDK for Python. The SDK is named lethingaccesssdk. This topic describes how to use the SDK for Python.

For more information about the source code of the SDK, visit Open-source SDK for Python.

Installation and usage

1. Run the following command to install the SDK:

```
pip3 install lethingaccesssdk
```

> **Note:**
> You can also use the development tools provided by Link IoT Edge to use the SDK to develop drivers.

2. After the SDK is installed, you can develop drivers based on the SDK interfaces.

> **Notice:**
> After the driver development is completed, running the driver directly will generate an error. You must deploy the driver to the gateway on the IoT Platform console before you run the driver. For more information about how to deploy the driver to the gateway, see Driver development.

The following is an example of how to use the SDK to develop a driver:

```
# -*- coding: utf-8 -*-
import logging
import time
import lethingaccesssdk
from threading import Timer
# Base on device, User need to implement the getProperties, setProperties and
callService function.
class Temperature_device(lethingaccesssdk.ThingCallback):
    def __init__(self):
        self.temperature = 41
        self.humidity = 80
    def getProperties(self, input_value):
        '''
        Get properties from the physical thing and return the result.
        :param input_value:
        :return:
        '''
        retDict = {
            "temperature": 41,
            "humidity": 80
        }
        return 0, retDict
    def setProperties(self, input_value):
        '''
        Set properties to the physical thing and return the result.
        :param input_value:
        :return:
        '''
        return 0, {}
    def callService(self, name, input_value):
        '''
        Call services on the physical thing and return the result.
        :param name:
        :param input_value:
        :return:
        '''
        return 0, {}
def thing_behavior(client, device):
    while True:
        properties = {"temperature": device.temperature,
                      "humidity": device.humidity}
        client.reportProperties(properties)
        client.reportEvent("high_temperature", {"temperature": 41})
        time.sleep(2)
try:
    thing_config = lethingaccesssdk.Config().getThingInfos()
    for config in thing_config:
        device = Temperature_device()
```

```
        client = lethingaccesssdk.ThingAccessClient(config)
        client.registerAndonline(device)
        t = Timer(2, thing_behavior, (client, device))
        t.start()
except Exception as e:
    logging.error(e)
# don't remove this function
def handler(event, context):
    return 'hello world'
```

Config

The information related to driver configurations.

- **Config()**

  Constructs a new Config object based on a configuration string.

- **getThingInfos()**

  Returns all device-related information.

  The return values are described as follows:

  Returns the encapsulated configuration information that the device uses to connect to Link IoT Edge.

  Table 1-14: Response Parameters

Parameter	Type	Description
ThingInfo	List	The device information.

  Table 1-15: ThingInfo description

Parameter	Type	Description
productKey	String	The unique identifier of the product.
deviceName	String	The name of the device.
custom	Object	The custom device configurations.

- **getDriverInfo()**

  Returns the driver information.

  Response:

  ```
  dict
  ```

ThingCallback

Create a class (for example, Demo_device) that inherits ThingCallback based on the physical device. Then implement the **setProperties**, **getProperties**, and **callService** functions in the class (Demo_device).

- **setProperties**

  Sets device properties.

  Table 1-16: Request Parameters

Parameter	Type	Description
properties	Dict	The property object. The format of the property value is as follows: `{ "property1": "value1", "property2": "value2" }`

  Table 1-17: Response Parameters

Parameter	Type	Description
code	Integer	If the request is successful, 0 is returned. If the request fails, a non-zero error code is returned.
output	Dict	The returned data. You can customize the content. Example: `{ "key1": xxx, "key2": yyy, ... }` If no data is returned, {} is returned.

- **getProperties**

  Obtains device properties.

  Table 1-18: Request Parameters

Parameter	Type	Description
keys	List	The names of the properties. The format is as follows: `['key1', 'key2']`

  Table 1-19: Response Parameters

Parameter	Type	Description
code	Integer	If the request is successful, 0 is returned. If the request fails, a non-zero error code is returned.
output	Dict	The return value. Example: `{ 'property1': xxx, 'property2': yyy, ..}.`

- **callService**

  Calls a device service.

  Table 1-20: Request Parameters

Parameter	Type	Description
name	String	The name of the device service.

Parameter	Type	Description
args	Dict	The list of service input parameters. The format is as follows: `{ "key1": "value1", "key2": "value2" }`

Table 1-21: Response Parameters

Parameter	Type	Description
code	Integer	If the request is successful, 0 is returned. If the request fails, a non-zero error code is returned.
output	Dict	The returned data. You can customize the content. Example: `{ "key1": xxx, "key2": yyy, ... }` If no data is returned, {} is returned.

ThingAccessClient(config)

The client that allows devices to connect to IoT Platform. You can use the client to report properties and events. You can also use the client to send commands from the cloud to the devices.

Table 1-22: Request Parameters

Parameter	Type	Description

config	Dict	The productKey and deviceName assigned by IoT Platform.
		Example:
		{
		"productKey": "xxx",
		"deviceName": "yyy"
		}

- **registerAndOnline(ThingCallback)**

  Registers the device on the gateway and notifies the gateway to connect the device to IoT Platform. You must register and connect the devices to IoT platform before they can receive commands from or send data to IoT platform.

  Table 1-23: Request Parameters

Parameter	Type	Description
ThingCallback	Object	The callback object of the device.

- **reportProperties(properties)**

  Reports device properties to IoT Platform.

  Table 1-24: Request Parameters

Parameter	Type	Description
properties	Dict	The keys and values in the properties. The format is as follows: `{ "key1": "value1", "key2": "value2" }`

- **reportEvent(eventName, args)**

  Reports device events to IoT Platform.

  Table 1-25: Request Parameters

Parameter	Type	Description
eventName	String	The event name, which is the same as the name of the event you specified in the product definition.

Parameter	Type	Description

args	Dict	The keys and values in the event. The format is as follows:
		{
		"key1": "value1",
		"key2": "value2"
		}

- **getTsl()**

  Returns the TSL string of the device. The data format is the same as in IoT Platform.

  Response:

  ```
  A TSL string
  ```

- **getTslExtInfo()**

  Returns the extended information of the TSL.

  Response:

  ```
  The extended information of the TSL.
  ```

- **online()**

  Notifies the gateway that the device is online. The interface is used when the device is offline and then is online again.

- **offline()**

  Notifies the gateway that the device is offline.

- **cleanup()**

  Recycles resources. You can use this interface to free up resources.

- **unregister()**

  Unbinds a device from a gateway. Use this interface with caution.

getConfig()

Obtains information related to driver configurations.

The return value is a driver configuration string.

2 Function Compute SDK

2.1 Overview

The SDK for Edge Function Compute mainly includes interfaces for device operations, message publishing, and function calls.

Function

The following sections describe the functions of the SDK for Edge Function Compute.

- Device operations

  Device operations are the most common requirements in IoT scenarios. To facilitate reading and writing device information, the Edge Function Compute SDK provides the following four types of API operations:

  - Obtain a device property on IoT Platform
  - Configure properties of a device
  - Invoke a service on a device
  - Obtain a device list based on tags
- Message publishing

  The SDK provides functions that publishes messages to a specified topic. You can also use the message routing feature to forward messages to other functions that have subscribed to the topic and forward messages to the cloud.
- Function calls

  A Function Compute-based application is a type of event-driven programming. Event sources can be device messages and timers, or calls to other functions. Each Function Compute-based application exists as an independent program. You can also invoke a Function Compute-based application the way you invoke a function. The application can accept input parameters and return processed results.

Install the SDK

The Edge Function Compute SDK provides the Node.js version and the Python version. Both versions have been integrated into the Link IoT Edge software package and can be directly used.

1. Obtain the SDK Library.
   - For the source code of the SDK for Node.js, see Link IoT Edge SDK for Node.js.
   - For the source code of the SDK for Python, see Link IoT Edge SDK for Python.
2. Import the SDK.
   - Import the SDK for Node.js.

     ```
     const leSdk = require('linkedge-core-sdk');
     ```
   - Import the SDK for Python.

     ```
     import lecoresdk
     ```

2.2 Nodejs

2.2.1 Publish

Publishes a message to a specified topic.

publish(params,callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err)	function	The callback function. The callback function must follow standard JavaScript practices. • If the call is successful, err is null. • If the call fails, err contains the error information.

Table 2-1: Parameter descriptions of params

Parameter	Type	Description
topic	String	The topic that receives the message.
payload	String	The message payload.

Sample requests

In the following example, each time an external event is triggered, a message is sent to the `/topic/hello` topic.

```
'use strict';
```

```
const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();

exports.handler = function (event, context, callback) {
  var message = {
    topic: '/hello/world',
    payload: 'Hello World.',
  };
  iotData.publish(message, (err, data)=> {
    if (err == null) {
      console.log("-- Publish HelloWorld success.");
    }
    callback(err);
  });
};
```

2.2.2 getThingProperties

Obtains the value of a property for a specified device.

getThingProperties(params, callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err, data)	function	The callback function. The callback function must follow standard JavaScript practices. For more information, see the table Parameter descriptions of callback.

Table 2-2: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the devices belong. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
payload	Array	The array of identifiers for the device properties that you want to obtain. If the call is sucessful, the current values of the properties are returned. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **View TSL** to view the identifier of each service.

Table 2-3: Parameter descriptions of callback

Parameter	Type	Description
err	Error	• If the call is successful, err is null. • If the call fails, err contains the error information.
data	object	The value of the specified device properties.

Sample requests

The following example obtains the LightSwitch property value of the LightDev2 device. The LightSwitch property indicates whether the device is switched on.

```
'use strict';

const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();

const getPropertiesParams = {
  productKey: 'a1ZJTVsqj2y', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
  payload: ['LightSwitch']   // The property defined in the Product TSL.
};
/* Promise wrapper for getThingProperties. */
function getThingProperties(params) {
  return new Promise((resolve, reject) => {
    iotData.getThingProperties(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}

exports.handler = function (event, context, callback) {
  getThingProperties(getPropertiesParams).then((data) => {
    console.log(data); // Return value example: { LightSwitch: 0 }
    if (null == data.LightSwitch) {
      console.log("-- [Warn] No property LightSwitch return.");
    } else {
      console.log("-- [Success] LightSwitch = " + data.LightSwitch);
      if (data.LightSwitch == '0') {
        console.log("-- Light is off.");
      } else {
        console.log("-- Light is on.");
      }
    }
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
```

```
};
```

2.2.3 setThingProperties

Sets the value of a property for a specified device.

setThingProperties(params, callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err)	function	The callback function. The callback function must follow standard JavaScript practices. • If the call is successful, err is null. • If the call fails, err contains the error information.

Table 2-4: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the devices belong. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
payload	Object	The property identifiers and the property values to be set. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **View TSL** to view the identifier of each service.

Sample requests

The following example sets the LightSwitch property value of the LightDev2 device to 1. A value of 1 indicates the device is switched on.

```
'use strict';

const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();

const setPropertiesParams = {
  productKey: 'a1ZJTVsqj2y', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
```

```
  payload: {'LightSwitch': '1'} // The property defined in the Product TSL.
};
/* Promise wrapper for setThingProperties. */
function setThingProperties(params) {
  return new Promise((resolve, reject) => {
    iotData.setThingProperties(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}

exports.handler = function (event, context, callback) {
  setThingProperties(setPropertiesParams).then((data) => {
    console.log("-- Set Property Success. Return " + JSON.stringify(data));
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.4 callThingService

Calls a service of a specified device.

callThingService(params, callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err, data)	function	The callback function. The callback function must follow standard JavaScript practices. For more information, see the table Parameter descriptions of callback.

Table 2-5: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the device belongs. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
service	String	The identifier of the service that is called. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **TSL Model** to view the identifier of each service.

Parameter	Type	Description
payload	String|Buffer	The identifiers and values of the input parameters for the specified service. You can click **TSL Model** to view the values of the identifier parameter and other parameters.

Table 2-6: Parameter descriptions of callback

Parameter	Type	Description
err	Error	• If the call is successful, err is null. • If the call fails, err contains the error information.
data	object	The return value of the called service.

Sample requests

The following example calls the turn service for the LightDev2 device.

```
'use strict';

const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();

const callServiceParams = {
  productKey: 'a1ZJTVsqj2y', // Please replace it with your Product Key.
  deviceName: 'LightDev2',   // Please replace it with your Device Name.
  service: 'turn',           // The service defined in the Product TSL.
  payload: {'LightSwitch': 0},
};
/* Promise wrapper for callThingService. */
function callThingService(params) {
  return new Promise((resolve, reject) => {
    iotData.callThingService(params, (err, data) => {
      err ? reject(err) : resolve(data);
    });
  });
}

exports.handler = function (event, context, callback) {
  callThingService(callServiceParams).then((data) => {
    console.log("-- Call service Success. Return " + JSON.stringify(data));
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
```

```
};
```

2.2.5 getThingsWithTags

Queries devices by tags.

Description

If you specify multiple tags, IoT Platform returns a list of devices that match all the specified tags.

getThingsWithTags(params, callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err, data)	function	The callback function. The callback function must follow standard JavaScript practices. For more information, see the table Parameter descriptions of callback.

Table 2-7: Parameter descriptions of params

Parameter	Type	Description
payload	Array	The tag information. It is an array that consists of the {<tag name>:<tag value>} key-value pairs.

Table 2-8: Parameter descriptions of callback

Parameter	Type	Description
err	Error	• If the call is successful, err is null. • If the call fails, err contains the error information.
data	Array	This function returns an array of devices that meet the tag conditions.

Sample requests

The following example queries devices that have the tag location: master bedroom.

```
'use strict';

const leSdk = require('linkedge-core-sdk');
const iotData = new leSdk.IoTData();
```

```
const deviceTags = {
  payload: [{'location': 'master bedroom'}],
};
/* Promise wrapper for getThingsWithTags. */
function getThingsWithTags(params) {
  return new Promise((resolve, reject) => {
    iotData.getThingsWithTags(params, (err, things) => {
      err ? reject(err) : resolve(things);
    });
  });
}

exports.handler = function (event, context, callback) {
  getThingsWithTags(deviceTags).then((things) => {
    console.log(JSON.stringify(things));
    for(var i=0; i<things.length; i++) {
       console.log('-- productKey='+things[i]["productKey"] + ', deviceName='+things[i]["
deviceName"]);
    }
    callback(null);
  }).catch((err) => {
    console.log(err);
    callback(err);
  });
};
```

2.2.6 invokeFunction

Calls a specified function.

Description

You can exchange information between processes or functions by calling functions or publishing messages. The difference is that:

- Calling functions is bidirectional. After the caller sends a message to the receiver, the caller will receive a message returned by the receiver.
- Publishing messages is unidirectional. The sender does not need a response from the receiver.

invokeFunction(params, callback)

Parameter	Type	Description
params	object	The parameter object. For more information, see the table Parameter descriptions of params.
callback(err, data)	function	The callback function. The callback function must follow standard JavaScript practices. For more information, see the table Parameter descriptions of callback.

Table 2-9: Parameter descriptions of params

Parameter	Type	Description
serviceName	String	The service name. It is the name of the service to which the function belongs is in Alibaba Cloud Function Compute.
functionName	String	The function name that is defined in Alibaba Cloud Function Compute.
invocationType	String	The call method. • Sync: synchronous • Async: asynchronous This is a synchronous call by default.
payload	String|Buffer	The input parameter information of the function.

Table 2-10: Parameter descriptions of callback

Parameter	Type	Description
err	Error	• If the call is successful, err is null. • If the call fails, err contains the error information.
data	object	The return value of the called function.

Sample requests

- **Sample code for calling a function**

  In the code, the function with serviceName=EdgeFC and functionName=helloworld is called.

```
'use strict';

const leSdk = require('linkedge-core-sdk');
const fc = new leSdk.FCClient();

module.exports.handler = function (event, context, callback) {
  var ctx = {
    custom: {
      data: 'Custom context from Invoker',
    },
  };
  var invokerContext = Buffer.from(JSON.stringify(ctx)).toString('base64');
  var invokeParams = {
    serviceName: 'EdgeFC',
    functionName: 'helloworld',
    invocationType: 'Sync',
    invokerContext: invokerContext,
    payload: 'String message from Invoker.',
```

```
  };
  fc.invokeFunction(invokeParams, (err, data) => {
    if (err) {
      console.log(err);
    } else {
      console.log(data.payload); // Message from helloworld
    }
    callback(null);
  });
};
```

- **Sample code of a function to be called**

  The following helloworld function code indicates how a called function parses input parameters from the caller and returns the result to the caller.

```
module.exports.handler = function(event, context, callback) {
  console.log(event);    // String message from Invoker.
  console.log(context); // { requestId: '4', invokerContext: {custom: { data: 'Custom
data from Invoker' }}}
  console.log(context.invokerContext.custom.data); // Custom data from Invoker
  console.log('-- hello world');
  callback(null, 'Message from helloworld'); // Return the result to Invoker.
};
```

2.2.7 CredentialProviderChain

Obtains the access credentials to cloud services.

CredentialProviderChain().resolvePromise()

This function returns a Promise object, through which a temporary credential for accessing cloud services is obtained. The credential is updated every 15 minutes by default. Therefore, you can call this function to obtain the updated credential when you want to access a cloud service. The format of the credential is as follows:

Parameter	Type	Description
accessKeyId	String	The AccessKey ID. The AccessKey pair consists of an AccessKey ID and an AccessKey secret. You can use the AccessKey pair to authenticate API requests that you make to Alibaba Cloud.
accessKeySecret	String	The AccessKey secret.
securityToken	String	The STS token.

Sample requests

This topic describes the usage of the **CredentialProviderChain().resolvePromise()** method. The sample code is as follows:

Note:

For detailed description of the sample code for accessing OSS, see #unique_30.

```
'use strict';

var leSdk = require('linkedge-core-sdk');
var credChain= new leSdk.CredentialProviderChain();
var OSS = require('./node_modules/ali-oss');

exports.handler = function (event, context, callback) {
  // Retrieves group role credential.
  credChain.resolvePromise()
    .then((cred) => {
      console.log('Access Key Id: %s, Access Key Secret: %s, Security Token: %s',
        cred.accessKeyId, cred.accessKeySecret, cred.securityToken);

      // Constructs a client to interact with OSS cloud service.
      var client = new OSS({
        accessKeyId: cred.accessKeyId,
        accessKeySecret: cred.accessKeySecret,
        stsToken: cred.securityToken,
        region: 'oss-cn-shanghai',
        bucket: 'le-fc-bucket',
      });

      /* Upload local file to cloud. */
      return client.put('fileFromEdge.txt', "/linkedge/run/localFile.txt");
    })
    .then((result) => {
      console.log(result);
      callback(null);
    })
    .catch((err) => {
      console.log(err);
      callback(err);
    });
};
```

2.3 Python

2.3.1 publish

Publishes a message to a specified topic.

publish(params)

Table 2-11: Request parameters

Parameter	Type	Description
params	dict	The parameter object. For more information, see the table Parameter descriptions of params.

Table 2-12: Parameter descriptions of params

Parameter	Type	Description
topic	String	The topic that receives the message.
payload	String	The message payload.

Table 2-13: Response parameters

Parameter	Type	Description
return	dict	The call is successful, a success message is returned. Otherwise, an error message is returned. **Note:** The success message only indicates that the call was successful, but does not indicate that the message was sent.

Sample requests

In the following example, each time an external event is triggered, a message is sent to the `/topic/hello` topic.

```
# -*- coding: utf-8 -*-
import lecoresdk

lesdk = lecoresdk.IoTData()

def handler(event, context):
  pub_params = {"topic": "/topic/hello",
          "payload": "Hello World"}
  lesdk.publish(pub_params)
  print("Publish message to /topic/hello.")
  return 'Hello world'
```

2.3.2 getThingProperties

Obtains the value of a property for a specified device.

getThingProperties(params)

Table 2-14: Request parameters

Parameter	Type	Description
params	dict	The request parameter object. For more information, see the table Parameter descriptions of params.

Table 2-15: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the device belongs. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
payload	List	The array of identifiers for the device properties that you want to obtain. A successful call will return the current corresponding value of the property. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **TSL Model** to view the identifier of each service.

Table 2-16: Response parameters

Parameter	Type	Description
return	dict	If the call is successful, the property values of the device are returned. Otherwise, the error message reported by the device driver is returned.

Sample requests

The following example obtains the LightSwitch property value of the LightDev2 device (whether the device is switched on).

```
# -*- coding: utf-8 -*-
import lecoresdk

lesdk = lecoresdk.IoTData()

def handler(event, context):
  get_params = {"productKey": "a1ZJTVsqj2y", # Please replace it with your Product Key.
                "deviceName": "LightDev2",   # Please replace it with your Device Name.
                "payload": ["LightSwitch"]}  # The property defined in the Product TSL.
  res = lesdk.getThingProperties(get_params)
  print(res)
  if 'LightSwitch' in res.keys():
    print("-- [Success] LightSwitch = %s" % res['LightSwitch'])
    if res['LightSwitch'] == '0':
      print("-- Light is off.")
    else:
      print("-- Light is on.")
  else:
    print("-- [Warn] No property LightSwitch return.");
```

```
return 'OK'
```

2.3.3 setThingProperties

Sets the value of a property for a specified device.

setThingProperties(params)

Table 2-17: Request parameters

Parameter	Type	Description
params	dict	The request parameter object. For more information, see the table Parameter descriptions of params.

Table 2-18: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the device belongs. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
payload	dict	The property identifiers and the property values to be set. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **TSL Model** to view the identifier of each service.

Table 2-19: Response parameters

Parameter	Type	Description
return	dict	If the call is successful, true is returned. Otherwise, the error message reported by the driver is returned.

Sample requests

The following example sets the LightSwitch property value of the LightDev device to 0. The value 0 indicates the device is switched off.

```
# -*- coding: utf-8 -*-
import lecoresdk
```

```
lesdk = lecoresdk.IoTData()

def handler(event, context):
  set_params = {"productKey": "a1ZJTVs****", # Please replace it with your Product Key.
              "deviceName": "LightDev",    # Please replace it with your Device Name.
              "payload": {"LightSwitch":0}}# The property defined in the Product TSL.
  res = lesdk.setThingProperties(set_params)
  print(res)
  return 'OK'
```

2.3.4 callThingService

Calls a service of a specified device.

callThingService(params)

Table 2-20: Request parameters

Parameter	Type	Description
params	dict	The request parameter object. For more information, see the table Parameter descriptions of params.

Table 2-21: Parameter descriptions of params

Parameter	Type	Description
productKey	String	The key of the product to which the device belongs. It is a unique identifier issued by IoT Platform for the product.
deviceName	String	The name of the device. The device name is generated when the device is created.
service	String	The identifier of the service that is called. Log on to the IoT Platform console. On the **Define Feature** tab of the product details page, click **TSL Model** to view the identifier of each service.
payload	String|Bytes	The identifiers and values of the input parameters for the specified service. You can click **TSL Model** to view the values of the identifier parameter and other parameters.

Table 2-22: Response parameters

Parameter	Type	Description
return	dict	If the call is successful, the service value is returned. Otherwise, the error message reported by the driver is returned.

Sample requests

The following example calls the turn service for device lightdev2.

```
# -*- coding: utf-8 -*-
import lecoresdk

lesdk = lecoresdk.IoTData()

def handler(event, context):
  get_params = {"productKey": "a1ZJTVs****", # Please replace it with your Product Key.
              "deviceName": "LightDev2",   # Please replace it with your Device Name.
              "service":"turn",          # The service defined in the Product TSL.
              "payload": {"LightSwitch": 1}}
  res = lesdk.callThingService(get_params)
  print(res)
  return 'OK'
```

2.3.5 getThingsWithTags

You can call this operation to query device groups by tags.

Limits

If you specify multiple tags, this operation returns a list of devices that match all the specified tags.

getThingsWithTags(params)

Table 2-23: Request parameters

Parameter	Type	Description
params	dict	The parameter object. For more information, see the table Parameter descriptions of params.

Table 2-24: Parameter descriptions of params

Parameter	Type	Description
payload	List	The array of tag information. The array consists of {<tag name>:<tag value>} key-value pairs.

Table 2-25: Response parameters

Parameter	Type	Description
return	dict	If the call is successful, the property values of the device are returned. Otherwise, the error message reported by the device driver is returned.

Sample requests

The following example queries devices that have the tag location: master bedroom.

```
# -*- coding: utf-8 -*-
import lecoresdk

lesdk = lecoresdk.IoTData()

def handler(event, context):
  get_params = {"payload": [{"location":"master bedroom"}]}
  res = lesdk.getThingsWithTags(get_params)
  print(res)
  for device in res:
    print("device_%s: PK=%s DN=%s" % (res.index(device) + 1, device['productKey'], device['
deviceName']))
  return 'OK'
```

2.3.6 invokeFunction

Calls a specified function.

Description

You can exchange information between processes or functions by calling functions or publishing messages. The differences between calling functions and publishing messages are:

- The exchange of messages in a function call is bidirectional. After the caller sends a message to the receiver, the caller will receive the message returned by the receiver.
- Publishing messages is unidirectional. The sender does not need a response from the receiver.

invokeFunction(params)

Table 2-26: Request parameters

Parameter	Type	Description
params	dict	The parameter object. For more information, see the table Parameter descriptions of params.

Table 2-27: Parameter descriptions of params

Parameter	Type	Description
serviceName	String	The service name. It is the name of the service to which the function belongs in Alibaba Cloud Function Compute.
functionName	String	The function name that is defined in Alibaba Cloud Function Compute.
invocationType	String	The type of the function call. • Sync: synchronous • Async: asynchronous This is a synchronous call by default.
payload	String|Bytes	The input parameter information of the function.

Table 2-28: Response parameters

Parameter	Type	Description
return	dict	The return value of the called function.

Sample requests

- **Sample code for calling a function**

  In the code, the function with serviceName=EdgeFC and functionName=helloworld is called.

```
# -*- coding: utf-8 -*-
import lecoresdk
edgefc = lecoresdk.Client()

def handler(event, context):
  context = {"custom": {"data": "customData"}}
  invokeParams = {
    "serviceName": 'EdgeFC',
    "functionName": 'helloworld',
    "invocationType": 'Sync',
    "invokerContext": context,
    "payload": 'String message from Python Invoker.'
  };
  res = edgefc.invoke_function(invokeParams)
  print(res)
```

```
  return 'OK'
```

- **Sample code of a function to be called**

  The following helloworld function code indicates how a called function parses input parameters from the caller and returns the result to the caller.

```
# -*- coding: utf-8 -*-
import logging
import lecoresdk

def handler(event, context):
  logging.debug(event)
  logging.debug(context)
  return 'hello world'
```

2.3.7 CredentialProviderChain

Obtains the access credential for cloud services.

CredentialProviderChain().get_credential()

This operation returns the temporary credential for accessing a cloud service. The credential is updated every 15 minutes. Therefore, you can call this operation to obtain the updated credential when you want to access a cloud service. The format of the credential is as follows:

Parameter	Type	Description
accessKeyId	String	The AccessKey ID. The AccessKey pair consists of an AccessKey ID and an AccessKey secret. You can use the AccessKey pair to authenticate API requests that you make to Alibaba Cloud.
accessKeySecret	String	The AccessKey secret.
securityToken	String	The security token.

Sample requests

This example describes how to use the **CredentialProviderChain().resolvePromise()** method. The sample code is as follows:

Note:

The sample code depends on the third-party library oss2. Download the putOssFilePy-code.zip file to see the complete code.

```
# -*- coding: utf-8 -*-
import oss2
import lecoresdk
```

```
def handler(event, context):
  cred = lecoresdk.CredentialProviderChain().get_credential()
  auth = oss2.StsAuth(cred['accessKeyId'], cred['accessKeySecret'], cred['securityToken'])
  bucket = oss2.Bucket(auth, 'http://oss-cn-shanghai.aliyuncs.com', 'le-fc-bucket')
  bucket.put_object('fileFromEdgePy.txt', 'Content of object')
  print("-- Put fileFromEdgePy.txt to OSS.")
  return 'OK'
```

3 Edge HTTP API

3.1 Overview

Link IoT Edge allows you to manage the devices that are connected to the gateway by using the edge API. For example, you can query devices, set device properties, and subscribe to device events.

The port number that provides external services for the edge API is `9999`. All API operations support one-way HTTPS authentication.

Authentication mechanism

Edge API supports authentication based on cookies. To enable cookie authentication, you must call the CreateAuthCookie API operation to obtain the authentication cookie so that you can call other API operations. The **CreateAuthCookie** API operation uses the Basic schema for authentication. Developers must know the username and password of the gateway, and the user account must have the permissions to call the corresponding API operations.

Note:

The default username of the gateway is admin and the password is admin1234. You can log on to the gateway console to modify the username, password, and API access permissions. For information about how to access the gateway console, see #unique_48.

List of API operations by function

- Authentication

API operation	Description
CreateAuthCookie	Creates an authentication cookie.
DeleteAuthCookie	Deletes an authentication cookie.

- Device management

API operation	Description
ListThings	Queries devices and their status.
SetThingProperties	Sets device properties.
GetThingProperties	Obtains device properties.

API operation	Description
CallThingServices	Invokes a service on a device.
BulkActions	Performs batch API operations on devices.

- Function Compute

API operation	Description
InvokeFunction	Calls a specified function.

3.2 Status codes

The following table lists the status codes of the edge API.

Code	Response message	Description
200	Success	The call was successful.
201	Created	The request was successful and the server created new resources.
302	Move temporarily	The URL was redirected.
400	Bad Request	The error code may be reported due to the following two reasons: • The semantic of the request was invalid. The current request cannot be understood by the server. The client will not send the same request unless the request is modified. • The request parameters contained an error.
401	Unauthorized	The current request requires user authentication. This means that the cookie has expired and you need to call the **CreateAuthCookie**API operation to obtain the updated cookie. Otherwise, you cannot call other API operations.
403	Forbidden	The server understood the request. However , the server refused to execute the request . You must verify whether the current user has the permissions to perform the API operation.

Code	Response message	Description
404	Not Found	The request failed because no resources were found on the server. You must check whether the parameters in the request are correct.
405	Method Not Allowed	The HTTP method specified in the request cannot be used to request the specified resource. For example, the POST method is not supported.
421	Too Many Connections	The number of connections established between the server and the IP address where the current client is located has reached the upper limit. **Note:** The IP address is the address of the client that is seen on the server, such as the address of the client gateway or a proxy server used by the client.
500	Internal Server Error	The server encountered an unexpected error and was unable to process the request. This error code is usually returned when an error occurs in the source code of the server.
503	Service Unavailable	The server is currently unable to process requests due to temporary server maintenance or server overload.

3.3 Authentication

3.3.1 CreateAuthCookie

Creates a new authentication cookie.

Request syntax

```
POST /2019-09-30/auth/cookies
```

```
Authorization: Authorization
```

Request parameters

Parameter	Type	Required	Description
Authorization	String	Yes	The Basic authentication information. The format is `Basic base64(username:password)`. For example, the value of the parameter can be `Basic Y2h5aW5n********NTY=`.

Response syntax

```
HTTP/1.1 StatusCode
Set-Cookie: Cookie
Content-Type: application/json

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, an error code is returned. For information about error codes, see Status codes.
Cookie	String	The authentication cookie that used to call API operations.
Payload	JSON	The error message.

The format of the returned Payload is as follows:

```
{
  "Code": 201,
  "Message": "sucess|reason for failure"
}
```

Example

```
$ curl -i -c token.cookie -u admin:admin1234 -k -X POST https://127.0.0.1:9999/2019-09-30/auth/cookies

HTTP/1.1 201 Created
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 07:51:57 GMT
```

```
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
Set-Cookie: token=57e22d4f9fb237fcf5c0b59abf621ddeecde1ef740a84fbeb78540******
bbe4; Max-Age=3600; Path=/

{"Code":201,"Message":"success"}
```

3.3.2 DeleteAuthCookie

Deletes an authentication cookie.

Request syntax

```
DELETE /2019-09-30/auth/cookies/ Token
Authorization: Authorization
```

Request parameters

Parameter	Type	Required	Description
Token	String	Yes	The token of the authentication cookie.
Authorization	String	Yes	The Basic authorization information. The format is `Basic base64(username:password)`. For example, the value of the parameter can be `Basic Y2h5aW5n********NTY=`.

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: application/json

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, an error code is returned. For information about error codes, see Status codes.
Payload	JSON	The returned message.

The format of the returned Payload is as follows:

```
{
  "Code": 200,
  "Message": "sucess|reason for failure"
}
```

Example

```
$ curl -i -u admin:admin1234 -k -X DELETE https://127.0.0.1:9999/2019-09-30/auth/cookies/57e22d4f9fb237fcf5c0b59abf621ddeecde1ef740a84fbeb78540******bbe4

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 07:57:23 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"Code":200,"Message":"success"}
```

3.4 Device management

3.4.1 ListThings

Queries devices and their status.

Request syntax

```
GET /2019-09-30/things HTTP/1.1
Cookie: Cookie
```

Request parameters

Parameter	Type	Required	Description
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookieAPI operation is called.

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: application/json
```

```
Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The status code. If the request is successful, 200 is returned. If the request fails, an error code is returned. For information about error codes, see Status codes.
Payload	JSON	The device information that you have obtained.

The format of the returned Payload is as follows:

```
{
   "Code": number,
   "Message": "success|reason for failure",
   "Data": {
    "Things": [{
      "ProductName": "string",
      "ProductKey": "string",
      "DeviceName": "string",
      "DriverId": "string",
      "DriverName": "string",
      "Tags": [{"string": "string"}],
      "Status": "Inactivated|Failed|Online|Offline",
      "Connected": true|false,
      "ConnectTime": "string"
    }]
   }
 }
```

Example

```
$ curl -i -b token.cookie -k https://127.0.0.1:9999/2019-09-30/things

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 08:23:21 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"Data":{"Things":[{"DriverName":"e0bb1b48964e4519b4a52f9b719e1***","ConnectTime
":"1572510191943","DeviceName":"GjCb9LKXgcKXeluG****","DriverId":"","ProductKey":"","
```

```
Connected":true,"ProductName":"","Tags":[],"Status":"Online"}]},"Code":200,"Message":"
success"}
```

3.4.2 GetThingProperties

Obtains the value of a property for a specified device.

Request syntax

```
GET /2019-09-30/things/ProductKey/DeviceName/services HTTP/1.1
 identifiers= Identifier1&identifiers= Identifier2&... HTTP/1.1
Cookie: Cookie
```

Request Parameters

Parameter	Type	Required	Description
ProductKey	String	Yes	The unique identifier of product to which the device belongs. You can obtain the value of this parameter from the IoT Platform console.
DeviceName	String	Yes	The name of the device. You can obtain the value of this parameter from the IoT Platform console.
Identifiers	String	Yes	The identifiers for the properties of the device. You can query up to 100 property values.
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookie API operation is called.

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: application/json

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.

Parameter	Type	Description
Payload	JSON	The device properties that have been obtained.

The format of the returned Payload is as follows:

```
{
  "Code": 200,
  "Message": "sucess|reason for failure",
  "Data": {
    "Properties": {
      "Identifier1": "value1",
      "Identifier2": "value2",
      "Identifier3": "value3"
    },
    "Timestamp": 1568262117344
  }
}
```

Example

```
$ curl -b token.cookie -k https://127.0.0.1:9999/2019-09-30/things/a1WabAEC***/
N0hB9tiVWWZFMpALK***/properties?identifiers=LightSwitch

{"Data":{"Timestamp":1572512318420,"Properties":{"LightSwitch":1}},"Code":200,"
Message":"success"}
```

3.4.3 SetThingProperties

Sets properties of a specified device.

Request syntax

```
POST /2019-09-30/things/ProductKey/DeviceName/services HTTP/1.1
Cookie: Cookie

Payload
```

Request Parameters

Parameter	Type	Required	Description
ProductKey	String	Yes	The unique identifier of product to which the device belongs. You can obtain the value of this parameter from the IoT Platform console.
DeviceName	String	Yes	The name of the device. You can obtain the value of this parameter from the IoT Platform console.

Parameter	Type	Required	Description
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookie API operation is called.
Payload	JSON	Yes	Sets device properties. For more information, see the request Payload format.

The format of the request Payload is as follows:

```
{
  "Properties": {
    "Identifier1": value1,
    "Identifier2": value2,
    "Identifier3": value3
      ...
  }
}
```

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: application/json

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.
Payload	JSON	Sets the content returned by the interface.

The format of the returned Payload is as follows:

```
{
  "Code": 200,
  "Message": "sucess|reason for failure",
  "Data": {
    "Data": string|boolean|number|array|object, // A optional data that returned from the underlying set
    "Timestamp": 1568262117344
  }
```

```
}
```

Example

```
$ curl -b token.cookie -d '{"Properties":{"LightSwitch":0}}' -k -X POST https://127.0.0.1:
9999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZFMpALK***/properties

{"Data":{"Data":[],"Timestamp":1572512468781},"Code":200,"Message":"success"}
```

3.4.4 CallThingServices

Calls services on the device.

Request syntax

```
POST /2019-09-30/things/ ProductKey/DeviceName/services HTTP/1.1
Cookie: Cookie

Payload
```

Request Parameters

Parameter	Type	Required	Description
ProductKey	String	Yes	The unique identifier of product to which the device belongs. You can obtain the value of this parameter from the IoT Platform console.
DeviceName	String	Yes	The name of the device. You can obtain the value of this parameter from the IoT Platform console.
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookie API operation is called.
Payload	JSON	Yes	The name and the parameters of the called service. The name and the parameters of the service must be the same as those that are defined in the product TSL of the device in IoT Platform. For more information, see the request Payload format.

The format of the request Payload is as follows:

```
{
  "Services": [
    {
```

```
    "Name": "string",
    "Args": args // Optional arguments for the service.
   }
  ]
}
```

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: application/json

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The status code of the operation. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.
Payload	JSON	The response of the service call.

The format of the returned Payload is as follows:

```
{
  "Code": 200,
  "Message": "sucess|reason for failure",
  "Data": {
    "Services": [
      {
        "Name": "string",
        "Returns": {
          "Message": "success|reason for failure",
          "Data": string|boolean|number|array|object, // A optional data that returned from
the underlying call services call.
        }
      }
    ],
      "Timestamp": 1568262117344
  }
}
```

Example

```
$ curl -i -b token.cookie -d '{"Services":[{"Name":"setColor","Args": {"color":"red"}}]}' -k -
X POST https://127.0.0.1:9999/2019-09-30/things/a1WabAEC***/N0hB9tiVWWZFMpALK
***/services

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:17:47 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive
```

{"Data":{"Services":[{"Name":"setColor","Returns":{"Message":"success","Data":[]}}],"
Timestamp":1572520667899},"Code":200,"Message":"success"}

3.4.5 BulkActions

Manage devices in batches.

Request syntax

```
POST /2019-09-30/things/bulk HTTP/1.1
Cookie: Cookie

Payload
```

Request Parameters

Parameter	Type	Required	Description
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookie API operation is called.
Payload	JSON	Yes	The JSON object that contains the actions (BulkAction), the target devices (Things), and related parameters. For more information, see the Payload format.

The format of the request Payload is as follows:

```
{
  "BulkAction": "string"
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      ... // Parameters for BulkAction
    }
  ]
```

```
}
```

Table 3-1: Payload parameters

Parameter	Type	Required	Description
BulkAction	String	Yes	The action that you want to perform. Valid values: • SetProperites: sets device properties in batches. • GetProperites: obtains device properties in batches. • CallServices: calls device services in batches. • Watch: obtains a specified set of events from the device.
Things	Array	Yes	The JSON array that contains the object and operation parameters. For more information about the format, see the Things parameter defined in the GetProperties, SetProperties, CallServices, and Watch sections.

Response syntax

```
HTTP/1.1 StatusCode
Content-Type: ContentType

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.
ContentType	String	The data type of the returned message (Payload). The data type depends on the specified action (BulkAction).
Payload	JSON	The response of the batch operation.

```
{
```

```
  "Code": number,
  "Message": "success|reason for failure"
  "Data": {}
}
```

Table 3-2: Payload parameters

Parameter	Type	Description
Code	Number	The status code of the operation. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.
Message	String	If the request is successful, success is returned. If the call fails, the reason for the failure is returned.
Data	Object	The returned result of a batch operation. For more information about the format, see the Data parameter defined in the GetProperties, SetProperties, CallServices, and Watch sections.

GetProperties

- The format of the request Payload

```
{
  "BulkAction": "GetProperties",
  "Things": [
    {
      "ProductKey": "string",
      "DeviceName": "string",
      "Identifiers": [ // Property Identifiers
        "string",
        "string",
        "string",
        ...
      ]
    }
  ]
}
```

- The format of the returned Payload

  Content-Type：application/json

```
{
  "Code": number,
  "Message": "success|reason for failure",
  "Data": {
    "Results": [
      {
        "ProductKey": "string",
```

```
      "DeviceName": "string",
      "Returns": {
       "Message": "success|reason for failure",
       "Data": { // The properties returned from the underlying get properties call.
         "Identifier1": value1,
         "Identifier2": value2,
         "Identifier3": value3
           ...
       }
      },
    }
   ],
   "Timestamp": 1568262117344
 }
}
```

- Example

```
$ curl -i -b token.cookie -d '{"BulkAction":"GetProperties","Things":[{"ProductKey":"
a1WabAEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Identifiers":["LightSwitch"]}]}'
 -k -X POST https://127.0.0.1:9999//2019-09-30/things/bulk

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:30:40 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"Data":{"Results":[{"Returns":{"Message":"success","Data":{"LightSwitch":1}},"
DeviceName":"N0hB9tiVWWZFMpALK***","ProductKey":"a1WabAEC***"}],"Timestamp":
1572521440288},"Code":200,"Message":"success"}
```

SetProperties

- The format of the request Payload

```
{
 "BulkAction": "SetProperties",
 "Things": [
  {
   "ProductKey": "string",
   "DeviceName": "string",
   "Properties": {
    "Identifier1": value1,
    "Identifier2": value2,
    "Identifier3": value3
  ...
    }
  }
 ]
}
```

- The format of the returned Payload

  Content-Type：application/json

```
{
 "Code": number,
 "Message": "success|reason for failure",
 "Data": {
```

```
   "Results": [
    {
     "ProductKey": "string",
     "DeviceName": "string",
     "Returns": {
      "Message": "success|reason for failure",
      "Data": string|boolean|number|array|object // An optional data that returned
from the underlying set properties call.
     },
    }
   ],
   "Timestamp": 1568262117344
 }
}
```

- Example

```
$ curl -i -b token.cookie -d '{"BulkAction":"SetProperties","Things":[{"ProductKey":"
a1WabAEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Properties":{"LightSwitch":1
}}]}' -k -X POST https://127.0.0.1:9999//2019-09-30/things/bulk

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:46:53 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"Data":{"Results":[{"Returns":{"Message":"success","Data":[]},"DeviceName":"
N0hB9tiVWWZFMpALK***","ProductKey":"a1WabAEC***"}],"Timestamp":1572522413633
},"Code":200,"Message":"success"}
```

CallServices

- The format of the request Payload

```
{
 "BulkAction": "CallServices",
 "Things": [
  {
   "ProductKey": "string",
   "DeviceName": "string",
   "Services": [
    {
     "Name": "string",
     "Args": args // Optional arguments for the service.
    }
   ]
  }
 ]
}
```

- The format of the returned Payload

  Content-Type：application/json

```
{
 "Code": number,
 "Message": "success|reason for failure",
 "Data": {
  "Results": [
```

```
    {
     "ProductKey": "string",
     "DeviceName": "string",
     "Services": [
      {
       "Name": "string",
       "Returns": {
        "Message": "success|reason for failure",
        "Data": string|boolean|number|array|object // An optional data that returned
from the underlying call services call.
       }
      }
     ],
    }
   ],
   "Timestamp": 1568262117344
 }
}
```

- Example

```
$ curl -i -b token.cookie -d '{"BulkAction":"CallServices","Things":[{"ProductKey":"
a1WabAEC***","DeviceName":"N0hB9tiVWWZFMpALK***","Services":[{"Name":"setColor
","Args":{"color":"red"}}]}]}' -k -X POST https://127.0.0.1:9999//2019-09-30/things/bulk

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Thu, 31 Oct 2019 11:52:13 GMT
Content-Type: application/json; charset=utf-8
Transfer-Encoding: chunked
Connection: keep-alive

{"Data":{"Results":[{"Services":[{"Name":"setColor","Returns":{"Message":"success","
Data":[]}}],"DeviceName":"N0hB9tiVWWZFMpALK***","ProductKey":"a1WabAEC***"}],"
Timestamp":1572522733310},"Code":200,"Message":"success"}
```

Watch

- The format of the request Payload

```
{
 "BulkAction": "Watch",
 "Things": [
  {
   "ProductKey": "string",
   "DeviceName": "string",
   "Watches": "all|properties|events" // default is "all"
  }
 ]
}
```

- The format of the returned Payload

  Content-Type：text/plain

  - Regular messages

    ```
    {
     "Code": number,
     "Message": "success|reason for failure",
     "Data": {
    ```

```
    "ProductKey": "string",
    "DeviceName": "string",
    "Event": "string",
    "Args": {}, // Optional arguments along with the event.
    "Timestamp": 1568010749340
  }
}
```

- Heartbeat messages

```
{
  "Code": 200,
  "Message": "heartbeat",
  "Data": {
    "Timestamp": 1568010749340
  }
}
```

- Example

```
$ curl -b token.cookie -d '{"BulkAction":"Watch","Things":[]}' -k -X POST https://127.0.0.
1:9999//2019-09-30/things/bulk

{"Data":{"Timestamp":1572510704112,"Event":"propertiesChanged","ProductKey":"
a1t9b6Nn***","Args":{"MeasuredIlluminance":{"time":1572510704110,"value":400}},"
DeviceName":"GjCb9LKXgcKXeluGJ***"},"Code":200,"Message":"success"}
{"Data":{"Timestamp":1572510706116,"Event":"propertiesChanged","ProductKey":"
a1t9b6Nn***","Args":{"MeasuredIlluminance":{"time":1572510706114,"value":300}},"
DeviceName":"GjCb9LKXgcKXeluGJ***"},"Code":200,"Message":"success"}
{"Data":{"Timestamp":1572510706117},"Code":200,"Message":"heartbeat"}
{"Data":{"Timestamp":1572510708119,"Event":"propertiesChanged","ProductKey":"
a1t9b6Nn***","Args":{"MeasuredIlluminance":{"time":1572510708118,"value":200}},"
DeviceName":"GjCb9LKXgcKXeluGJ***"},"Code":200,"Message":"success"}
```

3.5 Function Compute

3.5.1 InvokeFunction

Calls a specified function.

Request syntax

```
POST /2019-09-30/functions/Function/invocations HTTP/1.1
Cookie: Cookie
```

```
Payload
```

Request Parameters

Parameter	Type	Required	Description
Function	String	Yes	The information about the function to be called. Two ARN formats are available: • Complete ARN format: The format is acs:fc:your_Region:your_AccountId:service:your_Serv • Partial ARN format: The format is service:your_ServiceName:function:your_FunctionNa **Notice:** The ARN format must match the following regular expression pattern: (^acs:fc:([a-z]{2}-[a-z]+(-\d)?)? :(\d{16 })? :)? service:([a-zA-Z][\w-]{0,127}): function:([a-zA-Z][\w-]{0,127})$
Cookie	String	Yes	The authentication cookie generated when the CreateAuthCookie API operation is called.
Payload	JSON	Yes	The Payload content is passed to the function without modifications.

Response syntax

```
HTTP/1.1 StatusCode
X-Fc-Error-Type: ErrorType

Payload
```

Response Parameters

Parameter	Type	Description
StatusCode	Number	The HTTP status code. If the request is successful, 200 is returned. If the request fails, other status codes are returned. For information, see Status codes.

Parameter	Type	Description
ErrorType	String	The error type. This parameter appears when a function encounters an error. The parameter has the following two values: • Handled: The error is reported by the function • Unhandled: The error is detected and reported by Function Compute. For example, the function execution has timed out or the memory is insufficient.
Payload	JSON	The content returned by the function or the error message that is in the following Payload format.

The format of the returned Payload is as follows:

```
{
   "Message": "string",
   "StackTrace": []
}
```

Example

```
$ curl -i -b token.cookie -k -X POST https://127.0.0.1:9999/2019-09-30/functions/service
:helloworld:function:helloworld/invocations

HTTP/1.1 200 OK
Server: openresty/1.13.6.2
Date: Tue, 19 Nov 2019 09:25:42 GMT
Content-Type: text/html; charset=UTF-8
Transfer-Encoding: chunked
Connection: keep-alive

HelloWorld
```