RT-THREAD ONENET User Manual

RT-THREAD Documentation Center

WWW.RT-THREAD.ORG

Friday 28th September, 2018

Versions and Revisions

Date	Version	Author	Note
2018-07-17	v0.1.0	zylx	initial version
2018-07-27	v0.1.1	zylx	Add mqtt upload function

Table of contents

Chapter 1

OneNET Software Package Introduction

OneNET platform is an ecological platform built by China Mobile based on the Internet of Things industry, with high concurrent availability and multi-protocol access.

The OneNET platform also provides the following features:

All-round support to accelerate the development of user products.

OneNET platform is an ecological environment built based on the characteristics of the Internet of Things industry, which can adapt to various network environments and

Protocol type, currently supported protocols include LWM2M (NB-IOT), EDP, MQTT, HTTP, MODBUS,

JT\T808, TCP transparent transmission, RGMP, etc. Users can choose different access protocols according to different application scenarios.

This component package is an adaptation of the RT-Thread system for the OneNET platform connection.

The device can be easily connected to the OneNet platform on RT-Thread to complete data sending, receiving, device registration and

Control and other functions.

1.1 File Directory Structure

```
OneNET
ÿ README.md ÿ                          // Software package instructions
SConscript ÿÿÿÿdocs                    //RT-Thread default build script

ÿ ÿÿÿÿfigures ÿ ÿ api.md ÿ ÿ           // Document using images
introduction.md ÿ ÿ                    // API usage instructions
principle.md ÿ ÿ README.md ÿ ÿ samples.md   // Software package details
ÿ ÿ user-guide.md ÿ ÿ port.md ÿ         // Implementation principle
ÿÿÿÿversion. md ÿÿÿÿports ÿ            // Document structure description
ÿÿÿÿrt_ota_key_port.c                  // Package Example
                                       // Instructions for use
                                       // Transplantation documentation
                                       // Version
                                       // Migrate files
                                       // Migrate file template
```

ÿÿÿÿsamples ÿ	// Sample code //
ÿÿÿÿonenet_sample.c ÿÿÿÿinc ÿÿÿÿsrc	Software package application sample code // Header file // Source file

1.2 Features of OneNET Software Package

The features of the RT-Thread OneNET software package are as follows:

Reconnect after disconnection

The RT-Thread OneNET software package implements a disconnection reconnection mechanism. When the connection is disconnected due to network disconnection or network instability, the login status will be maintained, the connection will be reconnected, and the OneNET platform will be automatically logged in. This improves the reliability of the connection and increases the ease of use of the software package.

Automatic Registration

The RT-Thread OneNET software package implements the automatic device registration function. There is no need to manually create devices one by one on the web page and enter the device name and authentication information. When the device registration function is turned on, when the device logs in to the OneNET platform for the first time, it will automatically call the registration function to register the device with the OneNET platform and save the returned device information for the next login.

Custom response function

The RT-Thread OneNET software package provides a command response callback function. When the OneNET platform issues a command, RT-Thread will automatically call the command response callback function. After the user processes the command, the response content to be sent is returned, and RT-Thread will automatically send the response back to the OneNET platform.

Custom **topic** and callback function

In addition to responding to commands issued by the OneNET official topic, the RT-Thread OneNET software package can also subscribe to user-defined topics and set a command processing callback function for each topic, making it easier for users to develop custom functions.

Uploading binary data

In addition to uploading numbers and strings, the RT-Thread OneNET software package also supports uploading binary files. When the RT-Thread file system is enabled, files in the file system can be directly uploaded to the cloud in binary format.

chapter **2**

OneNET Sample Application

2.1 Preparation

2.1.1 Register an account on **OneNET** Cloud

Before connecting your device to the OneNET cloud, you need to register a user account on the platform. The OneNET cloud platform address is: https://open.iot.10086.cn

Create a product

After successfully registering and logging in to your account, click Developer Center to enter the Developer Center interface;

Click Create Product, enter the basic parameters of the product, and select MQTT protocol for the device access protocol at the bottom of the page , as shown in the following figure:

Figure **2.1:** *onenet*

Figure **2.2:** *onenet_create_product*

After the product is successfully created, you can view the basic product information (such as product ID, access agreement, creation time, product API key, etc., which will be useful later) in the product overview on the left side of the developer center.

Access device

In the device management on the left side of the developer center, click the Add Device button to add a device. We fill in test1 as the device name. The authentication information is used to distinguish each different device. If multiple devices are created, make sure that the authentication information of each device is different. We fill in 201807171718 here. After filling in, click Connect Device

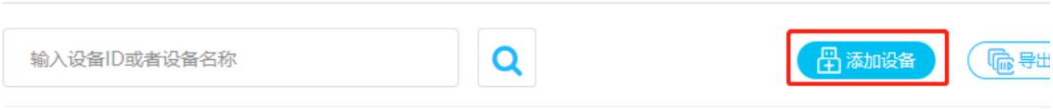

Figure **2.3:** *onenet_add_device*

Figure **2.4:** *onenet_create_device*

Add **API key**

After connecting to the device, you can see that there is one more device on the device management interface. There are some operation functions on the right side of the device.

button, click the View Details button

Figure **2.5:** *onenet_info*

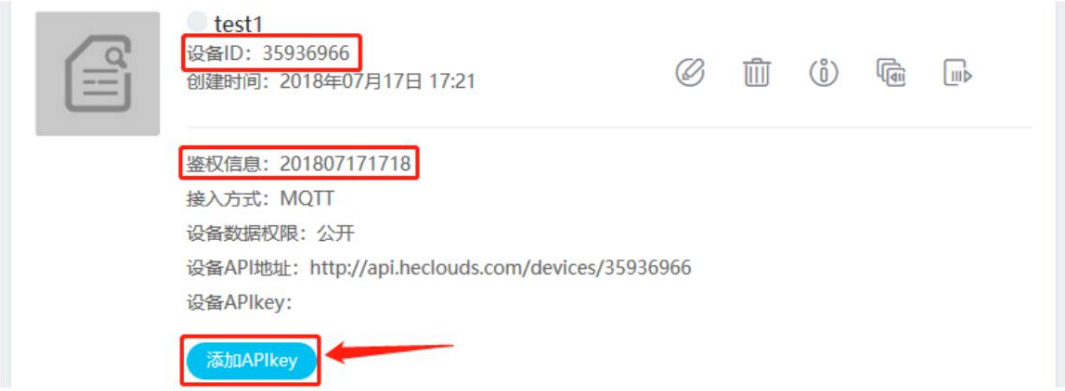

Figure **2.6:** *onenet_add_apikey*

The relevant information of this device is displayed, such as: device ID, authentication information, device API key. This information needs to be recorded and will be used in ENV configuration.

Click the button to add API key. The name of API key is usually related to the device. Here we fill in test_APIKey. Associate the device with the device test1 we just created.

Figure **2.7:** *onenet7*

Enable the **onenet** software package

Open the env tool and enter menuconfig and follow the path below to enable the onenet software package

```
RT-Thread online packages
    IoT - internet of things --->
        IoT Cloud --->
            [*] OneNET: China Mobile OneNet cloud SDK for RT-Thread
```

Enter the configuration menu of the onenet software package and configure as shown below. The information inside depends on your product and device. Fill in the actual situation

```
--- OneNET: China Mobile OneNet cloud SDK for RT-Thread
        [ ] Enable OneNET sample
        [*] Enable support MQTT protocol
        [ ] Enable OneNET automatic register device (NEW) (35936966) device
        id (201807171718) auth info

        (H3ak5Bbl0NxpW3QVVe33InnPxOg=) api key (156418)
        product id
        (dVZ=ZjVJvGjXIUDsbropzg1a8Dw=) master/product apikey (NEW) version (latest)
                --->
```

Enable OneNET sample : Enable OneNET sample code

Enable support MQTT protocol : Enable MQTT protocol connection OneNET support

Enable OneNET automatic register device : Enable OneNET automatic registration device function

device id : The device ID obtained when configuring the cloud to create a device

auth info : User-defined authentication information when configuring cloud-based product creation (unique for each device of each product)

api key : The API key obtained when configuring the cloud to create a device

product id : The product ID obtained when configuring the cloud to create a product

master/product apikey : configure the product APIKey obtained when creating a product on the cloud

2.2 Example File Introduction

After using ENV to generate the project, we can see the onenet_sample.c file in the onenet directory of the project. This file is a sample display of the **OneNET** software package, mainly showing how users can use **the OneNET** software package to upload data and receive commands.

2.3 Run the example

Before using the OneNET software package, you must first call the onenet_mqtt_init command to initialize it. After initialization is completed, the device will automatically connect to the OneNET platform.

```
msh />onenet_mqtt_init [D/
ONENET] (mqtt_connect_callback:85) Enter mqtt_connect_callback!
[D/[MQTT] ] ipv4 address port: 6002 [D/[MQTT] ]
HOST = '183.230.40.39'
[I/ONENET] RT-Thread OneNET package(V0.2.0) initialize success. msh />[I/[MQTT] ]
MQTT server connect success [D/ONENET]
(mqtt_online_callback:90) Enter mqtt_online_callback!
```

2.3.1 Upload Data

After initialization is completed, users can call the onenet_upload_cycle command to periodically upload data to the cloud platform . After entering this command, the device will upload a random value to the data stream temperature every 5 seconds and print the uploaded data to the shell window.

```
msh />onenet_upload_cycle msh /
>[D/ONENET] (onenet_upload_data:106) buffer : {"temperature":32}
[D/ONENET] (onenet_upload_data:106) buffer : {"temperature":51}
```

We open the OneNET platform and click the data flow management button in the device management interface to enter the data flow interface.

Figure **2.8:** *onenet_datastream*

Click the small arrow on the right side of the temperature data stream to display the data stream information, and we can see the data just uploaded.

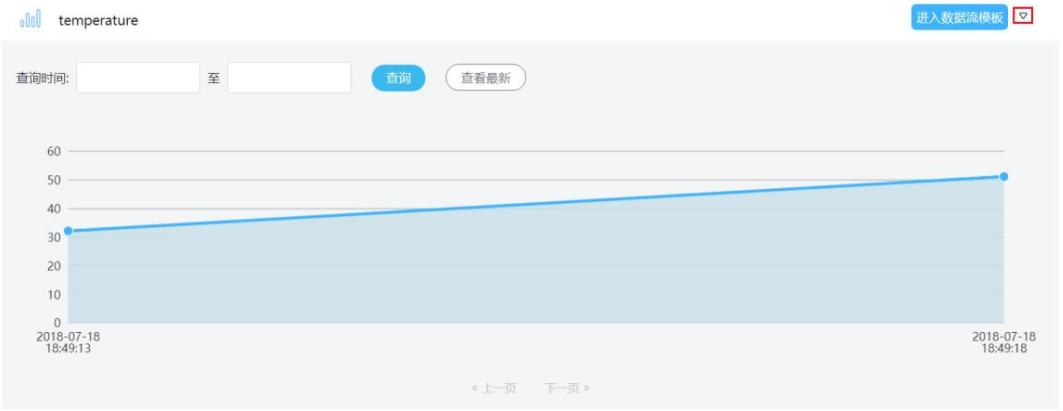

Figure **2.9:** *onenet_datapoints*

If users want to send information to other data streams, they can use the following API to upload data to the cloud platform.

onenet_mqtt_publish_digit onenet_mqtt_publish_string

The command format is as follows

```
onenet_mqtt_publish_digit Data stream name Data to be uploaded

onenet_mqtt_publish_string The string of the data stream name to be uploaded
```

If no error message is returned after entering the command, the upload is successful.

The following is an example

```
msh />onenet_mqtt_publish_digit test 1 msh /
>onenet_mqtt_publish_string test 1 msh /
>onenet_mqtt_publish_digit test 2 msh /
>onenet_mqtt_publish_string test 1
```

On the data stream management page, we can see an additional test data stream, the data in it is the data we just uploaded.

Figure **2.10:** *onenet_upload_dp*

2.3.2 Receiving Commands

During initialization, the command response callback function points to null by default. If you want to receive commands, you must set the command response callback function. Enter the command onenet_set_cmd_rsp in the shell to mount the command response callback function in the sample file. This response function will print out the command after receiving it.

```
msh />onenet_set_cmd_rsp
```

We click the Send Command button on the device management interface.

Figure **2.11:** *onenet_cmd*

Type hello rt-thread! in the pop-up window, and then click Send Command.

Figure **2.12:** *onenet_hello_rtthread*

You can see the commands issued by the cloud platform in the shell.

```
msh />onenet_set_cmd_rsp msh /
>[D/ONENET] (mqtt_callback:60) topic $creq/6db0c1b2-9a7e-5e4a-8897-
      bf62d4a3461f
receive a message [D/
ONENET] (mqtt_callback:62) message length is 18 [D/ONENET]
(onenet_cmd_rsp_cb:107) recv data is hello rt-thread!
```

Chapter **3**

How **OneNET** works

The upload of OneNET software package data and the reception of commands are based on MQTT. The initialization of OneNET is actually the initialization of the MQTT client. After the initialization is completed, the MQTT client will automatically connect to the OneNET platform. The upload of data is actually publishing messages to a specific topic. When the server has a command or response to send, it will push the message to the device.

Obtaining data streams, data points, and issuing commands are implemented based on HTTP Client. The corresponding request is sent to the OneNET platform through POST or GET, and OneNET returns the corresponding data. In this way, we can see the data uploaded by the device on the web page or mobile APP.

The following figure is a flowchart of the application display device uploading data

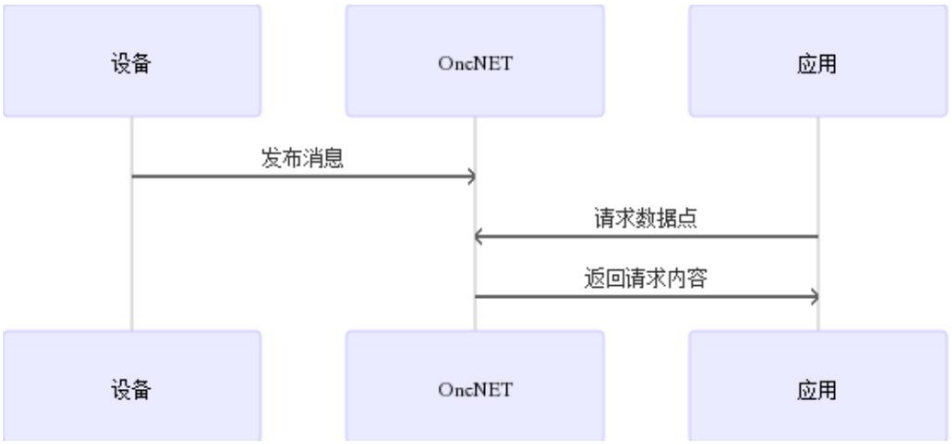

Figure **3.1:** *onenet_upload*

The following figure is a flowchart of the application sending commands to the device

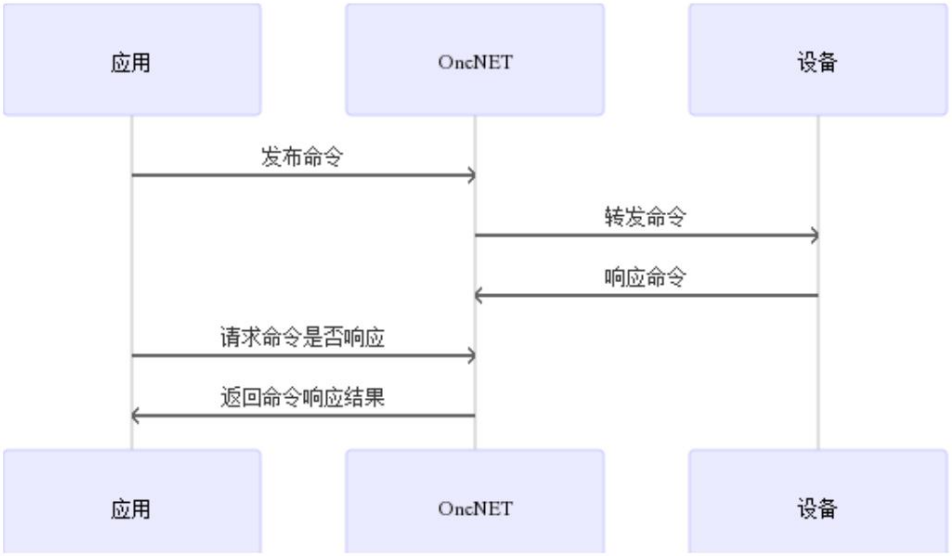

Figure **3.2:** *onenet_send_cmd*

Chapter 4

OneNET package migration instructions

This article mainly introduces the migration work that needs to be done after obtaining the OneNET software package.

The OneNET software package has stripped out the hardware platform-related features, so the porting work of OneNET itself is Very rarely, no interfaces need to be ported if auto-registration is not enabled.

If automatic registration is enabled, users need to create a new **onenet_port.c file** and add it to the project. **onenet_port.c** mainly implements functions such as obtaining registration information, obtaining device information, and saving device information after automatic registration is enabled. The interface definition is as follows:

```
/* Check if it is already registered */
rt_bool_t onenet_port_is_registed(void); /* Get
registration information*/
rt_err_t onenet_port_get_register_info(char *dev_name, char *auth_info); /* Save device information*/

rt_err_t onenet_port_save_device_info(char *dev_id, char *api_key); /* Get device information*/

rt_err_t onenet_port_get_device_info(char *dev_id, char *api_key, char *
    auth_info);
```

4.1 Obtaining registration information

rt_err_t onenet_port_get_register_info(char *ds_name, char* auth_info)

Developers only need to implement the reading and copying of registration information within this interface.

```
onenet_port_get_register_info(char *dev_name, char *auth_info) {
```

```
    /* Read or generate device name and authentication information*/

    /* Copy the device name and authentication information to dev_name and auth_info respectively*/
}
```

4.2 Save device information

rt_err_t onenet_port_save_device_info(char *dev_id, char* api_key)

Developers only need to save the device information returned by registration in the device within this interface.

```
onenet_port_save_device_info(char *dev_id, char *api_key) {

    /* Save the returned dev_id and api_key */

    /* Save the device status as registered */
}
```

4.3 Check if you are already registered

rt_bool_t onenet_port_is_registed(void)

Developers only need to return whether the device has been registered on the OneNET platform in this interface.

```
onenet_port_is_registed(void) {

    /* Read and determine the registration status of the device*/

    /* Returns whether the device has been registered*/
}
```

4.4 Get device information

rt_err_t onenet_port_get_device_info(char *dev_id, char* api_key, char *auth_info)

Developers only need to read and return device information in this interface.

```
onenet_port_get_device_info(char *dev_id, char *api_key, char *auth_info) {

        /* Read device id, api_key and authentication information*/

        /* Copy the device id, api_key and authentication information to dev_id, api_key and auth_info respectively
                middle*/
}
```

Chapter 5

OneNET Software Package User Guide

5.1 Preparation

5.1.1 OneNET registration and ENV configuration

For this part, please read the sample documentation in the software package to complete the OneNET platform registration and ENV configuration.

5.1.2 OneNET port interface migration (only applicable when automatic registration function is enabled)

If you do not enable automatic registration, you do not need to migrate the interface. If you need to enable the automatic registration function, please read the migration instructions in the software package in detail and complete the migration work.

5.2 Getting Started

5.2.1 OneNET Initialization

All the information needed to connect to the cloud platform has been configured in ENV. Simply call the onenet_mqtt_init function to initialize the device, and the device will automatically connect to the OneNET platform.

5.2.2 Pushing Data

When you need to upload data, you can select the corresponding API according to the data type to upload the data. The code example is as follows:

```
char str[] = { "hello world" };

/* Get temperature value */
```

```
temp = get_temperature_value(); /* Upload the
temperature value to the temperature data stream*/
onenet_mqtt_upload_digit("temperature",temp);

/* Upload hello world to string data stream*/
onenet_mqtt_upload_string("string",str);
```

In addition to supporting uploading numbers and strings, the package also supports uploading binary files.

You can upload binary files through onenet_mqtt_upload_bin or onenet_mqtt_upload_bin_by_path . The code example is as follows

```
uint8_t buf[] = {0x01, 0x02, 0x03};

/* Upload the 1.bin file in the root directory to the bin data stream*/
onenet_mqtt_upload_bin_by_path("bin", "/1.bin"); /* Upload the data in
buf to the bin data stream*/
onenet_mqtt_upload_bin(("bin", buf, 3);
```

5.2.3 Command Reception

OneNET supports issuing commands, which are user-defined. Users need to implement the command response callback function by themselves, and then use onenet_set_cmd_rsp_cb to load the callback function. When the device receives the command issued by the platform, it will call the command response callback function implemented by the user, wait for the callback function to be executed, and then send the response content returned by the callback function to the cloud platform. The memory to save the response must be dynamically applied. After sending the response, the program will automatically release the applied memory. The code example is as follows:

```
static void onenet_cmd_rsp_cb(uint8_t *recv_data, size_t recv_size,
        uint8_t **resp_data, size_t
*resp_size) {

    /* Apply for memory */

    /* Parsing command */

    /* Execute action */

    /* Return response */

}

int main()
```

```
{
    /* User code */

    onenet_mqtt_init();

    onenet_set_cmd_rsp_cb(onenet_cmd_rsp_cb);

    /* User code */

}
```

5.2.4 Information Acquisition

Data flow information acquisition

Users can use onenet_http_get_datastream to obtain data stream information, including data stream id, data stream last update time, data stream unit, data stream current value, etc. The obtained data stream information will be saved in the structure pointed to by the passed in datastream structure pointer. The code example is as follows

```
struct rt_onenet_ds_info ds_temp;

/* Get the temperature data stream information and save it to the ds_temp structure*/
onenet_http_get_datastream("temperature",ds_temp);
```

Data point information acquisition

Data point information can be obtained through the following 3 APIs

cJSON *onenet_get_dp_by_limit(char* ds_name, size_t limit);

cJSON *onenet_get_dp_by_start_end(char* ds_name, uint32_t start, uint32_t end, size_t limit);

cJSON *onenet_get_dp_by_start_duration(char* ds_name, uint32_t start, size_t duration, size_t limit);

All three APIs return data point information in cJSON format. The only difference is the query method.

This section explains how to use these three APIs through examples.

```
/* Get the last 10 data points of the temperature data stream*/ dp =
onenet_get_dp_by_limit("temperature",10);
```

```
/* Get the temperature data stream from 14:50:00 on July 19, 2018 to 14:55 on July 19, 2018
        The first 10 data
points in 20 seconds*/ /
* The second and third parameters are Unix timestamps*/
dp = onenet_get_dp_by_start_end("temperature",1531983000, 1531983320, 10)
        ;

/* Get the first 10 data of temperature data stream within 50 seconds after 14:50:00 on July 19, 2018
        Point information*/
/* The second parameter is the Unix
timestamp */ dp = onenet_get_dp_by_start_end("temperature",1531983000ÿ50ÿ10);
```

5.3 Notes

- Before setting the command response callback function, you must first call the onenet_mqtt_init function.
  Set the callback function to RT_NULL.
- The buffer used to store the response content in the command response callback function must be malloced.
  After that, the program will release the buffer.

Chapter 6

OneNET API

6.1 Initialization

6.1.1 OneNET Initialization

int onenet_mqtt_init(void);

OneNET initialization function, which needs to be called before using OneNET functions.

parameter	describe
none	none
return	describe
0	success
-1	Failed to obtain device information
-2	mqtt client initialization failed

6.1.2 Setting the command response function

void onenet_set_cmd_rsp_cb(void(cmd_rsp_cb) *(uint8_t* recv_data, size_t recv_size, uint8_t **resp_data, size_t *resp_size));

Set the command response callback function.

parameter	describe
recv_data	Received data

parameter	describe
recv_size	The length of the data
resp_data	Response Data
resp_size	Length of response data
return	describe
none	none

6.2 Data Upload

6.2.1 mqtt uploads data to a specified topic

rt_err_t onenet_mqtt_publish(const char *topic, const uint8_t* msg, size_t len);

Use mqtt to send messages to the specified topic.

parameter	describe
topic	theme
msg	Data to upload
len	Data length
return	describe
0	Upload Successfully
-1	upload failed

6.2.2 mqtt upload string to OneNET

rt_err_t onenet_mqtt_upload_string(const char *ds_name, const char* str);

Use mqtt to send string data to the OneNET platform.

parameter	describe
ds_name	Data Stream Name
str	The string to upload
return	describe
0	Upload Successfully

parameter	describe
-5	Not enough storage

6.2.3 Upload numbers to **OneNET** via mqtt

rt_err_t onenet_mqtt_upload_digit(const char *ds_name, const double digit);

Use mqtt to send digital data to the OneNET platform.

parameter	describe
ds_name	Data Stream Name
digit	Number to upload
return	describe
0	Upload Successfully
-5	Not enough storage

6.2.4 **mqtt** upload binary files to **OneNET**

rt_err_t onenet_mqtt_upload_bin(const char *ds_name, const uint8_t* bin,
size_t len);

Use mqtt to send binary files to the OneNET platform. It will dynamically apply for memory to save binary files.

Please make sure there is enough memory before use.

parameter	describe
ds_name	Data Stream Name
bin	binary file
len	Binary file size
return	describe
0	Upload Successfully
-1	upload failed

6.2.5 mqtt uploads binary files to OneNET via path

rt_err_t onenet_mqtt_upload_bin_by_path(const char *ds_name, const char* bin_path);

Use mqtt to send binary files to the OneNET platform.

parameter	describe
ds_name	Data Stream Name
bin_path	Binary file path
return	describe
0	Upload Successfully
-1	upload failed

6.2.6 http upload string to OneNET

rt_err_t onenet_http_upload_string(const char *ds_name, const char* str);

It is not recommended to use http to send string data to OneNET platform. It is recommended to use mqtt to upload.

parameter	describe
ds_name	Data Stream Name
str	The string to upload
return	describe
0	Upload Successfully
-5	Not enough storage

6.2.7 Upload numbers to OneNET via http

rt_err_t onenet_http_upload_digit(const char *ds_name, const double digit);

It is not recommended to use http to send digital data to the OneNET platform. It is recommended to use mqtt to upload.

parameter	describe
ds_name	Data Stream Name
digit	Number to upload

parameter	describe
return	describe
0	Upload Successfully
-5	Not enough storage

6.3 Information Acquisition

6.3.1 Obtaining data flow information

rt_err_t onenet_http_get_datastream(const char *ds_name, struct rt_onenet_ds_info* datastream);

Get the specified data stream information from the OneNET platform and save the information in the datastream structure.

parameter	describe
ds_name	Data Stream Name
datastream	Structure to store data stream information
return	describe
0	success
-1	Failed to get response
-5	Not enough storage

6.3.2 Get the last **N** data points

cJSON *onenet_get_dp_by_limit(char* ds_name, size_t limit);

Get n data point information of the specified data stream from the OneNET platform.

parameter	describe
ds_name	Data Stream Name
limit	The number of data points to obtain
return	describe
cJSON	Data point information
RT_NULL	fail

6.3.3 Get data point information within a specified time

cJSON *onenet_get_dp_by_start_end(char* ds_name, uint32_t start, uint32_t end, size_t limit);

Get n data points in a specified time period of a specified data stream from the OneNET platform. The time parameter needs to be filled in Unix timestamp.

parameter	describe
ds_name	Data Stream Name
start	Start query time
end	End query time
limit	The number of data points to obtain
return	describe
cJSON	Data point information
RT_NULL	fail

6.3.4 Get data point information at a specified time **n** seconds

cJSON *onenet_get_dp_by_start_duration(char* ds_name, uint32_t start, size_t duration, size_t limit);

Get n data points from the OneNET platform within n seconds after the specified time of the specified data stream. The time parameter requires Fill in the Unix timestamp.

parameter	describe
ds_name	Data Stream Name
start	Start query time
duration	The number of seconds to query
limit	The number of data points to obtain
return	describe
cJSON	Data point information
RT_NULL	fail

6.4 Device Management

6.4.1 Registering a device

rt_err_t onenet_http_register_device(const char *dev_name, const char* auth_info);

Register the device to the OneNET platform and return the device id and apikey. The device id and apikey will call onenet_port_save_device_info is left to the user.

parameter	describe
dev_name	Device Name
auth_info	Authentication information
return	describe
0	registration success
-5	Not enough storage

6.4.2 Save device information

rt_err_t onenet_port_save_device_info(char *dev_id, char* api_key);

Save the device information returned after registration, which needs to be implemented by the user.

parameter	describe
dev_id	Device ID
api_key	Device API Key
return	describe
0	success
-1	fail

6.4.3 Obtain device registration information

rt_err_t onenet_port_get_register_info(char *dev_name, char* auth_info);

Obtaining the information required to register the device requires user implementation.

parameter	describe
ds_name	Pointer to the device name
auth_info	Pointer to the storage of authentication information
return	describe
0	success
-1	fail

6.4.4 Obtaining device information

```
rt_err_t onenet_port_get_device_info(char dev_id, char api_key, char
*auth_info);
```

Obtaining device information for logging into the OneNET platform requires user implementation.

parameter	describe
dev_id	Pointer to the device id
api_key	Pointer to the device apikey
auth_info	Pointer to the storage of authentication information
return	describe
0	success
-1	fail

6.4.5 Is the device registered?

```
rt_bool_t onenet_port_is_registed(void);
```

Determine whether the device has been registered and needs to be implemented by the user.

parameter	describe
none	none
return	describe
RT_TURE	Already registered
RT_FALSE	unregistered