BESTLab Platform

Release 0.2.2

Wei Mu

Nov 16, 2021

CONTENTS

CHAPTER

ONE

BESTLAB_PLATFORM PACKAGE

Packages to access APIs for different platforms

1.1 Subpackages

1.1.1 bestlab_platform.hobo

class bestlab_platform.hobo.**HoboAPI**(*client_id*, *client_secret*, *user_id*, *endpoint='https://webservice.hobolink.com'*)

Bases: `object`

HOBO API

Example

hobo_api = HoboAPI(client_id, client_secret, user_id) hobo_api.get_data(["1234567", "8912345"], start_date_time, end_date_time)

__init__(*client_id*, *client_secret*, *user_id*, *endpoint='https://webservice.hobolink.com'*)

get_data(*loggers*, *start_date_time*, *end_date_time*, *warn_on_empty_data=False*)

Get data from HOBO Web Services

Parameters

- **loggers** (*List[Union[str, int]] | Union[str, int]*) – A list of Device IDs, or a single comma separated string of device ids.
- **start_date_time** (*str*) – Must be in yyyy-MM-dd HH:mm:ss format
- **end_date_time** (*str*) – Must be in yyyy-MM-dd HH:mm:ss format
- **warn_on_empty_data** (*bool*) – If True, print a warning message (to HoboLogger, which by default is your console). Has no effect on function return.

Returns JSON decoded response

Return type response (dict)

Raises **TypeError** – The "loggers" parameter type is incorrect

get(*path*, *params=None*)

Http Get.

Requests the server to return specified resources.

Parameters

- **path** (*str*) – api path
- **params** (*map*) – request parameter

Returns JSON decoded response body

Return type response (dict)

post(*path*, *body=None*)

Http Post.

Requests the server to update specified resources.

Parameters

- **path** (*str*) – api path
- **body** (*map*) – request body

Returns JSON decoded response body

Return type response (dict)

class bestlab_platform.hobo.**HoboTokenInfo**(*token_response*)

Bases: object

Hobo token info.

access_token

Access token.

token_type

"bearer".

expire_time

Time in seconds when the token will be expired.

Init HoboTokenInfo.

__init__(*token_response*)

Init HoboTokenInfo.

need_refresh()

Should we get a new the token?

Return type bool

1.1.2 bestlab_platform.tuya

class bestlab_platform.tuya.**TuyaOpenAPI**(*endpoint*, *access_id*, *access_secret*, *lang='en'*, *auto_connect=True*)

Bases: object

Open Api.

Typical usage example:

openapi = TuyaOpenAPI(ENDPOINT, ACCESS_ID, ACCESS_KEY)

Init TuyaOpenAPI.

__init__(*endpoint*, *access_id*, *access_secret*, *lang='en'*, *auto_connect=True*)

Init TuyaOpenAPI.

connect()

Connect to Tuya Cloud.

Returns connect response

Return type response

is_connect()

Whether we have an access token. Note: will return true even if the access token is expired. Token refreshing is handled internally.

Return type `bool`

get(*path*, *params=None*)

Http Get.

Requests the server to return specified resources.

Parameters

- **path** (*str*) – api path
- **params** (*map*) – request parameter

Returns response body

Return type response

post(*path*, *body=None*)

Http Post.

Requests the server to update specified resources.

Parameters

- **path** (*str*) – api path
- **body** (*map*) – request body

Returns response body

Return type response

put(*path*, *body=None*)

Http Put.

Requires the server to perform specified operations.

Parameters

- **path** (*str*) – api path
- **body** (*map*) – request body

Returns response body

Return type response

delete(*path*, *params=None*)

Http Delete.

Requires the server to delete specified resources.

Parameters

- **path** (*str*) – api path
- **params** (*map*) – request param

Returns response body

Return type response

class bestlab_platform.tuya.**TuyaTokenInfo**(*token_response*)

Bases: object

Tuya token info.

access_token

Access token.

expire_time

Valid period in seconds.

refresh_token

Refresh token.

uid

Tuya user ID.

 # platform_url

user region platform url

Init TuyaTokenInfo.

__init__(*token_response*)

Init TuyaTokenInfo.

class bestlab_platform.tuya.**TuyaDeviceManager**(*api*, *device_map=None*, *device_list=None*)

Bases: object

Manages multiple devices and provides functions to call APIs for all devices in batch Note: This is different from upstream Tuya SDK.

__init__(*api*, *device_map=None*, *device_list=None*)

get_device_status_in_batch()

Get device status for all devices in this instance in batch

Returns API response in a dictionary.

get_device_log_in_batch(*start_timestamp*, *end_timestamp*, *warn_on_empty_data=False*, *type_=7*)

Get device log stored on the Tuya platform. Note that free version of Tuya Platform only stores 7 days' data.

Parameters

- **start_timestamp** (*int | float | str*) – Start timestamp for log to be queried. Must be an 10 digit or 13 digit unix timestamp. Note that free version of Tuya only keeps one week's data.
- **end_timestamp** (*int | float | str*) – End timestamp for log to be queried. Must be an 10 digit or 13 digit unix timestamp Note that free version of Tuya only keeps one week's data.
- **warn_on_empty_data** (*bool*) – If True, print a warning message to the logger an empty page or empty final result is detected. Default: False.
- **type** (*int*) – Usually this field should be 7 ("the actual data" from the device), unless you want something else. See https://developer.tuya.com/en/docs/cloud/device-management?id=K9g6rfntdz78a#sjlx1

Returns Map of device name -> device log.

get_device_info_in_batch(*include_device_status=True*)
Get device info in batch

Parameters **include_device_status** (*bool*) – Include device status in the return fields. Default: True

Returns API response in a dictionary.

get_factory_info_in_batch()
"Query the factory information of the devices. Possible return fields are: id, uuid, sn, mac.

Returns API response in a dictionary.

send_command_in_batch(*commands*)
Issue standard instructions to control equipments.

Parameters **commands** (*list*) – issue commands.

Returns API response in a dictionary.

class bestlab_platform.tuya.**SmartHomeDeviceAPI**(*api*)
Bases: object

Tuya Smart Home Device API. See https://developer.tuya.com/en/docs/cloud/device-management?id=K9g6rfntdz78a for the list of APIs.

Example

tuya_api = tuya_api = TuyaOpenAPI("https://openapi.tuyaus.com", CLIENT_ID, CLIENT_SECRET) device_api = SmartHomeDeviceAPI(tuya_api) print(device_api.get_device_status("YOUR_DEVICE_ID_HERE"))

__init__(*api*)

get_device_info(*device_id*, *include_device_status=True*)
Get device details, including properties and the latest status of the device.

Parameters

- **device_id** (*str*) – Device ID
- **include_device_status** (*bool*) – Whether device status field should be included. Default: True

Returns API Response in a dictionary.

get_device_list_info(*device_ids*, *include_device_status=True*)
Get device info for a list of devices.

Parameters

- **device_ids** (*list[str]*) – a list of device ids.
- **include_device_status** – Include device status in the return fields. Default: True

Returns API Response in a dictionary.

get_device_status(*device_id*)
Get device status

Parameters **device_id** (*str*) – Device ID

Returns API Response in a dictionary.

get_device_list_status(*device_ids*)
Get device status for a list of devices.

Parameters **device_ids** (*list[str]*) – List of Device IDs.

Returns API Response in a dictionary.

get_factory_info(*device_ids*)
Query the factory information of the device. Possible return fields are: id, uuid, sn, mac.

Parameters **device_ids** (*list[str]*) – List of Device IDs.

Returns API Response in a dictionary.

get_device_functions(*device_id*)
Get the instruction set supported by the device, and the obtained instructions can be used to issue control.

Parameters **device_id** (*str*) – Device ID.

Returns API Response in a dictionary.

get_category_functions(*category_id*)
Query the instruction set supported by Tuya Platform in the given category. You should not need this unless you are a platform developer.

See also: https://iot.tuya.com/cloud/explorer?id=p1622082860767nqhjxa&groupId=group-home&interfaceId=470224763027539

Parameters **category_id** (*str*) – Product category.

Returns API Response in a dictionary.

get_device_specification(*device_id*)
Acquire the instruction set and status set supported by the device according to the device ID.

Parameters **device_id** (*str*) – Device ID.

Returns API Response in a dictionary.

send_commands(*device_id*, *commands*)
Issue standard instructions to control equipment

Parameters

- **device_id** (*str*) – Device ID.
- **commands** – issue commands.

Returns API Response in a dictionary.

get_device_log(*device_id*, *start_timestamp*, *end_timestamp*, *device_name=None*, *warn_on_empty_data=False*, *type_=7*)
Get device log stored on the Tuya platform. Note that free version of Tuya Platform only stores 7 days' data.

Parameters

- **device_id** (*str*) – Device ID.
- **start_timestamp** (*int | float | str*) – Start timestamp for log to be queried. Must be an 10 digit or 13 digit unix timestamp. Note that free version of Tuya only keeps one week's data.
- **end_timestamp** (*int | float | str*) – End timestamp for log to be queried. Must be an 10 digit or 13 digit unix timestamp Note that free version of Tuya only keeps one week's data.

- **device_name** (*str*) – User friendly name for your convenience. It can be any string you like, such as "PIR3"
- **warn_on_empty_data** (*bool*) – If True, print a warning message to the logger an empty page or empty final result is detected. Default: False.
- **type** (*int*) – Usually this field should be 7 ("the actual data" from the device), unless you want something else. See https://developer.tuya.com/en/docs/cloud/device-management?id=K9g6rfntdz78a#sjlx1

Returns A list of device logs. Note that the return type is not a dictionary and is not the raw response, because multiple page is expected.

1.2 Shared Exceptions

1.2.1 bestlab_platform.exceptions

Shared exception class

Exceptions

ResponseError(status_code, response_text, *args)	Exception raised for errors in the HTTP response.

exception bestlab_platform.exceptions.**ResponseError**(*status_code*, *response_text*, **args*)

Bases: Exception

Exception raised for errors in the HTTP response.

message
explanation of the error

status_code
HTTP status code

response_text
HTTP response text

__init__(*status_code*, *response_text*, **args*)

CHAPTER

TWO

INTRODUCTION

One package to access multiple different data sources through their respective API platforms.

2.1 Install

In conda or virtualenv environment, run the following commad:

```
python3 -m pip install -U bestlab_platform
```

If you are using Windows with Anaconda installed, use the following command in Anaconda Prompt:

```
pip install -U bestlab_platform
```

2.2 Usage

2.2.1 HOBO Platform

Example

```
import json
from bestlab_platform.hobo import HoboAPI


CLIENT_ID = "aaaaa"
CLIENT_SECRET = "bbbbbbbbbbbbb"
USER_ID = "123456"

# Uncomment the following lines to show all debug output
#
# import logging
# from bestlab_platform.hobo import HoboLogger
# HoboLogger.setLevel(logging.DEBUG)

hobo_api = HoboAPI(CLIENT_ID, CLIENT_SECRET, USER_ID)
print(f"access token: {hobo_api.token_info.access_token}")

devices = [
```

(continues on next page)

(continued from previous page)

```
    "123456789",
    "987654321"
]
start_time = '2021-10-15 00:00:00'
end_time = '2021-10-15 01:00:00'
response = hobo_api.get_data(devices, start_time, end_time, warn_on_empty_data=True)
# Pretty print the JSON object from response
print(json.dumps(response, indent=2))
```

hobo_example.py is another working example which reads in the secrets from a single `.env` file. It requires `python-dotenv` package.

Note: Since HOBO APIs are extremely straightforward, you can definitely write your own script without any extra packages (including this one) except for `requests`package. However, there are some extra functionality provided by this package:

- exception handling
- logging with standard format (including timestamps etc.)
- caching and reusing of existing unexpired access tokens

2.2.2 Tuya Platform

Example

```
#!/usr/bin/env python
# -*- coding: UTF-8 -*-
"""Tuya API"""
from __future__ import annotations

import json
from bestlab_platform.tuya import TuyaOpenAPI, SmartHomeDeviceAPI, TuyaDeviceManager

if __name__ == '__main__':
    ENDPOINT = "https://openapi.tuyaus.com"
    CLIENT_ID = "aaabbbbcccc"
    CLIENT_SECRET = "ddddddddddd12345"

    # Uncomment the following line to print messages when querying device logs on Tuya␣
↪platform
    #
    # import logging
    # from bestlab_platform.tuya import TUYA_LOGGER
    # TUYA_LOGGER.setLevel(logging.INFO)
    #
    # If you want to debug requests and responses, uncomment the following line.
    # TUYA_LOGGER.setLevel(logging.DEBUG)

    tuya_api = TuyaOpenAPI(ENDPOINT, CLIENT_ID, CLIENT_SECRET)
    print(tuya_api.token_info.access_token)

    # map of device name (your choice, can be any string, for readability) -> device id␣
↪(in Tuya's system)
```

(continues on next page)

(continued from previous page)

```
    devices = {
        "PIR3": "asdasdadx",
        "PIR4": "12345abcde"
    }

    # Unix timestamp in your local zone, can be 10 digit or 13 digit int, float, or␣
↪string
    start_timestamp = "1634005305000"
    end_timestamp = "1634523705000"

    # Example 1: Query in batch
    device_group = TuyaDeviceManager(tuya_api, device_map=devices)
    devices_log_map = device_group.get_device_log_in_batch(
        start_timestamp=start_timestamp,
        end_timestamp=end_timestamp,
        warn_on_empty_data=True
    )

    # Save to JSON files
    for dev_name, device_log in devices_log_map.items():
        with open(f'{dev_name}_historical_1017.json', 'w') as f:
            json.dump(device_log, f)

    # Example 2: call API for a single device
    # You can use the code above or the following. It's flexible.
    response_device_status = SmartHomeDeviceAPI(tuya_api).get_device_status(devices["PIR3
↪"])
    print(response_device_status)

    response_device_log = SmartHomeDeviceAPI(tuya_api).get_device_log(
        device_id=devices["PIR3"],
        start_timestamp=start_timestamp,
        end_timestamp=end_timestamp,
        device_name="PIR3",
        warn_on_empty_data=True
    )
    print(response_device_log)
```

tuya_example.py is another working example which reads in the secrets from a single `.env` file in your working directory. It requires `python-dotenv` package.

Why should I use this package for Tuya platform?

This package **correctly and automatically** handles connection, token caching and refreshing behind the scene so you can focus on your work. It provides functions to call most of the APIs available on their platform (available to our project account), and also added functionalities to:

- Call API for multiple devices in batch.
- Query device logs, correctly follow the pagination and return the entire log available for the period.

It is inspired by Tuya's own python SDK, but their SDK does not work for our projects, because of the following reasons:

- It is only suitable for B-to-C scenarios. It uses API endpoints **scoped to users within the cloud project**. In

order to use these endpoints, we have to physically go to where the devices are located and add them again with another mobile app, and add those devices into the correct "Asset".

- It requires subscription to Tuya's message service, which is over complicated.
- It contains too many APIs that we will never use.
- It does not have any function to query device logs. Also, Tuya's API to query the device log is paginated, which requires manual handling.

TinyTuya is another python project which uses a simple function to connect and fetch data from the Tuya IoT cloud. However, their function does not work seamlessly for us because:

- Tuya platform never refreshes current access token, unless you use the refresh token to get a new one. Access token expires two hours later after it is first obtained, which means if we don't refresh the token, we will see an error message.

Update 10/25/2021: I have managed to find out Tuya's B-to-B platform package here, which uses unscoped API endpoint and Pulsar as message service. However, there is a bug which has not been properly fixed in both of their packages. Tokens are still not refreshed in the correct way with their packages. I have already fixed on my side when I rewrote the Tuya package.

2.2.3 eGauge Platform

Not implemented yet.

2.3 API Reference

https://bestlab-platform.readthedocs.io/en/latest/index.html

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

b

INDEX

Symbols

A

B

C

D

E

G

H

I

M

N

P

R

S

T

U