python-cozify Documentation

Release v0.2.14

Juho-Pekka Kuitunen

Mar 24, 2018

Contents:

1 Low-level Cloud API calls **1**

2 Raised Exceptions **3**

3 High-level Hub functions **5**

4 Low-level Hub API calls **7**

5 python-cozify **11**
- 5.1 Installation . . . 11
- 5.2 Basic usage . . . 11
- 5.3 On Capabilities . . . 12
- 5.4 Keeping authentication valid . . . 12
- 5.5 Working Remotely . . . 13
- 5.6 Using Multiple Hubs . . . 13
- 5.7 Enconding Pitfalls . . . 13
- 5.8 Sample projects . . . 13
- 5.9 Development . . . 14

Python Module Index **15**

CHAPTER 1

Low-level Cloud API calls

Module for handling Cozify Cloud API 1:1 functions

cozify.cloud_api.**cloudBase**
str – API endpoint including version

cozify.cloud_api.**emaillogin**(*email*, *otp*)
Raw Cloud API call, request cloud token with email address & OTP.

Parameters

- **email** (*str*) – Email address connected to Cozify account.
- **otp** (*int*) – One time passcode.

Returns cloud token

Return type str

cozify.cloud_api.**hubkeys**(*cloud_token*)
1:1 implementation of user/hubkeys

Parameters **cloud_token** (*str*) –

Returns Map of hub_id: hub_token pairs.

Return type dict

cozify.cloud_api.**lan_ip**()
1:1 implementation of hub/lan_ip

This call will fail with an APIError if the requesting source address is not the same as that of the hub, i.e. if they're not in the same NAT network. The above is based on observation and may only be partially true.

Returns List of Hub ip addresses.

Return type list

cozify.cloud_api.**refreshsession**(*cloud_token*)
1:1 implementation of user/refreshsession

Parameters **cloud_token** (*str*) –

Returns New cloud remote authentication token. Not automatically stored into state.

Return type str

cozify.cloud_api.**remote**(*cloud_token*, *hub_token*, *apicall*, *payload=None*, ***kwargs*)
1:1 implementation of 'hub/remote'

Parameters

- **cloud_token** (*str*) – Cloud remote authentication token.
- **hub_token** (*str*) – Hub authentication token.
- **apicall** (*str*) – Full API call that would normally go directly to hub, e.g. '/cc/1.6/hub/colors'
- **payload** (*str*) – json string to use as payload, changes method to PUT.

Returns Requests response object.

Return type requests.response

cozify.cloud_api.**requestlogin**(*email*)
Raw Cloud API call, request OTP to be sent to account email address.

Parameters **email** (*str*) – Email address connected to Cozify account.

CHAPTER 2

Raised Exceptions

exception cozify.Error.**APIError**(*status_code*, *message*)
Error raised for non-200 API return codes

Parameters

- **status_code** (*int*) – HTTP status code returned by the API
- **message** (*str*) – Potential error message returned by the API

status_code
int – HTTP status code returned by the API

message
str – Potential error message returned by the API

exception cozify.Error.**AuthenticationError**(*message*)
Error raised for nonrecoverable authentication failures.

Parameters **message** (*str*) – Human readable error description

message
str – Human readable error description

CHAPTER 3

High-level Hub functions

CHAPTER 4

Low-level Hub API calls

Module for all Cozify Hub API 1:1 calls

cozify.hub_api.**apiPath**
str – Hub API endpoint path including version. Things may suddenly stop working if a software update increases the API version on the Hub. Incrementing this value until things work will get you by until a new version is published.

cozify.hub_api.**devices**(***kwargs*)
1:1 implementation of /devices API call. For remaining kwargs see cozify.hub_api.get()

Parameters ****mock_devices** (*dict*) – If defined, returned as-is as if that were the result we received.

Returns Full live device state as returned by the API

Return type dict

cozify.hub_api.**devices_command**(*command*, ***kwargs*)
1:1 implementation of /devices/command. For kwargs see cozify.hub_api.put()

Parameters **command** (*dict*) – dictionary of type DeviceData containing the changes wanted. Will be converted to json.

Returns What ever the API replied or raises an APIEerror on failure.

Return type str

cozify.hub_api.**devices_command_generic**(*, *device_id*, *command=None*, *request_type*, ***kwargs*)
Command helper for CMD type of actions. No checks are made wether the device supports the command or not. For kwargs see cozify.hub_api.put()

Parameters

- **device_id** (*str*) – ID of the device to operate on.
- **request_type** (*str*) – Type of CMD to run, e.g. CMD_DEVICE_OFF

- **command** (*dict*) – Optional dictionary to override command sent. Defaults to None which is interpreted as { device_id, type }

Returns What ever the API replied or raises an APIError on failure.

Return type str

cozify.hub_api.**devices_command_off**(*device_id*, ***kwargs*)
Command helper for CMD_DEVICE_OFF.

Parameters **device_id** (*str*) – ID of the device to operate on.

Returns What ever the API replied or raises an APIException on failure.

Return type str

cozify.hub_api.**devices_command_on**(*device_id*, ***kwargs*)
Command helper for CMD_DEVICE_ON.

Parameters **device_id** (*str*) – ID of the device to operate on.

Returns What ever the API replied or raises an APIError on failure.

Return type str

cozify.hub_api.**devices_command_state**(***, *device_id*, *state*, ***kwargs*)
Command helper for CMD type of actions. No checks are made wether the device supports the command or not. For kwargs see cozify.hub_api.put()

Parameters

- **device_id** (*str*) – ID of the device to operate on.
- **state** (*dict*) – New state dictionary containing changes.

Returns What ever the API replied or raises an APIError on failure.

Return type str

cozify.hub_api.**get**(*call*, *hub_token_header=True*, *base='/cc/1.8'*, ***kwargs*)
GET method for calling hub API.

Parameters

- **call** (*str*) – API path to call after apiPath, needs to include leading /.
- **hub_token_header** (*bool*) – Set to False to omit hub_token usage in call headers.
- **base** (*str*) – Base path to call from API instead of global apiPath. Defaults to apiPath.
- ****host** (*str*) – ip address or hostname of hub.
- ****hub_token** (*str*) – Hub authentication token.
- ****remote** (*bool*) – If call is to be local or remote (bounced via cloud).
- ****cloud_token** (*str*) – Cloud authentication token. Only needed if remote = True.

cozify.hub_api.**hub**(***kwargs*)
1:1 implementation of /hub API call. For kwargs see cozify.hub_api.get()

Returns Hub state dict.

Return type dict

cozify.hub_api.**put**(*call*, *payload*, *hub_token_header=True*, *base='/cc/1.8'*, ***kwargs*)
PUT method for calling hub API. For rest of kwargs parameters see get()

Parameters

- **call** (*str*) – API path to call after apiPath, needs to include leading /.
- **payload** (*str*) – json string to push out as the payload.
- **hub_token_header** (*bool*) – Set to False to omit hub_token usage in call headers.
- **base** (*str*) – Base path to call from API instead of global apiPath. Defaults to apiPath.

cozify.hub_api.**tz**(***kwargs*)

1:1 implementation of /hub/tz API call. For kwargs see cozify.hub_api.get()

Returns Timezone of the hub, for example: 'Europe/Helsinki'

Return type str

CHAPTER 5

python-cozify

Unofficial Python3 API bindings for the (unpublished) Cozify API. Includes high-level helpers for easier use of the APIs, for example an automatic authentication flow, and low-level 1:1 API functions.

5.1 Installation

The recommended way is to install from PyPi:

```
sudo -H pip3 install cozify
```

or clone the master branch of this repo (master stays at current release) and:

```
sudo python3 setup.py install
```

To develop python-cozify clone the devel branch and submit pull requests against the devel branch. New releases are cut from the devel branch as needed.

5.2 Basic usage

These are merely some simple examples, for the full documentation see: *http://python-cozify.readthedocs.io/en/latest/*

5.2.1 read devices by capability, print temperature data

```
from cozify import hub
devices = hub.devices(capabilities=hub.capability.TEMPERATURE)
for id, dev in devices.items():
  print('{0}: {1}C'.format(dev['name'], dev['state']['temperature']))
```

5.2.2 only authenticate

```
from cozify import cloud
cloud.authenticate()
# authenticate() is interactive and usually triggered automatically
# authentication data is stored in ~/.config/python-cozify/python-cozify.cfg
```

5.2.3 authenticate with a non-default state storage

```
from cozify import cloud, config
config.setStatePath('/tmp/testing-state.cfg')
cloud.authenticate()
# authentication and other useful data is now stored in the defined location instead␣
↪of ~/.config/python-cozify/python-cozify.cfg
# you could also use the environment variable XDG_CONFIG_HOME to override where␣
↪config files are stored
```

5.3 On Capabilities

The most practical way to "find" devices for operating on is currently to filter the devices list by their capabilties. The most up to date list of recognized capabilities can be seen at cozify/hub.py

If the capability you need is not yet supported, open a bug to get it added. One way to compare your live hub device's capabilities to those implemented is running the util/capabilities_list.py tool. It will list implemented and gathered capabilities from your live environment. To get all of your previously unknown capabilities implemented, just copy-paste the full output of the utility into a new bug.

In short capabilities are tags assigned to devices by Cozify that mostly guarantee the data related to that capability will be in the same format and structure. For example the capabilities based example code in this document filters all the devices that claim to support temperature and reads their name and temperature state. Multiple capabilities can be given in a filter by providing a list of capabilities. By default any capability in the list can match (OR filter) but it can be flipped to AND mode where every capability must be present on a device for it to qualify. For example, if you only want multi-sensors that support both temperature and humidity monitoring you could define a filter as:

```
devices = hub.devices(capabilities=[ hub.capability.TEMPERATURE, hub.capability.
↪HUMIDITY ], and_filter=True)
```

5.4 Keeping authentication valid

If the cloud token expires, the only option to get a new one is an interactive prompt for an OTP. Since most applications will want to avoid that as much as possible there are a few tips to keep a valid token alive. At the time of writing tokens are valid for 28 days during which they can be seamlessly refreshed.

In most cases it isn't necessary to directly call cloud.refresh() if you're already using cloud.ping() to test token validity. cloud.ping() will also perform a refresh check after a successful ping unless explicitly told not to do so.

To refresh a token you can call as often as you want:

```
cloud.refresh()
```

By default keys older than a day will be re-requested and otherwise no refresh is performed. The refresh can be forced:

```
cloud.refresh(force=True)
```

And the expiry duration can be altered (also when calling cloud.ping()):

```
cloud.refresh(expiry=datetime.timedelta(days=20))
# or
cloud.ping(autorefresh=True, expiry=datetime.timedelta(days=20))
```

5.5 Working Remotely

By default queries to the hub are attempted via local LAN. Also by default "remoteness" autodetection is on and thus if it is determined during cloud.authentication() or a hub.ping() call that you seem to not be in the same network, the state is flipped. Both the remote state and autodetection can be overriden in most if not all funcions by the boolean keyword arguments 'remote' and 'autoremote'. They can also be queried or permanently changed by the hub.remote() and hub.autoremote() functions.

5.6 Using Multiple Hubs

Everything has been designed to support multiple hubs registered to the same Cozify Cloud account. All hub operations can be targeted by setting the keyword argument 'hub_id' or 'hub_name'. The developers do not as of yet have access to multiple hubs so proper testing of multi functionality has not been performed. If you run into trouble, please open bugs so things can be improved.

The remote state of hubs is kept separately so there should be no issues calling your home hub locally but operating on a summer cottage hub remotely at the same time.

5.7 Enconding Pitfalls

The hub provides data encoded as a utf-8 json string. Python-cozify transforms this into a Python dictionary where string values are kept as unicode strings. Normally this isn't an issue, as long as your system supports utf-8. If not, you will run into trouble printing for example device names with non-ascii characters:

> UnicodeEncodeError: 'ascii' codec can't encode character 'xe4' in position 34: ordinal not in range(128)

The solution is to change your system locale to support utf-8. How this is done is however system dependant. As a first test try temporarily overriding your locale:

```
LC_ALL='en_US.utf8' python3 program.py
```

5.8 Sample projects

- github.com/Artanicus/cozify-temp - Store Multisensor data into InfluxDB
- Take a look at the util/ directory for some crude small tools using the library that have been useful during development.
- File an issue to get your project added here

5.9 Development

To develop python-cozify clone the devel branch and submit pull requests against the devel branch. New releases are cut from the devel branch as needed.

5.9.1 Tests

pytest is used for unit tests. Certain tests are marked as "live" tests and require an active authentication state and a real hub to query against. Live tests are non-destructive. Some tests are marked as "destructive" and will cause changes such as a light being turned on or tokens getting invalidated on purpose.

During development you can run the test suite right from the source directory:

```
pytest -v cozify/
# or include the live tests as well:
pytest -v cozify/ --live
# or for the brave, also run destructive tests (also implies --live):
pytest -v cozify/ --destructive
```

To run the test suite on an already installed python-cozify:

```
pytest -v --pyargs cozify
```

5.9.2 Roadmap, aka. Current Limitations

- Authentication flow has been improved quite a bit but it would benefit a lot from real-world feedback.
- For now there are only read calls. Next up is implementing ~all hub calls at the raw level and then wrapping them for ease of use. If there's something you want to use sooner than later file an issue so it can get prioritized!
- Device model is non-existant and the old implementations are bad and deprecated. Active work ongoing to filter by capability at a low level first, then perhaps a more object oriented model on top of that.

Python Module Index

C

cozify.cloud_api, 1
cozify.hub_api, 7

Index

A

APIError, 3
apiPath (in module cozify.hub_api), 7
AuthenticationError, 3

C

cloudBase (in module cozify.cloud_api), 1
cozify.cloud_api (module), 1
cozify.hub_api (module), 7

D

devices() (in module cozify.hub_api), 7
devices_command() (in module cozify.hub_api), 7
devices_command_generic() (in module cozify.hub_api), 7
devices_command_off() (in module cozify.hub_api), 8
devices_command_on() (in module cozify.hub_api), 8
devices_command_state() (in module cozify.hub_api), 8

E

emaillogin() (in module cozify.cloud_api), 1

G

get() (in module cozify.hub_api), 8

H

hub() (in module cozify.hub_api), 8
hubkeys() (in module cozify.cloud_api), 1

L

lan_ip() (in module cozify.cloud_api), 1

M

message (cozify.Error.APIError attribute), 3
message (cozify.Error.AuthenticationError attribute), 3

P

put() (in module cozify.hub_api), 8

R

refreshsession() (in module cozify.cloud_api), 1
remote() (in module cozify.cloud_api), 2
requestlogin() (in module cozify.cloud_api), 2

S

status_code (cozify.Error.APIError attribute), 3

T

tz() (in module cozify.hub_api), 9