python-cozify Documentation

Release v0.3.0

Juho-Pekka Kuitunen

Jun 16, 2019

Contents:

Module for handling Cozify Cloud highlevel operations.

`cozify.cloud.`**`token_expiry`**

timedelta object for duration how often cloud_token will be considered valid for refresh.

Type datetime.timedelta

`cozify.cloud.`**`authenticate`**(*trustCloud=True*, *trustHub=True*, *remote=False*, *autoremote=True*, *autorefresh=True*, *expiry=None*)

Authenticate with the Cozify Cloud and any Hubs found.

Interactive only when absolutely needed, mostly on the first run. By default authentication is run selectively only for the portions needed. Hub authentication lives in the Cloud module since the authentication is obtained from the cloud.

Authentication is a multistep process:

- trigger sending OTP to email address
- perform email login with OTP to acquire cloud token
- acquire hub information and authenticate with hub with cloud token
- store hub token for further use

Parameters

- **`trustCloud`** (*`bool`*) – Trust current stored state of cloud auth. Default True.
- **`trustHub`** (*`bool`*) – Trust current stored state of hub auth. Default True.
- **`remote`** (*`bool`*) – Treat a hub as being outside the LAN, i.e. calls will be routed via the Cozify Cloud remote call system. Defaults to False.
- **`autoremote`** (*`bool`*) – Autodetect hub LAN presence and flip to remote mode if needed. Defaults to True.
- **`autorefresh`** (*`bool`*) – Autorefresh cloud and hub tokens if they're no longer valid but can still be rescued. Defaults to True.
- **`expiry`** (*`datetime.timedelta`*) – timedelta object for duration how often cloud_token will be auto-refreshed. Defaults to cloud.token_expiry

Returns True on authentication success. Failure will result in an exception.

Return type bool

`cozify.cloud.`**`email`**(*new_email=None*)

Get currently used cloud account email or set a new one.

Returns Cloud user account email address.

Return type str

`cozify.cloud.`**`ping`**(*autorefresh=True*, *expiry=None*)

Test cloud token validity. On success will also trigger a refresh if it's needed by the current key expiry.

Parameters

- **`refresh`** (*`bool`*) – Wether to perform a autorefresh check after a successful ping. Defaults to True.
- **`expiry`** (*`datetime.timedelta`*) – timedelta object for duration how often cloud_token will be auto-refreshed when cloud.ping() is called. Defaults to cloud.token_expiry()

Returns validity of stored token.

Return type bool

`cozify.cloud.`**`refresh`**(*force=False*, *expiry=None*)

Renew current cloud token and store new token in state.

This call will only succeed if the current cloud token is still valid. A new refreshed token is requested from the API only if sufficient time has passed since the previous refresh.

Parameters

- **`force`** (*`bool`*) – Set to True to always perform a refresh regardless of time passed since previous refresh.
- **`expiry`** (*`datetime.timedelta`*) – timedelta object for duration of refresh expiry. Defaults to cloud.token_expiry

Returns Success of refresh attempt, True also when expiry wasn't over yet even though no refresh was performed.

Return type bool

`cozify.cloud.`**`resetState`**()

Reset stored cloud state.

Any further authentication flow will start from a clean slate. Hub state is left intact.

`cozify.cloud.`**`token`**(*new_token=None*, *commit=True*)

Get currently used cloud_token or set a new one.

Parameters

- **`new_token`** (*`str`*) – New cloud_token to store. Use cautiously since an invalid token means interactive recovery.
- **`commit`** (*`bool`*) – Wether to commit the new value to persistent storage immediately or not.

Returns Cloud remote authentication token.

Return type str

CHAPTER 1

Low-level Cloud API calls

Module for handling Cozify Cloud API 1:1 functions

`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.`**`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

Low-level State functions

In general there is little need to touch these unless you want to alter some assumed defaults.

Module for handling consistent state storage.

`cozify.config.`**`state_path`**
file path where state storage is kept. By default XDG conventions are used. (Most likely ~/.config/python-cozify/python-cozify.cfg)

Type str

`cozify.config.`**`state`**
State object used for in-memory state. By default initialized with _initState.

Type configparser.ConfigParser

`cozify.config.`**`latest_version`**
Current up to date version number. Anything encountered lower than this will go through autoconversion.

Type int

`cozify.config.`**`commit`**`(`*tmpstate=None*`)`
Write current state to file storage.

Parameters **`tmpstate`** (*`configparser.ConfigParser`*) – State object to store instead of default state.

`cozify.config.`**`dump`**`()`
Print out current state file to stdout. Long values are truncated since this is only for visualization.

`cozify.config.`**`set_state_path`**`(`*filepath=None*, *copy_current=False*`)`
Set state storage path. Useful for example for testing without affecting your normal state. Call with no arguments to reset back to autoconfigured location.

Parameters

- **`filepath`** (*`str`*) – file path to use as new storage location. Defaults to XDG defined path.
- **`copy_current`** (*`bool`*) – Instead of initializing target file, dump previous state into it.

`cozify.config.`**`version`**`(`*new_version=None*`)`

Return or manipulate current config version.

Parameters **`new_version`** (*`int`*) – New version number to assume for the current config. You probably don't want to do this!

CHAPTER 3

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`
HTTP status code returned by the API

Type int

`message`
Potential error message returned by the API

Type str

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

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

`message`
Human readable error description

Type str

CHAPTER 4

Low-level HTTP functions

In general there is little need to touch these unless you want to do custom low-level calls.

Helper module for hub_api & cloud_api

`cozify.http.`**`session`**

Global session used for communications.

Type requests.Session

`cozify.http.`**`get`**(*call*, ***, *token*, *headers=None*, *params=None*, ***kwargs*)

GET method for calling hub or cloud APIs.

Parameters

- **`call`** (*`str`*) – Full API URL.
- **`token`** (*`str`*) – Either hub_token or cloud_token depending on target of call.
- **`headers`** (*`dict`*) – Any additional headers to add to the call.
- **`params`** (*`dict`*) – Any additional URL parameters to pass.
- **`**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.http.`**`post`**(*call*, ***, *token*, *headers=None*, *payload=None*, *params=None*, ***kwargs*)

POST method for calling hub our cloud APIs.

Parameters

- **`call`** (*`str`*) – Full API URL.
- **`payload`** (*`str`*) – json string to push out as the payload.
- **`token`** (*`str`*) – Either hub_token or cloud_token depending on target of call.
- **`headers`** (*`dict`*) – Any additional headers to add to the call.
- **`params`** (*`dict`*) – Any additional URL parameters to pass.
- **`**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.http.`**`put`**(*call*, *payload*, ***, *token*, *headers=None*, *params=None*, ***kwargs*)

PUT method for calling hub or cloud APIs.

Parameters

- **call** (*str*) – Full API URL.
- **payload** (*str*) – json string to push out as the payload.
- **token** (*str*) – Either hub_token or cloud_token depending on target of call.
- **headers** (*dict*) – Any additional headers to add to the call.
- **params** (*dict*) – Any additional URL parameters to pass.
- ****remote** (*bool*) – If call is to be local or remote (bounced via cloud).
- ****cloud_token** (*str*) – Cloud authentication token. Only needed if remote = True.

CHAPTER 5

High-level Hub functions

Module for handling highlevel Cozify Hub operations.

`cozify.hub.`**`capability`**

Enum of known device capabilities. Alphabetically sorted, numeric value not guaranteed to stay constant between versions if new capabilities are added.

Type capability

`cozify.hub.`**`autoremote`**(*hub_id=None*, *new_state=None*, *commit=True*)

Get autoremote status of hub or set a new value for it. Always returns current state at the end.

Parameters

- **`hub_id`** (*`str`*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub. Defaults to hub.default()
- **`new_state`** (*`bool`*) – New autoremoteness state to set for hub. True means remote will be automanaged. Defaults to None when only the current value will be returned.
- **`commit`** (*`bool`*) – True to commit new state after set. Defaults to True.

Returns True for a hub with autoremote enabled.

Return type bool

`class` `cozify.hub.`**`capability`**

An enumeration.

`cozify.hub.`**`default`**()

Return id of default Hub.

If default hub isn't known an AttributeError will be raised.

`cozify.hub.`**`device_eligible`**(*device_id*, *capability_filter*, *devs=None*, *state=None*, ***kwargs*)

Check if device matches a AND devices filter.

Parameters

- **`device_id`** (*`str`*) – ID of the device to check.

- **capability_filter** (*hub.capability*) – Single hub.capability or a list of them to match against.
- **devs** (*dict*) – Optional devices dictionary to use. If not defined, will be retrieved live.
- **state** (*dict*) – Optional state dictionary, will be updated with state of checked device if device is eligible. Previous data in the dict is preserved unless it's overwritten by new values.

Returns True if filter matches.

Return type bool

`cozify.hub.`**`device_exists`**(*device_id*, *devs=None*, *state=None*, ***kwargs*)

Check if device exists.

Parameters

- **device_id** (*str*) – ID of the device to check.
- **devs** (*dict*) – Optional devices dictionary to use. If not defined, will be retrieved live.
- **state** (*dict*) – Optional state dictionary, will be updated with state of checked device if device is eligible. Previous data in the dict is preserved unless it's overwritten by new values.

Returns True if filter matches.

Return type bool

`cozify.hub.`**`device_off`**(*device_id*, ***kwargs*)

Turn off a device that is capable of turning off. Eligibility is determined by the capability ON_OFF.

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

`cozify.hub.`**`device_on`**(*device_id*, ***kwargs*)

Turn on a device that is capable of turning on. Eligibility is determined by the capability ON_OFF.

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

`cozify.hub.`**`device_reachable`**(*device_id*, *devs=None*, *state=None*, ***kwargs*)

Check if device exists and is reachable.

Parameters

- **device_id** (*str*) – ID of the device to check.
- **devs** (*dict*) – Optional devices dictionary to use. If not defined, will be retrieved live.
- **state** (*dict*) – Optional state dictionary, will be updated with state of checked device if device is eligible. Previous data in the dict is preserved unless it's overwritten by new values.

Returns True if filter matches.

Return type bool

`cozify.hub.`**`device_state_replace`**(*device_id*, *state*, ***kwargs*)

Replace the entire state of a device with the provided state. Useful for example for returning to a stored state.

Parameters

- **device_id** (*str*) – ID of the device to toggle.
- **state** (*dict*) – State dictionary to push out.

- `**hub_id` (*str*) – optional id of hub to operate on. A specified hub_id takes presedence over a hub_name or default Hub.
- `**hub_name` (*str*) – optional name of hub to operate on.
- `**remote` (*bool*) – Remote or local query.

`cozify.hub.`**`device_toggle`**(*device_id*, ***kwargs*)

Toggle power state of any device capable of it such as lamps. Eligibility is determined by the capability ON_OFF.

Parameters

- `device_id` (*str*) – ID of the device to toggle.
- `**hub_id` (*str*) – optional id of hub to operate on. A specified hub_id takes presedence over a hub_name or default Hub.
- `**hub_name` (*str*) – optional name of hub to operate on.
- `**remote` (*bool*) – Remote or local query.

`cozify.hub.`**`devices`**(***, *capabilities=None*, *and_filter=False*, *devs=None*, ***kwargs*)

Get up to date full devices data set as a dict. Optionally can be filtered to only include certain devices.

Parameters

- `capabilities` (*cozify.hub.capability*) – Single or list of cozify.hub.capability types to filter by, for example: [cozify.hub.capability.TEMPERATURE, cozify.hub.capability.HUMIDITY]. Defaults to no filtering.
- `and_filter` (*bool*) – Multi-filter by AND instead of default OR. Defaults to False.
- `devs` (*dict*) – Optional devices dictionary to use. If not defined, will be retrieved live.
- `**hub_name` (*str*) – optional name of hub to query.
- `**hub_id` (*str*) – optional id of hub to query. A specified hub_id takes presedence over a hub_name or default Hub. Providing incorrect hub_id's will create cruft in your state but it won't hurt anything beyond failing the current operation.
- `**remote` (*bool*) – Remote or local query.

Returns full live devices dictionary as returned by the API

Return type dict

`cozify.hub.`**`exists`**(*hub_id=None*)

Check for existance of hub in local state.

Parameters `hub_id` (*str*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub.

`cozify.hub.`**`host`**(*hub_id=None*)

Get hostname of hub

Parameters `hub_id` (*str*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub.

Returns ip address of matching hub. Be aware that this may be empty if the hub is only known remotely and will still give you an ip address even if the hub is currently remote and an ip address was previously locally known.

Return type str

`cozify.hub.`**`hub_id`**(*hub_name*)

Get hub id by it's name.

Parameters **`hub_name`** (*`str`*) – Name of hub to query. The name is given when registering a hub to an account.

Returns hub_id on success, raises an attributeerror on failure.

Return type str

`cozify.hub.`**`light_brightness`**(*device_id*, *brightness*, *transition=0*, ***kwargs*)

Set brightness of a light.

Parameters

- **`device_id`** (*`str`*) – ID of the device to operate on.
- **`brightness`** (*`float`*) – Brightness in the range of [0, 1]. If outside the range a ValueError is raised.
- **`transition`** (*`int`*) – Transition length in milliseconds. Defaults to instant.

`cozify.hub.`**`light_color`**(*device_id*, *hue*, *saturation=1.0*, *transition=0*, ***kwargs*)

Set color (hue & saturation) of a light.

Parameters

- **`device_id`** (*`str`*) – ID of the device to operate on.
- **`hue`** (*`float`*) – Hue in the range of [0, Pi*2]. If outside the range a ValueError is raised.
- **`saturation`** (*`float`*) – Saturation in the range of [0, 1]. If outside the range a ValueError is raised. Defaults to 1.0 (full saturation.)
- **`transition`** (*`int`*) – Transition length in milliseconds. Defaults to instant.

`cozify.hub.`**`light_temperature`**(*device_id*, *temperature=2700*, *transition=0*, ***kwargs*)

Set temperature of a light.

Parameters

- **`device_id`** (*`str`*) – ID of the device to operate on.
- **`temperature`** (*`float`*) – Temperature in Kelvins. If outside the operating range of the device the extreme value is used. Defaults to 2700K.
- **`transition`** (*`int`*) – Transition length in milliseconds. Defaults to instant.

`cozify.hub.`**`name`**(*hub_id=None*)

Get hub name

Parameters **`hub_id`** (*`str`*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub.

Returns Hub name or None if the hub wasn't found.

Return type str

`cozify.hub.`**`ping`**(*autorefresh=True*, ***kwargs*)

Perform a cheap API call to trigger any potential APIError and return boolean for success/failure. For optional kwargs see cozify.hub_api.get()

Parameters

- **`autorefresh`** (*`bool`*) – Wether to perform a autorefresh after an initially failed ping. If successful, will still return True. Defaults to True.

- **`**hub_id`** (*str*) – Hub to ping or default if neither id or name set.
- **`**hub_name`** (*str*) – Hub to ping by name.

Returns True for a valid and working hub authentication state.

Return type bool

`cozify.hub.`**`remote`**(*hub_id=None*, *new_state=None*, *commit=True*)
Get remote status of hub or set a new value for it. Always returns current state at the end.

Parameters

- **`**hub_id`** (*str*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub. Defaults to hub.default()
- **`new_state`** (*bool*) – New remoteness state to set for hub. True means remote. Defaults to None when only the current value will be returned.
- **`commit`** (*bool*) – True to commit new state after set. Defaults to True.

Returns True for a hub considered remote.

Return type bool

`cozify.hub.`**`token`**(*hub_id=None*, *new_token=None*, *commit=True*)
Get hub_token or set a new value for it.

Parameters

- **`hub_id`** (*str*) – Id of hub to query. The id is a string of hexadecimal sections used internally to represent a hub.
- **`commit`** (*bool*) – True to commit new state after set. Defaults to True.

Returns Hub authentication token.

Return type str

`cozify.hub.`**`tz`**(***kwargs*)
Get timezone of given hub or default hub if no id is specified. For more optional kwargs see cozify.hub_api.get()

Parameters **`**hub_id`** (*str*) – Hub to query, by default the default hub is used.

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

Return type str

CHAPTER 6

Low-level Hub API calls

Module for all Cozify Hub API 1:1 calls

`cozify.hub_api.`**`colors`**`(`*`**kwargs`*`)`

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

Returns List of hexadecimal color codes of all defined custom colors.

Return type list

`cozify.hub_api.`**`devices`**`(`*`**kwargs`*`)`

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

Parameters **`**devs`** (*`dict`*) – If defined, returned as-is.

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.`**`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.`**`lpd433devices`**(***kwargs*)
1:1 implementation of /hub/433devices API call. For kwargs see cozify.hub_api.get()

Returns List of dictionaries describing all 433MHz devices paired with hub.

Return type list

`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 7

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.

7.1 Installation

Python3.4 is the current minimum supported version of Python. For example Ubuntu 14.04 LTS is still barely supported in theory but not explicitly tested for.

The recommended way is to install from PyPi:

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

To benefit from new features you'll need to update the library (pip does not auto-update):

```
sudo -H pip3 install -U 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.

7.2 Basic usage

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

7.2.1 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']))
```

7.2.2 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
```

7.2.3 authenticate with a non-default state storage

```
from cozify import cloud, config
config.set_state_path('/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
```

7.3 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)
```

7.4 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))
```

7.5 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.

7.6 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.

7.7 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
```

7.8 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

7.9 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.

7.9.1 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.
- A few tests are marked as "remote" and are only expected to succeed when testing remotely, i.e. outside the LAN of the hub.
- Most tests are marked as "logic" and do not require anything external. If no set is defined, only logic tests are run.

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

```
pytest
# or run only live tests:
pytest -m live
# run everything except destructive tests:
pytest -m "not destructive"
```

To run the test suite on an already installed python-cozify (defining a set is mandatory, otherwise ALL sets are run including destructive):

```
pytest -v -m logic --pyargs cozify
```

7.9.2 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

Index