toonlib Documentation

Release 0.3.0

Costas Tyfoxylos

Apr 26, 2017

Contents

Contents:

CHAPTER 1

toonlib

A library to interact with Eneco's toon.

Main information is cached for 5 minutes before reaching out to the api for freshness. Assigning values to either the thermostat or the thermostat state effectively changing the temperature clears the cache so the next call will get fresh info about the settings.

Most returned information is currently modeled as a named tuple since they need no intelligence. The smartplugs and lights are proper objects since they need to call the api and refresh their values. Everything else will evolve according to it's needs.

The library exposes the data that toon uses to graph its consumption both as flow data (hourly consumption for the day) and graph data (yearly, monthly, weekly, daily, hourly) consumption.

- Documentation: http://toonlib.readthedocs.io/en/latest/

Features

- Reads values for gas, electric, temperature.
- Identifies connected hue lights and fibaro smartplugs
- Can read and set temperature and thermostat state
- Can turn lights or plugs on, off or toggle their state
- Can get consumption values from fibaro plugs
- More ...

TODO

- Change the caching library to a name spaced one.
- Fine tune the caching sanely across all required objects

- Properly implement caching for flow and graph data information

CHAPTER 2

Installation

At the command line:

```
$ pip install toonlib
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv toonlib
$ pip install toonlib
```

CHAPTER 3

Usage

To use toonlib in a project:

```
from toonlib import Toon

eneco_username = 'USERNAME'
eneco_password = 'PASSWORD'
toon = Toon(eneco_username, eneco_password)
```

Print information about the agreement. Attributes are self explanatory.

```
print(toon.agreement.id)
print(toon.agreement.checksum)
print(toon.agreement.city)
print(toon.agreement.display_common_name)
print(toon.agreement.display_hardware_version)
print(toon.agreement.display_software_version)
print(toon.agreement.heating_type)
print(toon.agreement.house_number)
print(toon.agreement.boiler_management)
print(toon.agreement.solar)
print(toon.agreement.toonly)
print(toon.agreement.post_code)
print(toon.agreement.street_name)
```

Print information about the client. Attributes are self explanatory.

```
print(toon.client.id)
print(toon.client.checksum)
print(toon.client.hash)
print(toon.client.sample)
print(toon.client.personal_details.number)
print(toon.client.personal_details.email)
print(toon.client.personal_details.first_name)
print(toon.client.personal_details.last_name)
print(toon.client.personal_details.middle_name)
```

```
print(toon.client.personal_details.mobile_number)
print(toon.client.personal_details.phone_number)
```

Print information about the gas. Values are cached internally for 5 minutes so as to not overwhelm the api. After the 5 minutes any access to any of the attributes will refresh the information through a new call to the api.

```
print(toon.gas.average_daily)
print(toon.gas.average)
print(toon.gas.daily_cost)
print(toon.gas.daily_usage)
print(toon.gas.is_smart)
print(toon.gas.meter_reading)
print(toon.gas.value)
```

Print information about the electricity. Values are cached internally for 5 minutes so as to not overwhelm the api. After the 5 minutes any access to any of the attributes will refresh the information through a new call to the api.

```
print(toon.power.average_daily)
print(toon.power.average)
print(toon.power.daily_cost)
print(toon.power.daily_usage)
print(toon.power.is_smart)
print(toon.power.meter_reading)
print(toon.power.meter_reading_low)
print(toon.power.daily_usage_low)
print(toon.power.maximum)
print(toon.power.produced)
print(toon.power.solar)
print(toon.power.average_produced)
print(toon.power.meter_reading_low_produced)
print(toon.power.meter_reading_produced)
print(toon.power.daily_cost_produced)
print(toon.power.value)
```

Print information about connected hue lights.

```
# loop over all the lights
for light in toon.lights:
    print(light.is_connected)
    print(light.device_uuid)
    print(light.rgb_color)
    print(light.name)
    print(light.current_state)
    print(light.device_type)
    print(light.in_switch_all_group)
    print(light.in_switch_schedule)
    print(light.is_locked)
    print(light.zwave_index)
    print(light.zwave_uuid)

# or get a light by assigned name
light = toon.get_light_by_name('Kitchen Ceiling')

# print current status
print(light.status)

# checking whether the light can be toggled. For that to be able to
# happen the light needs to be connected and not locked.
```

```
# this state is checked internally from all the methods trying to toggle
# the switch state of the light
print(light.can_toggle)

# lights can be turned on, off or toggled
light.turn_on()
light.turn_off()
light.toggle()
```

Print information about connected fibaro smart plugs.

```
# get first smartplug
plug = toon.smartplugs[0]

# or get smartplug by assigned name
plug = toon.get_smartplug_by_name('Dryer')

# print all the information about the plug
print(plug.current_usage)
print(plug.current_state)
print(plug.average_usage)
print(plug.daily_usage)
print(plug.device_uuid)
print(plug.is_connected)
print(plug.name)
print(plug.network_health_state)
print(plug.device_type)
print(plug.in_switch_all_group)
print(plug.in_switch_schedule)
print(plug.is_locked)
print(plug.usage_capable)
print(plug.zwave_index)
print(plug.zwave_uuid)
print(plug.flow_graph_uuid)
print(plug.quantity_graph_uuid)


# print current status
print(plug.status)

# checking whether the plug can be toggled. For that to be able to
# happen the plug needs to be connected and not locked.
# this state is checked internally from all the methods trying to toggle
# the switch state of the plug
print(plug.can_toggle)

# plugs can be turned on, off or toggled
plug.turn_on()
plug.turn_off()
plug.toggle()
```

Get the current temperature

```
# show the current temperature
print(toon.temperature)
```

Work with thermostat states

```
# show the information about the current state
print(toon.thermostat_state.name)
print(toon.thermostat_state.id)
print(toon.thermostat_state.temperature)
print(toon.thermostat_state.dhw)

# set the current state by using a name out of ['comfort', 'home', 'sleep', away]
toon.thermostat_state = 'comfort' # Case does not matter. The actual
                                  # values can be overwritten on the
                                  # configuration.py dictionary.
```

Check out all the thermostat states configured

```
for state in toon.thermostat_states:
    print(state.name)
    print(state.id)
    print(state.temperature)
    print(state.dhw)
```

Work with the thermostat

```
# show current value of thermostat
print(toon.thermostat)

# manually assign temperature to thermostat. This will override the thermostat state
toon.thermostat = 20
```

The toon object exposes rrd measurement data in two forms, flow and graph and per interest item, gas, solar and for graph data type only, district_heat.

```
from pprint import pprint

# print flow data for gas
pprint(toon.data.flow.gas)

# print graph data for gas
pprint(toon.data.graph.gas)


# print flow data for power
pprint(toon.data.flow.power)

# print graph data for power
pprint(toon.data.graph.power)


# print flow data for solar
pprint(toon.data.flow.solar)

# print graph data for solar
pprint(toon.data.graph.solar)
```

CHAPTER 4

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

Submit Feedback

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

Get Started!

Ready to contribute? Here's how to set up *toonlib* for local development.

1. Clone your fork locally:

```
$ git clone https://github.com/costastf/toonlib.git
```

2. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your clone for local development:

```
$ mkvirtualenv toonlib
$ cd toonlib/
$ python setup.py develop
```

3. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

4. Commit your changes and push your branch to the server:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

5. Submit a merge request

CHAPTER 5

toonlib

toonlib package

Submodules

toonlib.configuration module

A place to store the configuration.

toonlib.helpers module

All helper objects will live here

class `toonlib.helpers.`**Agreement**(*id*, *checksum*, *city*, *display_common_name*, *display_hardware_version*, *display_software_version*, *heating_type*, *house_number*, *boiler_management*, *solar*, *toonly*, *post_code*, *street_name*)

Bases: `tuple`

boiler_management
Alias for field number 8

checksum
Alias for field number 1

city
Alias for field number 2

display_common_name
Alias for field number 3

display_hardware_version
Alias for field number 4

display_software_version
Alias for field number 5

heating_type
Alias for field number 6

house_number
Alias for field number 7

id
Alias for field number 0

post_code
Alias for field number 11

solar
Alias for field number 9

street_name
Alias for field number 12

toonly
Alias for field number 10

class toonlib.helpers.**Client**(*id*, *checksum*, *hash*, *sample*, *personal_details*)
Bases: tuple

checksum
Alias for field number 1

hash
Alias for field number 2

id
Alias for field number 0

personal_details
Alias for field number 4

sample
Alias for field number 3

class toonlib.helpers.**Data**(*toon_instance*)
Bases: object

Data object exposing flow and graph attributes.

class Flow(*toon_instance*)
Bases: object

The object that exposes the flow information of categories in toon

The information is rrd metrics and the object dynamically handles the accessing of attributes matching with the corresponding api endpoint if they are know, raises an exception if not.

class Data.**Graph**(*toon_instance*)
Bases: object

The object that exposes the graph information of categories in toon

The information is rrd metrics and the object dynamically handles the accessing of attributes matching with the corresponding api endpoint if they are know, raises an exception if not.

class toonlib.helpers.**Light**(*toon_instance*, *name*)
Bases: *toonlib.helpers.Switch*

Object modeling the hue light bulbs that toon can interact with.

It inherits from switch which is the common interface with the hue lamps to turn on, off or toggle

rgb_color

class toonlib.helpers.**Low**(*meter_reading_low*, *daily_usage_low*)
Bases: tuple

daily_usage_low
Alias for field number 1

meter_reading_low
Alias for field number 0

class toonlib.helpers.**PersonalDetails**(*number*, *email*, *first_name*, *last_name*, *middle_name*, *mobile_number*, *phone_number*)
Bases: tuple

email
Alias for field number 1

first_name
Alias for field number 2

last_name
Alias for field number 3

middle_name
Alias for field number 4

mobile_number
Alias for field number 5

number
Alias for field number 0

phone_number
Alias for field number 6

class toonlib.helpers.**PowerUsage**(*average_daily*, *average*, *daily_cost*, *daily_usage*, *is_smart*, *meter_reading*, *value*, *meter_reading_low*, *daily_usage_low*, *maximum*, *produced*, *solar*, *average_produced*, *meter_reading_low_produced*, *meter_reading_produced*, *daily_cost_produced*)
Bases: tuple

average
Alias for field number 1

average_daily
Alias for field number 0

average_produced
Alias for field number 12

daily_cost
Alias for field number 2

daily_cost_produced
Alias for field number 15

daily_usage
Alias for field number 3

daily_usage_low
Alias for field number 8

is_smart
Alias for field number 4

maximum
Alias for field number 9

meter_reading
Alias for field number 5

meter_reading_low
Alias for field number 7

meter_reading_low_produced
Alias for field number 13

meter_reading_produced
Alias for field number 14

produced
Alias for field number 10

solar
Alias for field number 11

value
Alias for field number 6

class toonlib.helpers.**SmartPlug**(*toon_instance*, *name*)
Bases: *toonlib.helpers.Switch*

Object modeling the fibaro smart plugs the toon can interact with.

It inherits from switch which is the common interface with the hue lamps to turn on, off or toggle

average_usage

current_usage

daily_usage

flow_graph_uuid

network_health_state

quantity_graph_uuid

usage_capable

class toonlib.helpers.**Solar**(*maximum*, *produced*, *solar*, *average_produced*, *meter_reading_low_produced*, *meter_reading_produced*, *daily_cost_produced*)
Bases: tuple

average_produced
Alias for field number 3

daily_cost_produced
Alias for field number 6

maximum
Alias for field number 0

meter_reading_low_produced
Alias for field number 4

meter_reading_produced
Alias for field number 5

produced
Alias for field number 1

solar
Alias for field number 2

class toonlib.helpers.**Switch**(*toon_instance*, *name*)
Bases: object

Core object to implement the turning on, off or toggle

Both hue lamps and fibaro plugs have a switch component that is shared. This implements that usage.

can_toggle

current_state

device_type

device_uuid

in_switch_all_group

in_switch_schedule

is_connected

is_locked

name

status

toggle()

turn_off()

turn_on()

zwave_index

zwave_uuid

class toonlib.helpers.**ThermostatInfo**(*active_state*, *boiler_connected*, *burner_info*, *current_displayed_temperature*, *current_modulation_level*, *current_set_point*, *current_temperature*, *error_found*, *have_ot_boiler*, *next_program*, *next_set_point*, *next_state*, *next_time*, *ot_communication_error*, *program_state*, *random_configuration_id*, *real_set_point*)
Bases: tuple

active_state
Alias for field number 0

boiler_connected
Alias for field number 1

burner_info
Alias for field number 2

current_displayed_temperature
Alias for field number 3

current_modulation_level
Alias for field number 4

current_set_point
Alias for field number 5

current_temperature
Alias for field number 6

error_found
Alias for field number 7

have_ot_boiler
Alias for field number 8

next_program
Alias for field number 9

next_set_point
Alias for field number 10

next_state
Alias for field number 11

next_time
Alias for field number 12

ot_communication_error
Alias for field number 13

program_state
Alias for field number 14

random_configuration_id
Alias for field number 15

real_set_point
Alias for field number 16

class toonlib.helpers.**ThermostatState**(*name*, *id*, *temperature*, *dhw*)
Bases: tuple

dhw
Alias for field number 3

id
Alias for field number 1

name
Alias for field number 0

temperature
Alias for field number 2

class toonlib.helpers.**Usage**(*average_daily*, *average*, *daily_cost*, *daily_usage*, *is_smart*, *meter_reading*, *value*)
Bases: tuple

average
Alias for field number 1

average_daily
Alias for field number 0

daily_cost
Alias for field number 2

daily_usage
Alias for field number 3

is_smart
Alias for field number 4

meter_reading
Alias for field number 5

value
Alias for field number 6

toonlib.toonlib module

A library overloading the api of the toon mobile app

class toonlib.toonlib.**Toon**(*username*, *password*, *state_retrieval_retry=3*)
Bases: object

Model of the toon smart meter from eneco.

gas
return – A gas object modeled as a named tuple

get_light_by_name(*name*)
Retrieves a light object by its name

Parameters **name** – The name of the light to return

Returns A light object

get_smartplug_by_name(*name*)
Retrieves a smartplug object by its name

Parameters **name** – The name of the smartplug to return

Returns A smartplug object

get_thermostat_state_by_id(*id_*)
Retrieves a thermostat state object by its id

Parameters **id** – The id of the thermostat state

Returns The thermostat state object

get_thermostat_state_by_name(*name*)
Retrieves a thermostat state object by its assigned name

Parameters **name** – The name of the thermostat state

Returns The thermostat state object

lights
return – A list of light objects modeled as named tuples

power
return – A power object modeled as a named tuple

smartplugs
return – A list of smartplug objects.

temperature
The current actual temperature as perceived by toon.

Returns A float of the current temperature

thermostat
The current setting of the thermostat as temperature

Returns A float of the current setting of the temperature of the thermostat

thermostat_info
return – A thermostatinfo object modeled as a named tuple

thermostat_state
The state of the thermostat programming

Returns A thermostat state object of the current setting

thermostat_states
return – A list of thermostatstate object modeled as named tuples

toonlib.toonlibexceptions module

Main module Exceptions file

Put your exception classes here

exception `toonlib.toonlibexceptions.`**IncompleteResponse**
Bases: `exceptions.Exception`

Vital information is missing from the response

exception `toonlib.toonlibexceptions.`**InvalidCredentials**
Bases: `exceptions.Exception`

The username and password combination was not accepted as valid

exception `toonlib.toonlibexceptions.`**InvalidThermostatState**
Bases: `exceptions.Exception`

Vital information is missing from the response

exception `toonlib.toonlibexceptions.`**UnableToGetSession**
Bases: `exceptions.Exception`

Could not refresh session

Module contents

toonlib package

CHAPTER 6

Credits

- Loosely based on the implementation found at https://github.com/rvdm/toon for the authentication part.

Development Lead

- Costas Tyfoxylos <costas.tyf@gmail.com>

Contributors

None yet. Why not be the first?

CHAPTER 7

History

CHAPTER 8

0.1 (13-04-2017)

- First release on pypi

CHAPTER 9

0.2 (25-04-2017)

- Added support for turning on, off and toggling lights and smartplugs

CHAPTER 10

0.3 (26-04-2017)

- Extended the info of lights and smartplugs. Added support for identification of locked state for switching them.

CHAPTER 11

Indices and tables

- genindex
- modindex
- search

Python Module Index

t

Index

U

V

Z