curb_energy Documentation

Release 0.0.2

Gino Ledesma

Apr 16, 2017

Contents

A Python library to interact with the Curb API built on top of asyncio and aiohttp.

Documentation: http://curb-energy.readthedocs.io/en/latest/

CHAPTER 1

Disclaimer

This project is not affiliated with Curb Inc.. Curb maintains a github repository of various projects and documents their API, which is built upon HAL.

I wanted something more pythonic than using HAL-tools to consume the API, and it was also a good opportunity for experimenting with using asyncio and aiohttp for handling streaming data.

CHAPTER 2

Requirements

curb_energy requires Python 3.5 or later, mostly due to the async and type hint syntax used in the library.

CHAPTER 3

Installation

curb_energy can be installed using `pip`, `easy_install` or `setup.py`

```
pip install curb_energy
```

You may want to install the library in a virtual environment to test things out.

CHAPTER 4

License

curb_energy is offered under the Apache License 2.0.

Getting Started

To connect to the Curb REST API, you can instantiate a *curb_energy.client.RestApiClient* object passing in the required credentials.

OAuth Tokens

Before you can interact with the Curb REST API, you'll need to get a client token and secret for your specific application. For that, you will need to reach out to the Curb support team asking them for developer access to their API. The defaults have been set to `CHANGE_ME` explicitly for this reason – you will be unable to proceed until getting these.

Once you have been given a client ID and token for your app, you can pass this to the *curb_energy.client.RestApiClient* along with the username and password (or existing OAuth2 access/refresh token) to interact with the API.

> **Warning:** **Just to be clear**: the Client Token and Client Secret are DIFFERENT from the username, password, or access and refresh tokens. The Client Token/Secret is used to identify the application interacting with the REST API, and the access and refresh tokens are used to identify the user.

The library will automatically fetch an access token when authenticating with a username and password. Subsequent requests to the REST API will use the access token. As well, the refresh token is automatically used to request a new access token when the previous access token has expired.

Runtime

The client can be used as a context manager to automatically handle logging in and cleaning-up:

```
import asyncio
from curb_energy.client import RestApiClient

async def main():
    async with RestApiClient(username='user',
                             password='pass',
                             client_id='APP_CLIENT_ID',
                             client_token='APP_CLIENT_TOKEN') as client:
        profiles = await client.profiles()
        devices = await client.devices()

    for profile in profiles:
        print(profile)

    for device in devices:
        print(device)

asyncio.get_event_loop().run_until_complete(main())
```

Or in a more traditional way:

```
import asyncio
from curb_energy.client import RestApiClient

async def main():
    client = RestApiClient(username='user', password='pass',
                           client_id='APP_CLIENT_ID',
                           client_token='APP_CLIENT_TOKEN')
    try:
        await client.authenticate()
        profiles = await client.profiles()
        devices = await client.devices()

        for profile in profiles:
            print(profile)

        for device in devices:
            print(device)
    finally:
        # Clean-up
        await client.session.close()

asyncio.get_event_loop().run_until_complete(main())
```

API Documentation

curb_energy.client

Client module for interacting with the Curb REST and Real-Time APIs

class curb_energy.client.**AuthToken**(*access_token=None*, *refresh_token=None*, *expires_in=0*, *user_id=0*, *token_type='bearer'*)

Curb API OAuth2 Token. For more information, refer to: <https://oauth.net/articles/authentication/>

__init__(*access_token=None*, *refresh_token=None*, *expires_in=0*, *user_id=0*, *token_type='bearer'*)

Create an *AuthToken* instance.

Parameters

- **access_token** (Optional[str]) – The access token
- **refresh_token** (Optional[str]) – The refresh token
- **expires_in** (int) – The time, in seconds, this token is valid for.
- **user_id** (int) – The unique ID of the user associated with this token
- **token_type** (str) – Bearer type

expiry
The expiration date of the token (in UTC)

Return type datetime

Returns Expiry date (in UTC)

Return type datetime.datetime

static from_json()
Creates an AuthToken object from the given JSON payload.

Parameters **data** (str) – Token data

Return type *AuthToken*

Returns Creates an AuthToken instance from the given payload

Raises ValueError

json()
Serializes the AuthToken into JSON

Return type str

Returns JSON version of the AuthToken

class curb_energy.client.**RestApiClient**(*loop=None*, *username=None*, *password=None*, *auth_token=None*, *api_url='https://app.energycurb.com'*, *client_token='CHANGE_ME'*, *client_secret='CHANGE_ME'*, *ssl_context=None*)
A client for the Curb REST API

__init__(*loop=None*, *username=None*, *password=None*, *auth_token=None*, *api_url='https://app.energycurb.com'*, *client_token='CHANGE_ME'*, *client_secret='CHANGE_ME'*, *ssl_context=None*)
Initialize the REST API client.

The Curb API uses Oauth2 authentication. An access token can be fetched by supplying a valid username and password as credentials. Subsequent authentication with the API will be done using the Oauth2 token in the form of *AuthToken*.

You can also pass an existing token instead of a username/password.

Parameters

- **username** (Optional[str]) – Username
- **password** (Optional[str]) – Password
- **auth_token** (Optional[*AuthToken*]) – Oauth2 client token
- **api_url** (str) – The URL to the Curb REST API
- **client_token** (str) – The application client token (app identifier)

- **client_secret** (str) – The application client secret (app password)
- **ssl_context** (Optional[SSLContext]) – Optional SSL

> **Warning:** As a client to the Curb REST API, you **MUST** provide your own client_token and client_secret which identifies the *APPLICATION* you are developing. This is separate from the username/password or access token that identifies the *USER* accessing their data.
>
> See <http://docs.energycurb.com/authentication.html> for more info. You'll need to contact the Curb Support Team at <http://energycurb.com/support/> for assistance.

Example:

```
async with RestApiClient(username=user,
                         password=pass,
                         client_token='CHANGE_ME',
                         client_secret='CHANGE_ME') as client:
    profiles = await client.profiles()
    devices = await client.devices()

    for profile in profiles:
        print(profile)

    for device in devices:
        print(device)
```

Or more traditionally:

```
client = RestApiClient(username=user,
                       password=pass,
                       client_token='CHANGE_ME',
                       client_secret='CHANGE_ME')
try:
    # Fetch and set the access token
    await client.authenticate()

    # code goes here

finally:
    await client.session.close()
```

auth_token
The AuthToken associated with the REST API session

Return type *AuthToken*

Returns The AuthToken associated with the session

authenticate()
Authenticates with the REST API by fetching an access token, raising an exception on failure. The access token is stored as a property. This method is automatically called when the client is used as a context manager.

Return type *AuthToken*

Returns The authentication token

Raises *CurbBaseException*

devices()
Return a list of devices associated with the authenticated user

Return type List[*Device*]

Returns A list of devices

entry_point()
Return the resources the authenticated user has access to, namely Profiles and Devices. This is automatically called when the client is used as a context manager.

Return type Dict[~KT, ~VT]

Returns a dict of links to the Profiles and Devices

fetch_access_token(*client_token=None*, *client_secret=None*, *username=None*, *password=None*)
Fetches an access token using the given credentials. The supplied parameters override the original credentials passed to instance of the REST API client.

Parameters

- **client_token** (Optional[str]) – The OAuth Client Token (app identifier)
- **client_secret** (Optional[str]) – The OAuth Client Secret (app password)
- **username** (Optional[str]) – The username to authenticate with
- **password** (Optional[str]) – The password to authenticate with

Return type Optional[*AuthToken*]

Returns Returns the access token after authentication

historical_data(*profile_id=0*, *granularity='1H'*, *unit='$/hr'*, *since=0*, *until=None*)
Return all recorded measurements for the given profile.

Parameters

- **profile_id** (int) – The profile configuration
- **granularity** (str) – Per Minute, Per Hour (default), or Per Day
- **unit** (str) – Dollars Per Hour, or Watts (default)
- **since** (int) – Start time of measurements (in epoch format). Use 0 to indicate the beginning, which is the default.
- **until** (Optional[int]) – End time of measurements (in epoch format)

Return type *Measurement*

profiles()
Return a list of profiles associated with the authenticated user.

Return type List[*Profile*]

Returns A list of profiles

refresh_access_token()
Get a new access token using an existing refresh token and associated user ID. All other access tokens are immediately invalidated. When calling this method, *auth_token* is automatically set to the new token.

Return type *AuthToken*

Returns a new access token

Raises *CurbBaseException* when *auth_token* is not set

class curb_energy.client.**RealTimeClient**(*config*, *driver=<class 'hbmqtt.client.MQTTClient'>*)
A client to the Curb Energy Real-Time Streaming API

Todo

Refactor to support different access mechanisms. For now, we're limited to using MQTT over WebSockets

---

__init__(*config*, *driver=<class 'hbmqtt.client.MQTTClient'>*)
Create an instance of *RealTimeClient*.

Parameters

- **config** (*RealTimeConfig*) – Real Time Config
- **driver** (Callable) – Real Time client driver

Example:

```
client = RealTimeClient(config)
await client.connect()
while condition:
    data = await client.stream()
await client.disconnect()
```

Used as a context manager:

```
async with RealTimeClient(config) as client:
    while condition:
        data = await client.stream()
```

config
Returns the configuration parameters for the client.

Return type *RealTimeConfig*

connect()
Connect to the Real-time API

disconnect()
Disconnect from the Real-time API

is_connected
Returns True if the real-time client has successfully established a connection with the Real-Time API, False otherwise.

Return type bool

read()
Returns a single stream from the real-time API, or None when an error occurs. This may raise a ValueError when the returned data is invalid JSON.

Return type *RealTimeMessage*

Returns Returns measurements

Raises ValueError

class curb_energy.client.**RealTimeMessage**(*timestamp*, *measurements*)

measurements
Alias for field number 1

timestamp
Alias for field number 0

curb_energy.errors

Errors and Exceptions module

exception curb_energy.errors.**CurbBaseException**
Base Exception for the Curb Energy library

curb_energy.models

Classes for representing the Curb API resources

class curb_energy.models.**Billing**(*profile_id=-1*, *billing_model=None*, *day_of_month=1*, *zip_code=None*, *dollar_per_kwh=None*, ***kwargs*)
Billing describes how and when the customer is billed and is associated with a *BillingModel* instance.

__init__(*profile_id=-1*, *billing_model=None*, *day_of_month=1*, *zip_code=None*, *dollar_per_kwh=None*, ***kwargs*)
The billing configuration for the customer

Parameters

- **profile_id** (int) – The Curb configuration profile
- **billing_model** (Optional[*BillingModel*]) – Billing model information
- **day_of_month** (int) – The start day of the billing period
- **zip_code** (Optional[int]) – The zip code of the dwelling being monitored
- **dollar_per_kwh** (Optional[float]) – The price per kilowatt-hour

class curb_energy.models.**BillingModel**(*sector='Residential'*, *label=None*, *utility=None*, *name=None*, ***kwargs*)
The Billing Model describes the utility and billing tier for a given customer.

__init__(*sector='Residential'*, *label=None*, *utility=None*, *name=None*, ***kwargs*)
Create an instance of the billing model for the customer

Parameters

- **sector** (str) – One of 'Residental' or 'Commercial'
- **label** (Optional[str]) – Unique ID of this instance
- **utility** (Optional[str]) – The name of the utility / power provider
- **name** (Optional[str]) – The billing tier

class curb_energy.models.**Device**(*id=-1*, *building_type=None*, *name=None*, *timezone=None*, *sensor_groups=None*, ***kwargs*)
A logical grouping of Sensor Groups. A "device" can be thought of as a unit representing a location being measured, such as a home.

Todo

Clarify with Curb what they really intend by this.

__init__(*id=-1*, *building_type=None*, *name=None*, *timezone=None*, *sensor_groups=None*, ***kwargs*)

Creates an instance of a dwelling location

Parameters

- **id** (int) – The unique ID of this monitored unit
- **building_type** (Optional[str]) – The type of building (home, commercial)
- **name** (Optional[str]) – The name of the monitored unit
- **timezone** (Optional[str]) – Timezone label
- **sensor_groups** (Optional[List[*SensorGroup*]]) – List of sensor groups associated

class curb_energy.models.**Measurement**(*granularity*, *since*, *until*, *unit*, *headers*, *data*)

data

Alias for field number 5

granularity

Alias for field number 0

headers

Alias for field number 4

since

Alias for field number 1

unit

Alias for field number 3

until

Alias for field number 2

class curb_energy.models.**Profile**(*id=-1*, *display_name=None*, *real_time=None*, *register_groups=None*, *registers=None*, *widgets=None*, *billing=None*, ***kwargs*)

A profile defines how to interpret data, access real time data, and various other configuration options.

__init__(*id=-1*, *display_name=None*, *real_time=None*, *register_groups=None*, *registers=None*, *widgets=None*, *billing=None*, ***kwargs*)

Create an instance of a configuration profile

Parameters

- **id** (int) – The unique ID of the profile
- **display_name** (Optional[str]) – The friendly name of this profile/configuration
- **real_time** (Optional[*RealTimeConfig*]) – The configuration for the real-time API
- **register_groups** (Optional[List[*RegisterGroup*]]) – The register groups associated with this config
- **registers** (Optional[List[*Register*]]) – The list of registers associated with this config
- **widgets** (Optional[List[type]]) – The list of widgets
- **billing** (Optional[*Billing*]) – The billing configuration

find_register(*id*)
Return a Register by its unique ID, or `None` if not found

Parameters **id** (`str`) – The unique ID of the register to look up

Return type `Optional[Register]`

class curb_energy.models.**RealTimeConfig**(*topic=None*, *format='curb'*, *prefix=None*, *ws_url=None*, ***kwargs*)
Configuration for the Real-Time client

__init__(*topic=None*, *format='curb'*, *prefix=None*, *ws_url=None*, ***kwargs*)
Create an instance of the RealTime configuration object

Parameters

- **topic** (`Optional[str]`) – The MQTT topic to subscribe to
- **format** (`str`) – Output format (currently only accepts 'curb')
- **prefix** (`Optional[str]`) – A prefix for each key within the measurement results
- **ws_url** (`Optional[str]`) – The URL to the real-time API

class curb_energy.models.**Register**(*id=''*, *multiplier=1*, *flip_domain=False*, *label=None*, ***kwargs*)
A source of power measurement data.

__init__(*id=''*, *multiplier=1*, *flip_domain=False*, *label=None*, ***kwargs*)
Creates an instance of a source of power measurement data, such as an individual circuit breaker of electic panel

Parameters

- **id** (`str`) – Unique identifier
- **multiplier** (`int`) – Power multiplier
- **flip_domain** (`bool`) – Invert the sign of the reported values (pos/neg)
- **label** (`Optional[str]`) – Name of the power source

class curb_energy.models.**RegisterGroup**(*grid*, *normals*, *solar*, *use*)
A logical grouping of registers according to classification

__init__(*grid*, *normals*, *solar*, *use*)
A group of registers

Parameters

- **grid** (`Optional[List[Register]]`) – Circuit breakers from the grid
- **normals** (`Optional[List[Register]]`) – "Normal" (non-grid) circuit breakers
- **solar** (`Optional[List[Register]]`) – Circuit breakers from solar power
- **use** (`Optional[List[Register]]`) – Used power

class curb_energy.models.**Sensor**(*id=-1*, *name=None*, *arbitrary_name=None*, ***kwargs*)
An energy monitoring device (in this case, the Curb Hub)

__init__(*id=-1*, *name=None*, *arbitrary_name=None*, ***kwargs*)
Creates an instance of a Sensor

Parameters

- **id** (`int`) – Unique identifier

- **name** (Optional[str]) – Unique name (serial number) of the Curb Hub
- **arbitrary_name** (Optional[str]) – User-assigned name for the Curb Hub

class curb_energy.models.**SensorGroup**(*id=-1*, *sensors=None*, ***kwargs*)
A logical grouping of sensors

__init__(*id=-1*, *sensors=None*, ***kwargs*)
Creates a logical grouping of sensors identified by a unique ID

Parameters

- **id** (int) – The unique ID of the sensor group
- **sensors** (Optional[List[*Sensor*]]) – List of sensors associated with this group

curb_energy.schema

The schema module helps convert the Curb API REST resources into Python-friendly objects.

class curb_energy.schema.**BillingModelSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Billing Model: Utility/Provider information

class curb_energy.schema.**BillingSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Billing Information for the monitored location

class curb_energy.schema.**DeviceSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
A monitored "location", such as a home/building.

Todo

Why does Curb API call it "Device"?

---

class curb_energy.schema.**ProfileSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Profiles define how to interpret data, access real time data, and various other configuration options.

class curb_energy.schema.**RealTimeSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Source for Real-time data

class curb_energy.schema.**RegisterGroupsSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
A logical grouping of Registers.

class curb_energy.schema.**RegisterSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Source for a single stream of power data. They can correspond to a physical circuit breaker.

class curb_energy.schema.**RegistersSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

A Collection of Registers

class curb_energy.schema.**SensorGroupSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

A group of one or more Sensors measuring power at a common location

class curb_energy.schema.**SensorSchema**(*extra=None*, *only=()*, *exclude=()*, *prefix=''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)

An energy measuring device, like the Curb Hub

ChangeLog

0.0.2 (2017-04-15)

Minor Enhancements + Doc Updates

Changes

- Documentation updated to emphasize the need for supplying a client token and secret when using the library (in addition to the username and password credentials for interacting on behalf of a user)

Features

- Access tokens are automatically refreshed when they expire

0.0.1 (2017-04-14)

Initial Version

Features

- Read access to the Curb REST API
- Read access to the Curb Streaming API using MQTT over WebSockets

Glossary

auth token An OAuth2 token used to authenticate with the Curb REST API. The token may be either an access token or refresh token. Tokens should be treated like passwords – keep them safe and secure!

profile A set of configuration options that collectively defines how to interpret data, and access real time data.

register A source of power measurement data, such as a circuit breaker.

register groups A collection of registers, logically grouped for a specific purpose, such as "use", "mains", "solar", or "grid".

sensor An energy monitoring device, which in this case is the Curb Hub. A sensor will have one or more registers associated with it.

sensor group A collection of sensor (Curb) devices, grouped in a logical manner, such as "Main Panel". A given sensor will usually have at most 18 registers, and multiple sensors may be needed to cover numerous circuit breakers/electric panels in a given location.

Python Module Index

C

Index

Symbols

A

B

C

D

E

F

G

H

I

J

M

P

R

S

T

U