Trace Python Client Documentation

Release 0.1.2

Diego Fernandez

Jul 24, 2018

Contents:

CHAPTER 1

Trace Python Client

Python client for interacting with the Trace api. For information on the API see the api documentation

- Free software: MIT license
- Documentation: https://py-trace.readthedocs.io.

1.1 Getting Started

To use in REPL or a non-GUI app:

```
from py_trace import Trace

client = Trace(<client_key>, <client_secret>)
client.get_authorization_url()
# output should be like https://www.alpinereplay.com/api/oauth_login?oauth_token=
↪<token>, go to the url and authorize the app
# after authorization, you should see a url like http://snow.traceup.com/api/oauth_
↪login?oauth_token=<token>&oauth_verifier=<verifier>
client.get_access_token(<verifier>)
client.get_user()
```

To use in a web app (using a very basic Flask example):

```
from flask import Flask, request, redirect
import json

app = Flask(__name__)
trace = Trace(<client_key>, <client_secret>, 'http://127.0.0.1:5000/auth_callback')

@app.route('/auth_callback')
def handle_redirect():
    trace.get_access_token(request.args['oauth_verifier'])
    return redirect('/user')
```

(continues on next page)

(continued from previous page)

```
@app.route('/auth')
def get_user_token():
    url = trace.get_auth_url()
    return redirect(url)

@app.route('/user')
def get_user():
    return json.dumps(trace.get_user())
```

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Trace Python Client, run this command in your terminal:

```
$ pip install py_trace
```

This is the preferred method to install Trace Python Client, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Trace Python Client can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/aiguofer/py-trace
```

Or download the tarball:

```
$ curl  -OL https://github.com/aiguofer/py-trace/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Modules

3.1 py_trace package

3.1.1 Module contents

Top-level package for Trace Python Client.

class py_trace.**Trace**(*client_key*, *client_secret*, *callback_uri=None*, *access_token=None*, *sport='snow'*)

api_request(*path*, *method='GET'*, ***kwargs*)
Make a request to the API and parse the results

Parameters

- **path** (*str*) – API endpoint to hit, must start with /
- **method** (*str*) – HTTP method (Default value = 'GET')
- ****kwargs** – Arguments to pass to the request, see `requests.request`

authenticate(*access_token*)
Step 4 of auth, authenticate the user by creating a session with the access token. This is optional as it is called by get_access_token. If you already have the access_token, you can call this function directly or pass the access_token when instanciating the class in order to authenticate.

Parameters **access_token** (*dict*) – A dict containing the oauth_token and oauth_token_secret.

create_event_comment(*event_id*, *message*)
Add a comment to an event

Parameters

- **event_id** (*int, str*) – Event ID
- **message** (*str*) – Comment text

create_visit_comment(*visit_id*, *facebook=False*, *twitter=False*, *photo=None*, *hide_resort_name=0*, *comment=None*, *equipment=None*)
Add comment to a visit

Parameters

- **visit_id** (*int, str*) – Visit ID
- **facebook** (*bool*) – Whether to post to facebook or not (Default value = False)
- **twitter** (*bool*) – Whether to post to twitter or not (Default value = False)
- **photo** – (Default value = None)
- **hide_resort_name** – (Default value = 0)
- **comment** (*str*) – Comment text (Default value = None)
- **equipment** (*list*) – List of strings for each equipment item (Default value = None)

create_visit_equipment(*visit_id*, *equipment*)
Add equipment info for a visit

Parameters

- **visit_id** (*int, str*) – Visit ID
- **equipment** (*list*) – List of strings for each equipment item

create_visit_export(*visit_id*)
Create an export of the visit gpx file

Parameters **visit_id** (*int, str*) – Visit ID

create_visit_overlay(*data*)
Upload track file to create video overlay

Parameters **data** –

create_visit_photo(*visit_id*, *photo*)
Add a photo to a visit

Parameters

- **visit_id** (*int, str*) – Visit ID
- **photo** –

delete_run(*run_id*)
Delete a run

Parameters **run_id** (*int, str*) – Run ID

get_access_token(*oauth_verifier*)
Step 3 of auth, get the access token. After the user authorizes the app, we get the access token using the oauth_verifier and then authenticate the user.

Parameters **oauth_verifier** (*str*) – A string containing the verifier

get_authorization_url()
Step 2 of auth, get the authorization URL. It will first get the request token if it needs it.

get_event(*event_id*)
Get event info

Parameters **event_id** (*int, str*) – Event ID

get_events(*min_timestamp=None*, *max_timestamp=None*, *limit=None*, *ffilter=None*)
Get all events for yourself

Parameters

- **min_timestamp** (*int*) – Minimum timestamp to filter results (Default value = None)
- **max_timestamp** (*int*) – Maximum timestamp to filter results (Default value = None)
- **limit** (*str, int*) – Limit the number of entries in responses, if None is given the API defaults to 50 (Default value = None)
- **ffilter** (*dict*) – A filter for fields to be returned, see documentation for more (Default value = None)

get_request_token()
Step 1 of auth, get the request token. This is optional as this is called by get_auth_url if needed.

get_user(*user_id='self'*)
Get user info

Parameters **user_id** (*int, str*) – User ID (Default value = 'self')

get_visit_equipment(*visit_id*)
Get equipment info for a visit

Parameters **visit_id** (*int, str*) – Visit ID

get_visit_events(*visit_id*, *ffilter=None*)
Get all events for a visit

Parameters

- **visit_id** (*int, str*) – Visit ID
- **ffilter** (*dict*) – A filter for fields to be returned, see documentation for more (Default value = None)

get_visit_list(*user_id='self'*, *ffilter=None*)
Get list of user visits

Parameters

- **user_id** (*int, str*) – User ID (Default value = 'self')
- **ffilter** (*dict*) – A filter for fields to be returned, see documentation for more (Default value = None)

get_visit_weather(*visit_id*)
Get weather report for visit if available

Parameters **visit_id** (*int, str*) – Visit ID

get_visits(*user_id='self'*, *limit=None*, *min_timestamp=None*, *max_timestamp=None*, *visit_ids=None*, *ffilter=None*)
Get all user visits

Parameters

- **user_id** (*int, str*) – User ID (Default value = 'self')
- **limit** (*str, int*) – Limit the number of entries in responses, if None is given the API defaults to 50 (Default value = None)
- **min_timestamp** (*int*) – Minimum timestamp to filter results (Default value = None)
- **max_timestamp** (*int*) – Maximum timestamp to filter results (Default value = None)

- **visit_ids** (*list*) – List of visit IDs to include (Default value = None)
- **ffilter** (*dict*) – A filter for fields to be returned, see documentation for more (Default value = None)

hide_run(*run_id*)
Hide a run

Parameters **run_id** (*int, str*) – Run ID

like_event(*event_id*)
Like an event

Parameters **event_id** (*int, str*) – Event ID

share_visit(*visit_id*, *stats*, *comment=None*, *photo=None*, *facebook=True*, *twitter=True*)
Share a visit

Parameters

- **visit_id** (*int, str*) – Visit ID
- **stats** (*list*) – List of stats to include, one of [total_distance, jumps, air_time, avg_speed, calories, vertical_drop, sustained_speed, slope_time, max_slope, turns_num, longest_ride]
- **comment** (*str*) – Comment text (Default value = None)
- **photo** – (Default value = None)
- **facebook** (*bool*) – Whether to post to facebook or not (Default value = True)
- **twitter** (*bool*) – Whether to post to twitter or not (Default value = True)

unlike_event(*event_id*)
Unlike an event

Parameters **event_id** (*int, str*) – Event ID

CHAPTER 4

Contributing

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

You can contribute in many ways:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/aiguofer/py-trace/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

4.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.1.4 Write Documentation

Trace Python Client could always use more documentation, whether as part of the official Trace Python Client docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/aiguofer/py-trace/issues.

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.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.2 Get Started!

Ready to contribute? Here's how to set up *py-trace* for local development.

1. Fork the *py-trace* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:aiguofer/py-trace.git
```

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

```
$ mkvirtualenv py-trace
$ cd py-trace/
$ python setup.py develop
```

4. Create a branch for local development:

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

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 py_trace tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

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

7. Submit a pull request through the GitHub website.

4.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.7, 3.4, 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/aiguofer/py-trace/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ py.test tests.test_py_trace
```

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Diego Fernandez <aiguo.fernandez@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2018-07-22)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index