signupto Documentation

Release 0.1

Luke Plant

April 16, 2015

Contents

Contents:

CHAPTER 1

signupto

NOTICE: This project is unmaintained. If you want to take over maintenance, please get in contact with me.

Minimalist client library for the sign-up.to HTTP API - http://sign-up.to

- Free software: BSD license
- Home page: https://bitbucket.org/spookylukey/signupto
- Bugs: https://bitbucket.org/spookylukey/signupto/issues?status=new&status=open
- Docs: http://signupto.readthedocs.org/en/latest/

1.1 Status

This is beta software. It is being used in production, and covers the complete sign-up.to HTTP API. There may still be improvements and fixes to be made.

If you use this library, let me know (at L.Plant.98@cantab.net), and I can consult you about improvements. Otherwise I may change things without notice.

There are also no automated tests yet. (Patches welcome!)

CHAPTER 2

Installation

At the command line:

```
$ easy_install signupto
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv signupto
$ pip install signupto
```

CHAPTER 3

Usage

These docs assume familiarity with the API docs for sign-up.to, which provide most of the details for endpoints, parameters etc.

Quickstart:

```
>>> from signupto import Client, HashAuthorization
>>> c = Client(auth=HashAuthorization(company_id=1234, user_id=4567,
...                                   api_key='e4cf7fe3b764a18c04f6792c09e3325d'))
>>> c.subscription.get(list_id=7890).data

[{u'cdate': 1374769049,
  u'confirmationredirect': u'',
  u'confirmed': True,
  u'id': 36154421,
  u'list_id': 7890,
  u'mdate': 1374769049,
  u'source': u'import',
  u'subscriber_id': 9180894},
 {u'cdate': 1374769049,
  u'confirmationredirect': u'',
  u'confirmed': True,
  u'id': 13654428,
  u'list_id': 7890,
  u'mdate': 1374769040,
  u'source': u'import',
  u'subscriber_id': 9186895}]
```

3.1 Authorization

Hash authorization:

```
>>> from signupto import Client, HashAuthorization
>>> c = Client(auth=HashAuthorization(company_id=1234, user_id=4567,
...                                   api_key='e4cf7fe3b764a18c04f6792c09e3325d'))
```

Token authorization:

```
>>> from signupto import Client, TokenAuthorization
>>> c = Client(auth=TokenAuthorization(username='joe', password='my_secret'))
```

This will do an unauthenticated API call to get the token, which will be used in subsequent API calls. If you want to get the actual token returned, along with the expiry timestamp, for storage and re-use, they can be found as attributes on the `TokenAuthorization` instance, after it has been passed to the `Client` constructor:

```
>>> token_auth = TokenAuthorization(username='joe', password='my_secret')
>>> c = Client(auth=token_auth)
>>> token, expiry = token_auth.token, token_auth.expiry
```

And then to re-use later:

```
>>> token_auth_2 = TokenAuthorization()
>>> token_auth_2.token = token
>>> token_auth_2.initialized = True
>>> c = Client(auth=token_auth_2)
```

3.2 API calls

`Client` instances have attributes representing all the resources/endpoints. The spelling of the attribute is the same as the path for the endpoint e.g. `subscription`, `clickAutomation`.

class `signupto.client.`**Endpoint**

get(***kwargs*)

post(***kwargs*)

put(***kwargs*)

delete(***kwargs*)

Each endpoint then has methods for the HTTP verbs: get(), post(), put(), delete() and head().

Parameters to the endpoint are passed as keyword arguments to these methods.

Endpoints and their parameters are described in the sign-up.to docs here: https://dev.sign-up.to/documentation/reference/latest/endpoints/

These methods return a `SignuptoResponse` object, which contains the 'response' attribute of the API call, that is, an object with these attributes:

- `data` - the data returned by the API call, converted to native Python objects e.g. a Python dictionary containing list information, or an array. The `signupto` library does not convert the data beyond converting into native Python types.
- `next` - value representing the resource following the last returned resource.
- `count` - the number of resources returned.

Usually you will just need the `data` attribute. See https://dev.sign-up.to/documentation/reference/latest/making-requests/response-format/ for more information.

Example:

```
>>> c.subscription.post(list_id=1234, subscriber_id=4567)

SignuptoResponse(data={u'confirmed': False, u'mdate': 1384265219,
                       u'confirmationredirect': u'', u'subscriber_id': 4567,
                       u'source': u'api', u'cdate': 1384265219, u'list_id': 1234,
                       u'id': 19486109}, next=None, count=1)
```

head(***kwargs*)

The `head()` method works similarly to the other methods. However as there is no response dictionary for `HEAD` verbs, the `head()` method does not return a `SignuptoResponse`, but instead returns None. It will raise an error like the other calls for HTTP codes in 4XX range.

3.3 Errors

Errors returned by the server in the 5XX range will raise `signupto.ServerError`.

Errors returned by the server in the 4XX range will raise `signupto.ClientError`. For example:

```
>>> c = Client(auth=HashAuthorization(company_id=1234, user_id=4567,
...                                   api_key='oops'))
>>> c.list.get()

ClientError: {u'message': u'Bad signature:
9ec621a4c27dcb28bdb2148f1475f990f7adfbd6', u'code': 401, u'subcode': None,
u'additional_information': u'GET /v0/list\\r\\nDate: Tue, 12 Nov 2013
14:24:58 GMT\\r\\nX-SuT-CID: 28711\\r\\nX-SuT-UID: 6235\\r\\nX-SuT-Nonce:
h91eq7qwh43go4tpu20hiaira2mfuu7yqf8e4t'}
```

The dictionary is stored on the exception object in the attribute `error_info`.

When the error has code 404, indicating something not found, a subclass of `ClientError`, `ObjectNotFound`, is used instead. This can be especially useful when you are applying filters such that there are no matching objects, which is often not an error condition for your application, so needs to be handled differently:

```
from signupto import ObjectNotFound

try:
    unconfirmed = c.subscription.get(list_id=1234, confirmed=False).data
except ObjectNotFound:
    unconfirmed = []
```

Alternatively you can use the convenience methods below:

3.4 Convenience methods

class `signupto.client.`**Endpoint**

get_list(***kwargs*)

This is similar to the `get()` method, except that it will catch 404 errors, and convert them to an empty list. As a consequence, it never returns a full `SignuptoResponse` object, but just the data (i.e. `SignuptoResponse.data` or an empty list).

get_all(***kwargs*)

This is similar to `get_list()`, but it will repeatedly follow the `next` parameter in order to get the full list of items.

delete_any(***kwargs*)

This is similar to `delete()`, but will catch 404 error, so that you do not get an error if you delete something that doesn't exist. As a consequence, the return value is just the `data` attribute.

```
>>> c.list.delete_any(id=1234)
[{u'id': 1234}]
>>> c.list.delete_any(id=1234)
[]
```

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://bitbucket.org/spookylukey/signupto/issues.

If you are reporting a bug, please include:

- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

4.1.2 Fix bugs and implement features

Look through the BitBucket issues for bugs or features that you want to tackle.

4.1.3 Write tests

There are no tests yet, and they are a bit tricky to do, as you really have to test a client library like this against the actual web service, which has no sandbox. Testing strategies will probably depend on how well `requests` can be mocked out.

4.1.4 Write documentation

signupto could always use more documentation, whether as part of the official signupto 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://bitbucket.org/spookylukey/signupto/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 *signupto* for local development.

1. Fork the *signupto* repo on BitBucket.
2. Clone your fork locally:

   ```
   $ hg clone ssh://hg@bitbucket.org/your_name_here/signupto
   ```

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 signupto
   $ cd signupto/
   $ python setup.py develop
   ```

4. Create a branch for local development:

   ```
   $ hg branch 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 the tests, including testing other Python versions with tox:

   ```
   $ tox
   ```

   You will need to:

   ```
   $ pip install tox
   ```

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

   ```
   $ hg record
   $ hg push
   ```

7. Submit a pull request through the BitBucket website.

CHAPTER 5

Credits

5.1 Development Lead

- Luke Plant <L.Plant.98@cantab.net>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1 (2013-11-21)

- Cleanups

6.2 0.0.6 (2013-11-14)

- Added convenience methods `get_list`, `get_all` and `delete_any`.

6.3 0.0.5 (2013-11-13)

- Python 3 fixes

6.4 0.0.4 (2013-11-13)

- Rewrote to remove drest dependency. It's simpler and shorter now.

6.5 0.0.3 (2013-11-12)

- Fixed put methods

6.6 0.0.2 (2013-11-12)

- Fixed delete methods

6.7 0.0.1 (2013-11-12)

- First release on PyPI.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*

S

D

E

G

H

P

S