SmokeAPI Documentation

Release 0.2.0

Andrew Wegner

Feb 19, 2018

Contents

Release v0.2.0. (*Installation*)

SmokeAPI is a simple Python wrapper for the MetaSmoke API .

Retrieving data from the API is simple:

```
from smokeapi import SmokeAPI
SMOKE = SmokeAPI('your_api_key')
posts = SMOKE.fetch('posts/feedback', type="naa-")
```

The above, will issue a call to the `Posts Feedback`. end point on MetaSmoke.

CHAPTER 1

Supported Features

- Read and write functionality via the API.
- Retrieve multiple pages of results with a single call and merge all the results into a single response.
- Throw exceptions returned by the API for easier troubleshooting.
- Utilize Requests.

SmokeAPI is supported on Python 2.7 - 3.5.

CHAPTER 2

User Guide

This portion of documentation provides details on how to utilize the library, and provides advanced examples of various use cases.

2.1 Introduction

SmokeAPI was written to interact with the MetaSmoke project and make my life a bit easier than messing with `curl` or `urllib` while doing so.

I am releasing it to the world in hopes that it helps someone else while they work with MetaSmoke's API. In return, I hope you will consider adding features you need and releasing them back to me so we can help others.

2.1.1 MIT License

A large number of open source projects you find today are GPL Licensed. While the GPL has its time and place, it should most certainly not be your go-to license for your next open source project.

A project that is released as GPL cannot be used in any commercial product without the product itself also being offered as open source.

The MIT, BSD, ISC, and Apache2 licenses are great alternatives to the GPL that allow your open-source software to be used freely in proprietary, closed-source software.

Requests is released under terms of the MIT License.

2.1.2 SmokeAPI's License

2.2 Installation

This part of the documentation provides an overview of how to install SmokeAPI.

2.2.1 Pip Install

SmokeAPI can be installed by simply running this command in your terminal:

```
$ pip install smokeapi
```

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

2.2.2 Source Code

SmokeAPI is developed on GitHub where the code is always available.

You can close the repository:

```
$ git clone git://github.com/AWegnerGitHub/smokeapi.git
```

Or download the tarball:

```
$ curl -OL https://github.com/AWegnerGitHub/smokeapi/tarball/master
  # optionally, zipball is also available (for Windows users).
```

Once you have a copy of the source, you can embed it in your own Python package or install it into your site-packages easily:

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

2.3 Quickstart

Ready to start talking to the Meta Smoke API? This page gives an introduction on how to get started with SmokeAPI.

First, you need to:

- *Install* SmokeAPI

2.3.1 Basic Data Retrieval

Retrieving data is very simple. First, though, you will need an API Key. This key will be used every place you see the variable `your_api_key`.

Note: This is an API *key*. It is a unique identifier for your application. It's not a disaster if you end up sharing it with other people unintentionally, but generally try to keep it protected. When you start making use of write capabilities, you'll be issued with API *tokens* that provide authorization on an app-user-pair basis; these tokens are sensitive information and should be protected no matter what.

First, import the SmokeAPI module:

```
>>> from smokeapi import SmokeAPI
```

Now we want to retrieve a list of posts that have been marked as "Not an Answer" by users:

```
>>> SMOKE = SmokeAPI('your_api_key')
>>> posts = SMOKE.fetch('posts/feedback', type="naa-")
```

This will return the 500 most recent posts that have been classified as "Not an Answer". The value passed to *`fetch`* is an end point defined in the MetaSmoke API Documentation.

If you are looking for more information on how to tailor the results of your queries. Take a look at the *Advanced Usage* examples.

2.3.2 Change number of results

By default, SmokeAPI will return up to 500 items in a single call. It may be less than this, if there are less than 500 items to return.

The number of results can be modified by changing the `per_page` and `max_pages` values. These are multiplied together to get the maximum total number of results. The API paginates the results and SmokeAPI recombines those pages into a single result.

The number of API calls that are made is dependant on the `max_pages` value. This will be the maximum number of calls that is made for this particular request.

All of these changes to `per_page` and `max_pages` need to occur before calls to `fetch` or `send_data`.

Let's walk through a few examples:

```
>>> SMOKE.per_page = 10
>>> SMOKE.max_pages = 10
```

This will return up to 100 results. However, it will hit the API up to 10 times.

```
>>> SMOKE.per_page = 100
>>> SMOKE.max_pages = 1
```

This will result up to 100 results as well, but it will only hit the API one time.

MetaSmoke limits the number of results per page to 100. If you want more than 100 results, you need to increase the `max_pages`.

```
>>> SMOKE.per_page = 100
>>> SMOKE.max_pages = 2
```

This will return up to 200 results and hit the API up to twice.

2.3.3 Getting exact number of results

If you want a specific number of results, but no more than that, you need to perform some manipulations of these two values.

```
>>> SMOKE.per_page = 50
>>> SMOKE.max_pages = 3
```

This will return up to 150 results. It will also hit the API 3 times to get these results. You can save an API hit by changing the values to:

```
>>> SMOKE.per_page = 75
>>> SMOKE.max_pages = 2
```

This will also return up to 150 results, but do so in only 2 API hits.

Note: Each "page" in an API call can have up to 100 results. In the first scenario, above, we are "wasting" 150 results because we only allow each page 50 results. In the second scenario, we are wasting 50 results. If you do not need an exact number of results, it is more efficient - number of API calls-wise - to set the `per_page` to 100 and return the highest number of results per page that the system allows.

2.3.4 Errors

SmokeAPI will throw an error if the MetaSmoke API returns an error. This can be caught in an exception block by catching *`smokeapi.SmokeAPIError`*. The exception has several values available to help troubleshoot the underlying issue:

```
except smokeapi.SmokeAPIError as e:
    print("   Error URL: {}".format(e.url))
    print("   Error Code: {}".format(e.error_code))
    print("   Error Name: {}".format(e.error_name))
    print("   Error Message: {}".format(e.error_message))
```

This will print out the URL that was being accessed, the error code that the API returns, the error name the API returns and the error message the API returns. Using these values, it should be possible to determine the cause of the error.

2.4 Advanced Usage

This portion of the documentation covers some of the more advanced features of SmokeAPI.

2.4.1 Calling `fetch` for specific IDs

Some of the end points accept IDs. The documentation says these are semicolon delimited lists of values. SmokeAPI, however, can handle this for you. You just need to pass a `list` to the `ids` keyword argument:

```
>>> from smokeapi import SmokeAPI
>>> SMOKE = SmokeAPI('your_api_key')
>>> post_ids = [44800, 44799, 800000]
>>> posts = SMOKE.fetch('posts', ids=post_ids)
>>> posts
{'has_more': False,
 'items': [{u'body': u'<p>You schould remove your redirection.\nAnd put a CName on␣
↪mydomain.co with value mydomain.herokuapp.com.</p>\n',
```

```
            u'created_at': u'2016-10-26T12:51:51.000Z',
            u'downvote_count': None,
            u'id': 44800,
            u'is_fp': True,
            u'is_naa': False,
            u'is_tp': False,
            u'link': u'//stackoverflow.com/a/40262819',
            u'post_creation_date': None,
            u'score': None,
            u'site_id': 1,
            u'stack_exchange_user_id': 37856,
            u'title': u'Heroku - custom domain DNS',
            u'updated_at': u'2016-10-26T12:52:13.000Z',
            u'upvote_count': None,
            u'user_link': u'//stackoverflow.com/u/4720079',
            u'user_reputation': 1,
            u'username': u'Lars Skogshus',
            u'why': u'Body - Position 1-111: <p>You should remove your redirection.
↪\nAnd put a CName on mydomain.co with value mydomain.herokuapp.com.</p>'},
           {u'body': u"<p>BUT IF YOU WANT TO CROSS ITALIN BORDER WITH CARTA D
↪'IDENTITA FROM NON SCHENGEN AREA IS IT POSSIBLE ?</p>\n",
            u'created_at': u'2016-10-26T12:50:11.000Z',
            u'downvote_count': None,
            u'id': 44799,
            u'is_fp': False,
            u'is_naa': True,
            u'is_tp': False,
            u'link': u'//travel.stackexchange.com/a/81481',
            u'post_creation_date': None,
            u'score': None,
            u'site_id': 108,
            u'stack_exchange_user_id': 37855,
            u'title': u"Travel in the Schengen area with only carta d'identita␣
↪italiana and permesso di soggiorno",
            u'updated_at': u'2016-10-26T12:51:23.000Z',
            u'upvote_count': None,
            u'user_link': u'//travel.stackexchange.com/u/52978',
            u'user_reputation': 1,
            u'username': u'IRINA',
            u'why': u'Post - All in caps'}],
 'page': 1,
 'total': 2}
```

Notice that we searched for 3 posts and only 2 results were returned. This is how the API operates. If an ID doesn't exist, a result will not be returned or indicated that it has been missed. It may be important for you to compare results to what you searched for to see if any values are missing.

Another thing to notice here is that only `posts` was passed as the end point. This works because the official end point is `posts/{ids}`. If you leave the `{ids}` off and it is the last part of the end point, SmokeAPI will automatically add it for you. An identical call would look like this, with `{ids}` included in the end point declaration.

```
>>> posts = SMOKE.fetch('posts/{ids}', ids=post_ids)
```

If `{ids}` is not at the end of the end point, then leaving it out of the target end point is **not** optional. This will **not work**:

```
>>> posts = SMOKE.fetch('reason/posts', ids=reason_ids)
```

However, this will work and will return posts associated with the selected close reasons:

```
>>> posts = SMOKE.fetch('reason/{ids}/posts', ids=reason_ids)
```

2.4.2 Proxy Usage

Some users sit behind a proxy and need to get through that before accessing the internet at large. SmokeAPI can handle this workflow.

A failure due to a proxy may look like this:

```
>>> from smokeapi import SmokeAPI, SmokeAPIError
>>> try:
...     SMOKE = SmokeAPI('your_api_key')
... except SmokeAPIError as e:
...     print(e.message)
...
('Connection aborted.', error(10060, 'A connection attempt failed
because the connected party did not properly respond after a period of
time, or established connection failed because connected host has failed
to respond'))
```

This can be fixed, by passing a dictionary of http and https proxy addresses when creating the *SmokeAPI* class:

```
>>> from smokeapi import SmokeAPI, SmokeAPIError
>>> proxies = {'http': 'http://proxy.example.com', 'https': 'http://proxy.example.com
↪'}
>>> try:
...     SMOKE = SmokeAPI('your_api_key', proxy=proxies)
... except SmokeAPIError as e:
...     print(e.message)
...
```

The two important lines are where `proxies` is defined and the modified *SmokeAPI* initialization, which passes the `proxies` dictionary to the `proxy` argument.

2.4.3 Calling `fetch` with various API parameters

Some end points take multiple arguments to help filter the number of results you return. SmokeAPI will accept all of these as parameters.

As an example, lets look at the search end point. This end point will accept the following parameters:

- feedback_type
- from_date
- to_date
- site

`page` and `per_page` are handled by SmokeAPI through usage of the `max_pages` and `per_page` values of the *SmokeAPI* class. The others, are part of the `kwargs` accepted by *fetch*.

Let's create an example using all of these. This should return a list of posts created between October 28, 2016 and October 29, 2016 that have a feedback type of `naa-` and were on Stack Overflow.

In this example, notice that we are passing a datetime object and not the expected UNIX timestamp representation. SmokeAPI handles the conversion for you automatically.

```
>>> from smokeapi import SmokeAPI
>>> import datetime
>>> SMOKE = SmokeAPI('your_api_key')
>>> end = datetime.datetime(2016, 10, 29)
>>> start = datetime.datetime(2016, 10, 28)
>>> feedbacktype = "naa-"
>>> site = "stackoverflow.com"
>>> posts = SMOKE.fetch('posts/search', from_date=start, to_date=end, feedback_
↪type=feedbacktype, site=site)
>>> posts
{   'has_more': False,
'items': [   {   u'body': <trimmed>,
                 u'created_at': u'2016-10-28T04:11:30.000Z',
                 u'downvote_count': None,
                 u'id': 44960,
                 u'is_fp': False,
                 u'is_naa': True,
                 u'is_tp': False,
                 u'link': u'//stackoverflow.com/a/40297916',
                 u'post_creation_date': None,
                 u'score': None,
                 u'site_id': 1,
                 u'stack_exchange_user_id': 38004,
                 u'title': u'Does LINQ have any easy/elegant way to take the first␣
↪element and put it at the end?',
                 u'updated_at': u'2016-10-29T16:38:31.000Z',
                 u'upvote_count': None,
                 u'user_link': u'//stackoverflow.com/u/7083522',
                 u'user_reputation': 1,
                 u'username': u'HuangKai',
                 u'why': u'Post - Text contains 24 non-Latin characters out of 26'}],
'page': 1,
'total': 1}
```

CHAPTER 3

The API Documentation

Information about specific functions, classes, and methods are available in this portion of documentation.

3.1 SmokeAPI Classes and Methods

This portion of the documentation covers all the interfaces of SmokeAPI.

3.1.1 SmokeAPI

class smokeapi.**SmokeAPI**(*key=None*, ***kwargs*)

__init__(*key=None*, ***kwargs*)

The object used to interact with the MetaSmoke API

Parameters

- **key** – (string) **(Required)** A valid API key. An API key can be received by following the current instructions in the API Documentation.
- **token** – (string) **(Required for write access/Optional is no write routes are called)** This is a valid write token retreived by following instructions in the API Documentation. If this is not set, calls to *send_data* will fail.
- **proxy** – (dict) (optional) A dictionary of http and https proxy locations Example:

```
{'http': 'http://example.com',
 'https': 'https://example.com'}
```

  By default, this is `None`.

- **max_pages** – (int) (optional) The maximum number of pages to retrieve (Default: `5`)
- **per_page** – (int) (optional) The number of elements per page. The API limits this to a maximum of 100 items on all end points (Default: `100`)

fetch(*endpoint=None*, *page=1*, ***kwargs*)
Returns the results of an API call.

This is the main work horse of the class. It builds the API query string and sends the request to MetaSmoke. If there are multiple pages of results, and we've configured *max_pages* to be greater than 1, it will automatically paginate through the results and return a single object.

Returned data will appear in the *items* key of the resulting dictionary.

Parameters

- **endpoint** – (string) The API end point being called. Available endpoints are listed on the official API Documentation.

  This can be as simple as `fetch('posts/feedback')`, to call feedback end point

  If calling an end point that takes additional parameter, such as *id's pass the ids as a list to the 'ids* key:

  ```
  fetch('posts/{ids}', ids=[1,2,3])
  ```

  This will attempt to retrieve the posts for the three listed ids.

  If no end point is passed, a `ValueError` will be raised

- **page** – (int) The page in the results to start at. By default, it will start on the first page and automatically paginate until the result set reaches `max_pages`.

- **kwargs** – Parameters accepted by individual endpoints. These parameters **must** be named the same as described in the endpoint documentation

Return type (dictionary) A dictionary containing wrapper data regarding the API call and the results of the call in the *items* key. If multiple pages were received, all of the results will appear in the `items` tag.

send_data(*endpoint=None*, ***kwargs*)
Sends data to the API.

This call is similar to `fetch`, but **sends** data to the API instead of retrieving it.

Returned data will appear in the `items` key of the resulting dictionary.

Sending data **requires** that the `token` is set.

Parameters

- **endpoint** – (string) **(Required)** The API end point being called. Available endpoints are listed on the official API Documentation.

  If no end point is passed, a `ValueError` will be raised

- **kwargs** – Parameters accepted by individual endpoints. These parameters **must** be named the same as described in the endpoint documentation

Return type (dictionary) A dictionary containing wrapper data regarding the API call and the results of the call in the *items* key. If multiple pages were received, all of the results will appear in the `items` tag.

3.1.2 SmokeAPIError

class smokeapi.**SmokeAPIError**(*url*, *code*, *name*, *message*)
The Exception that is thrown when ever there is an API error.

Parameters

- **`url`** – (string) The URL that was called and generated an error
- **`code`** – (int) The *error_code* returned by the API
- **`name`** – (string) The *error_name* returned by the API and is human friendly
- **`message`** – (string) The *error_message* returned by the API

CHAPTER 4

Contributor Guidelines

Information about how to contribute to the project is available in this portion of the documentation.

4.1 Contributor's Guide

If you're reading this, you're probably interested in contributing to SmokeAPI. Thank you! The fact that you're even considering contributing to the SmokeAPI project is *very* generous of you.

This document lays out guidelines and advice for contributing to this project. If you're thinking of contributing, please start by reading this document and getting a feel for how contributing to this project works. If you have any questions, feel free to reach out to Andrew Wegner, the primary maintainer.

The guide is split into sections based on the type of contribution you're thinking of making, with a section that covers general guidelines for all contributors.

4.1.1 Be Nice

SmokeAPI has one important rule covering all forms of contribution, including reporting bugs or requesting features. This golden rule is "Be Nice".

All contributions are welcome, as long as everyone involved is treated with respect.

4.1.2 Early Feedback

If you are contributing, do not feel the need to sit on your contribution until it is polished and complete. It helps everyone involved for you to seek feedback as early as you possibly can. Submitting an early, unfinished version of your contribution for feedback does not decrease your chances of getting that contribution accepted, and can save you from putting a lot of work into a contribution that is not suitable for the project.

4.1.3 Contribution Suitability

Our project maintainers have the last word on whether or not a contribution is suitable. All contributions will be considered carefully, but from time to time, contributions will be rejected because they do not suit the current goals or needs of the project.

4.1.4 Code Contributions

Steps for Submitting Code

When contributing code, you'll want to follow this checklist:

1. Fork the repository on GitHub.
2. Run the tests to confirm they all pass on your system. If they don't, you'll need to investigate why they fail. If you're unable to diagnose this yourself, raise it as a bug report by following the guidelines in this document: *Bug Reports*.
3. Write tests that demonstrate your bug or feature. Ensure that they fail. Note that many of our tests use mock to prevent burning through API quota. We ask that you do them and provide a mocked response.
4. Make your change.
5. Run the entire test suite again, confirming that all tests pass *including the ones you just added*.
6. Send a GitHub Pull Request to the main repository's `master` branch. GitHub Pull Requests are the expected method of code collaboration on this project.

Running Tests

Note: While this note exists, there are **NO** unit tests. This is a known limitation and will be corrected at some point in the future.

To be able to run the test suite, you'll need to have mock installed. Mock is on the Python Package Index, so you can install it simply with one command:

```
$ pip install mock
```

Tests are built and run using unittest, which comes as a standard package with every Python installation. You can run the tests using the following command (from the root directory of your clone):

```
$ python -m unittest discover
```

The `mock` installation step can be handled automatically, if you run tests via:

```
$ python setup.py test
```

Code Review

Contributions will not be merged until they've been reviewed. You should implement any code review feedback unless you strongly object to it. In the event that you object to the code review feedback, you should make your case clearly and calmly. If, after doing so, the feedback is judged to still apply, you must either apply the feedback or withdraw your contribution.

4.1.5 Documentation Contributions

Documentation improvements are always welcome! The documentation files live in the `docs/` directory of the codebase. They're written in reStructuredText, and use Sphinx to generate the full suite of documentation.

When contributing documentation, please do your best to follow the style of the documentation files. This means a soft-limit of 79 characters wide in your text files and a semi-formal, but friendly and approachable, prose style.

When presenting Python code, use single-quoted strings (`'hello'` instead of `"hello"`).

4.1.6 Bug Reports

Bug reports are hugely important! Before you raise one, though, please check through the GitHub issues, **both open and closed**, to confirm that the bug hasn't been reported before. Duplicate bug reports can be a huge drain on the time of other contributors, and should be avoided as much as possible.

4.1.7 Feature Requests

If you believe there is a feature missing, feel free to raise a feature request. Please provide as much detail about the request as you can including some of the following information:

- Intended use case(s)
- Short falls you have with the current version
- Possible expected results

4.2 How to Help

SmokeAPI is under active development, and contributions are more than welcome! There are details at *Contributing*, but the short version is:

1. Check for open issues or open a fresh issue to start a discussion around a bug.
2. Fork the repository on GitHub and start making your changes to a new branch.
3. Write a test which shows that the bug was fixed.
4. Send a pull request. Make sure to add yourself to AUTHORS.

4.3 Authors

SmokeAPI is written and maintained by Andrew Wegner and (hopefully soon) various contributors.

4.3.1 Project Owner

Andrew Wegner @AWegnerGitHub

4.3.2 Patches and Suggestions

Contribute a feature and get your name here!

CHAPTER 5

Release History

5.1 0.2.0 (2016-11-06)

- Adds support for `from_date` and `to_date` and automatic conversion of `datetime` objects to expected integer for these two parameters
- Updates documentation

5.2 0.1.0 (2016-10-26)

- Initial Release

Python Module Index

S

Index

Symbols

F

S