StackAPI Documentation

Release 0.1.12

Andrew Wegner

Jan 18, 2019

Contents

Release v0.1.12. (*Installation*)

StackAPI is a simple Python wrapper for the Stack Exchange API and supports the 2.2 API.

Retrieving data from the API is simple:

```
from stackapi import StackAPI
SITE = StackAPI('stackoverflow')
comments = SITE.fetch('comments')
```

The above, will issue a call to the `comments` end point on Stack Overflow and retrieve the 600 newest comments.

CHAPTER 1

Supported Features

- Automatically obeys the `backoff` parameter.
- 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.

StackAPI is supported on Python 2.7 - 3.7.

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

StackAPI was written to scratch an itch and in the process 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 Stack Exchange'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 StackAPI License

2.2 Installation

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

2.2.1 Pip Install

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

```
$ pip install stackapi
```

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

2.2.2 Source Code

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

You can close the repository:

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

Or download the tarball:

```
$ curl -OL https://github.com/AWegnerGitHub/stackapi/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 Stack Exchange API? This page gives an introduction on how to get started with StackAPI.

First, you need to:

- *Install* StackAPI

2.3.1 Basic Data Retrieval

Retrieving data is very simple.

First, import the StackAPI module:

```
>>> from stackapi import StackAPI
```

Now we want to retrieve the most recent comments from Stack Overflow:

```
>>> SITE = StackAPI('stackoverflow')
>>> comments = SITE.fetch('comments')
```

This will return the 500 most recent comments on Stack Overflow, using the default filter the API provides. The value passed to *fetch* is an end point defined in the Stack Exchange API.

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, StackAPI 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. This is common on new or low traffic sites.

The number of results can be modified by changing the `page_size` and `max_pages` values. These are multiplied together to get the maximum total number of results. The API paginates the results and StackAPI 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 `page_size` and `max_pages` need to occur before calls to `fetch` or `send_data`.

Let's walk through a few examples:

```
>>> SITE.page_size = 10
>>> SITE.max_pages = 10
```

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

```
>>> SITE.page_size = 100
>>> SITE.max_pages = 1
```

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

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

```
>>> SITE.page_size = 100
>>> SITE.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.

```
>>> SITE.page_size = 50
>>> SITE.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:

```
>>> SITE.page_size = 75
>>> SITE.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 - API quota-wise - to set the `page_size` to 100 and return the highest number of results per page that the system allows.

2.3.4 Errors

StackAPI will throw an error if the Stack Exchange API returns an error. This can be caught in an exception block by catching *`stackapi.StackAPIError`*. The exception has several values available to help troubleshoot the underlying issue:

```
except stackapi.StackAPIError as e:
    print("   Error URL: {}".format(e.url))
    print("   Error Code: {}".format(e.code))
    print("   Error Error: {}".format(e.error))
    print("   Error Message: {}".format(e.message))
```

This will print out the URL that was being accessed, the error number that the API returns, the error code the API returns and the error message the API returns. Using these values, it should be possible to determine the cause of the error. The error code, message and number are all available in the documentation.

2.3.5 Note about Quotas

Stack Exchange attempts to prevent abuse of their API by implementing a number of throttles. The quote values that you have remaining are returned with more filters that StackAPI utilizes. These appear as `quota_remaining` and `quota_max` values.

In some instances though (for example when using a `filter='total'`), these are not part of the Stack Exchange response. In these instances, StackAPI will set the values to `-1`. This is to make it clear that StackAPI does not know the values and not provide you with an incorrect, positive, value.

To get accurate values for these two fields, you need to ensure that your filters contain `quota_remaining` and `quota_max`.

2.4 Advanced Usage

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

2.4.1 Calling `fetch` with various API parameters

The various end points all take multiple parameters to help filter the number of results you return. StackAPI will accept all of these parameters.

As an example, let's look at the comments end point. This end point will accept the following parameters:

- page
- pagesize
- fromdate
- todate
- order
- min
- max
- sort

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

Let's create an example using several of these parameters. This should return a list of questions created between March 5, 2016 and March 6, 2016 that have a score of at least 20 and are tagged `python`:

```
>>> from stackapi import StackAPI
>>> SITE = StackAPI('stackoverflow')
>>> questions = SITE.fetch('questions', fromdate=1457136000, todate=1457222400,
    min=20, tagged='python', sort='votes')
>>> questions
{
    'backoff': 0,
    'has_more': False,
    'items': [
                {
                    u'accepted_answer_id': 35818398,
                    u'answer_count': 2,
                    u'creation_date': 1457186774,
                    u'is_answered': True,
                    u'last_activity_date': 1457246584,
                    u'last_edit_date': 1457200889,
                    u'link': u'http://stackoverflow.com/questions/35815093/sine-
↪calculation-orders-of-magnitude-slower-than-cosine',
                    u'owner': {u'accept_rate': 80,
                               u'display_name': u'Finwood',
                               u'link': u'http://stackoverflow.com/users/1525423/
↪finwood',
                               u'profile_image': u'https://i.stack.imgur.com/xkRry.
↪png?s=128&g=1',
                               u'reputation': 1575,
                               u'user_id': 1525423,
                               u'user_type': u'registered'},
                    u'question_id': 35815093,
                    u'score': 22,
                    u'tags': [u'python', u'numpy', u'scipy', u'signal-processing'],
                    u'title': u'sine calculation orders of magnitude slower than␣
↪cosine',
```

(continues on next page)

(continued from previous page)

```
                    u'view_count': 404
                }
            ],
    'page': 1,
    'quota_max': 300,
    'quota_remaining': 171,
    'total': 0
}
```

We can see that a single question matched our criteria.

Note: In the above example, we passed a timestamp value to the `fromdate` and ``todate` parameters. StackAPI will also allow you to pass in `datetime` objects and automatically perform this conversion for you.

StackAPI will perform this conversion to the following parameters:

- fromdate
- todate
- since
- min
- max

The `min` and `max` parameters are not exclusively date fields in the API. StackAPI will only convert these to timestamps if a `datetime` object is passed.

2.4.2 Calling `fetch` for specific IDs

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

```
>>> from stackapi import StackAPI
>>> SITE = StackAPI('stackoverflow')
>>> badges = SITE.fetch('badges', ids=[222, 1306, 99999])
>>> badges
{
    'backoff': 0,
    'has_more': False,
    'items': [
                {
                    u'award_count': 6036,
                    u'badge_id': 222,
                    u'badge_type': u'named',
                    u'link': u'http://stackoverflow.com/badges/222/altruist',
                    u'name': u'Altruist',
                    u'rank': u'bronze'
                },
                {
                    u'award_count': 43954,
                    u'badge_id': 1306,
                    u'badge_type': u'named',
                    u'link': u'http://stackoverflow.com/badges/1306/analytical',
                    u'name': u'Analytical',
                    u'rank': u'bronze'
                }
```

(continues on next page)

(continued from previous page)

```
            ],
    'page': 1,
    'quota_max': 300,
    'quota_remaining': 160,
    'total': 0
}
```

Notice that we searched for 3 badges 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 `badges` was passed as the end point. This works because the official end point is `badges/{ids}`. If you leave the `{ids}` off and it is the last part of the end point, StackAPI will automatically add it for you. An identical call would look like this, with `{ids}` included in the end point declaration.

```
>>> badges = SITE.fetch('badges/{ids}', ids=[222, 1306, 99999])
```

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**:

```
>>> answers = SITE.fetch('/answers/comments', ids=[1, 2, 3])
```

However, this will work and will return comments associated with the selected answers:

```
>>> answers = SITE.fetch('/answers/{ids}/comments', ids=[1, 2, 3])
```

2.4.3 Proxy Usage

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

A failure due to a proxy may look like this:

```
>>> from stackapi import StackAPI, StackAPIError
>>> try:
...     SITE = StackAPI('stackoverflow')
... except StackAPIError 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 *StackAPI* class:

```
>>> from stackapi import StackAPI, StackAPIError
>>> proxies = {'http': 'http://proxy.example.com', 'https': 'http://proxy.example.com
↪'}
>>> try:
...     SITE = StackAPI('stackoverflow', proxy=proxies)
... except StackAPIError as e:
...     print(e.message)
...
```

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

2.4.4 End points that don't accept `site` parameter

Not all end points accept a site parameter. This is limited to only a few end points. In this case, you'll receive a *`StackAPIError`* with

- `code` equal to `bad_parameter`
- `message` equal to `This method does not accept a 'site' parameter`

The solution is to clear the `_api_key`.

```
>>> from stackapi import StackAPI
    >>> SITE = StackAPI('stackoverflow')
>>> SITE._api_key = None
>>> associated_users = SITE.fetch('/users/{}/associated'.format(63984), pagesize=1)
>>> associated_users
{
    "items": [
        {
            "badge_counts": {
                "bronze": 91,
                "silver": 56,
                "gold": 11
            },
        "question_count": 94,
        "answer_count": 627,
        "last_access_date": 1457107832,
        "creation_date": 1255441385,
        "account_id": 63984,
        "reputation": 17438,
        "user_id": 189134,
        "site_url": "http://stackoverflow.com",
        "site_name": "Stack Overflow"
        }
    ],
    "has_more": true,
    "quota_max": 10000,
    "quota_remaining": 9989
}
```

2.4.5 Send data via the API

StackAPI can send data to a Stack Exchange site, provided you have an application registered on StackApps and registered a token with write access.

The following example assumes that you have a registered application and registered a user against that application. The `key` value is provided on the Stack Apps App Management page. The token is provided when a user authorizes the application to act on their behalf. StackAPI does not perform or support this authorization. Stack Exchange provides three methods for performing this authorization.

Reminder: The `access_token` is *per user*. It is associated with a single user. If StackAPI is acting on behalf of multiple users, the token **must** be changed each time `send_data` is utilized.

```
>>> from stackapi import StackAPI
>>> SITE = SEAPI.SEAPI('stackoverflow', key=APP_KEY, access_token=ACCESS_TOKEN)
>>> flag = SITE.send_data('comments/123/flags/add', option_id=option_id)
```

This block will flag comment `123` with the value of `option_id`. This value can be found by checking which options are valid for the flag.

```
>>> flag_options = SITE.fetch('comments/123/flags/options')
```

Assuming that comment `123` is valid, this call will return a list of valid flags. From here, you select the `option_id` that matches the reason you want to flag. When you pass that via `send_data` in the `option_id` flag, StackAPI will issue a flag on behalf of the user associated with the provided `access_token`.

2.5 Complex API Queries

The API Documentation explains that advanced queries can be generated using combinations of the `min`, `max`, `sort`, `fromdate` and `todate` parameters. This portion of the documentation explains how to replicate the examples provided using StackAPI.

2.5.1 All Stack Overflow Users Created on Feb. 27th of 2011

```
>>> from stackapi import StackAPI
>>> from datetime import datetime
>>> SITE = StackAPI('stackoverflow')
>>> SITE.max_pages=10
>>> users = SITE.fetch('users', fromdate=datetime(2011,02,27), todate=datetime(2011,
↪02,28))
>>> len(users['items'])
623
```

This one is pretty straight forward. The biggest thing here is that `max_pages` is adjusted upward because there are more new users created on that day than StackAPI returns by default.

2.5.2 Comments with at least a score of 10 on Ask Ubuntu

```
>>> from stackapi import StackAPI
>>> SITE = StackAPI('askubuntu')
>>> comments = SITE.fetch('comments', min=10, sort='votes')
>>> len(comments['items'])
769
```

Note This value differs from the result in the `total` element. This is *not* a StackAPI bug, but appears to be a problem on the Stack Exchange side. This has been reported as a bug and this documentation will be adjusted depending on the outcome of the bug.

2.5.3 Of three specific posts on Server Fault, which one has the most recent activity

```
>>> from stackapi import StackAPI
>>> SITE = StackAPI('serverfault')
>>> SITE.max_pages=1
>>> SITE.page_size=1
>>> post = SITE.fetch('posts', ids=[3743, 327738, 339426], sort='activity', order=
↪'desc')
>>> post['items'][0]['post_id']
339426
```

2.5.4 Any favorites added in the month of December 2011 by Darin Dimitrov

```
>>> SITE = StackAPI('stackoverflow')
>>> from datetime import datetime
>>> favorites = SITE.fetch('users/{ids}/favorites', min=datetime(2011, 12, 1),␣
↪max=datetime(2012, 1, 1), sort='added', ids=[29407])
>>> len(favorites['items'])
9
```

2.5.5 Questions created during the Modern Warfare 3 VS Skyrim Contest with the `skryim` tag and a score greater than 10 on Gaming Stack Exchange

```
>>> SITE = StackAPI('gaming')
>>> from datetime import datetime
>>> questions = SITE.fetch('questions', fromdate=datetime(2011,11,11),␣
↪todate=datetime(2011,11,19), min=10, sort='votes', tagged='skyrim')
>>> len(questions['items'])
231
```

CHAPTER 3

The API Documentation

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

3.1 StackAPI Classes and Methods

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

3.1.1 StackAPI

class stackapi.**StackAPI**(*name=None*, *version='2.2'*, ***kwargs*)

__init__(*name=None*, *version='2.2'*, ***kwargs*)

The object used to interact with the Stack Exchange API

Parameters

- **name** – (string) **(Required)** A valid `api_site_parameter` (available from http://api.stackexchange.com/docs/sites) which will be used to connect to a particular site on the Stack Exchange Network.
- **version** – (float) **(Required)** The version of the API you are connecting to. The default of `2.2` is the current version
- **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: `100`)
- **page_size** – (int) (optional) The number of elements per page. The API limits this to a maximum of 100 items on all end points except `site`

- **key** – (string) (optional) An API key
- **access_token** – (string) (optional) An access token associated with an application and a user, to grant more permissions (such as write access)

__repr__(*) <==> *repr(x)*

fetch(*endpoint=None*, *page=1*, *key=None*, *filter='default'*, ***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 Stack Exchange. 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: http://api.stackexchange.com/docs

  This can be as simple as `fetch('answers')`, to call the answers 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('answers/{}', ids=[1,2,3])
  ```

  This will attempt to retrieve the answers 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 reached `max_pages`.
- **key** – (string) The site you are issuing queries to.
- **filter** – (string) The filter to utilize when calling an endpoint. Different filters will return different keys. The default is `default` and this will still vary depending on what the API returns as default for a particular endpoint
- **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*, *page=1*, *key=None*, *filter='default'*, ***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 `access_token` is set. This is enforced on the API side, not within this library.

Parameters

- **endpoint** – (string) The API end point being called. Available endpoints are listed on the official API documentation: http://api.stackexchange.com/docs

  This can be as simple as `fetch('answers')`, to call the answers 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('answers/{}', ids=[1,2,3])
```

This will attempt to retrieve the answers 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 reached `max_pages`.
- **key** – (string) The site you are issuing queries to.
- **filter** – (string) The filter to utilize when calling an endpoint. Different filters will return different keys. The default is `default` and this will still vary depending on what the API returns as default for a particular endpoint
- **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 StackAPIError

class stackapi.**StackAPIError**(*url*, *error*, *code*, *message*)

The Exception that is thrown when ever there is an API error.

This utilizes the values returned by the API and described here: http://api.stackexchange.com/docs/types/error

Parameters

- **url** – (string) The URL that was called and generated an error
- **error** – (int) The *error_id* returned by the API (should be an int)
- **code** – (string) The *description* returned by the API and is human friendly
- **message** – (string) The *error_name* returned by the API

__init__(*url*, *error*, *code*, *message*)

x.__init__(...) initializes x; see help(type(x)) for signature

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 StackAPI. Thank you! The fact that you're even considering contributing to the StackAPI 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

StackAPI 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

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

StackAPI 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

StackAPI 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

@ArtOfCode-

Contribute a feature and get your name here!

Python Module Index

S

Index

Symbols

F

S