mypolr Documentation

Release 1.3.11

Thomas Fauskanger

Feb 23, 2018

Contents:

This package, *mypolr*, is a python package to easily create and manage short links using the Polr Project's REST API. Mypolr also has CLI support.

User Guide and documentation: https://mypolr.readthedocs.io

GitHub: https://github.com/fauskanger/mypolr

Clone source: `git clone git://github.com/fauskanger/mypolr.git`

PyPI: `pip install mypolr` [PyPI.org | Legacy]

Note: Disclaimer: This package, *mypolr*, is not affiliated with the Polr Project.

CHAPTER 1

Overview

This package, *mypolr*, is a python package to easily create and manage short links using the Polr Project's REST API. Mypolr also has CLI support.

User Guide and documentation: https://mypolr.readthedocs.io

GitHub: https://github.com/fauskanger/mypolr

Clone source: `git clone git://github.com/fauskanger/mypolr.git`

PyPI: `pip install mypolr` [PyPI.org | Legacy]

Project	
Git	
Statuses	

1.1 Requirements

1.1.1 Polr Project

Documentation: https://docs.polrproject.org

To use *mypolr*, you need a valid API key to a server with the Polr Project installed.

You can obtain the API key by logging in to your Polr site and navigate to *<polr project root>/admin#developer*.

Note: **Disclaimer:** This package, *mypolr*, is not affiliated with the Polr Project.

1.1.2 Python

There is only one requirement:

- `requests`, an awesome HTTP library. (Documentation).

When installing with *pip* or *conda* this will be installed automatically (if not already installed).

Tested on Python 2.7, 3.4+, but should also work with version 3.3.

1.2 Installation

With *pip*: `pip install mypolr`

With *conda*: `conda install -c fauskanger mypolr`

1.3 ToBeDone

- Add `:raises:` docstring fields to methods/docs.
- Implement the `/data/link`-endpoint if necessary.

1.4 License

This project is licensed under the MIT Licence. (See link for details.)

CHAPTER 2

User Guide

2.1 Quick Example

Here is an incomplete example. See *Advanced Usage* below for a more detailed description.

```
from mypolr import PolrApi, exceptions

# Replace with your values
server_url = 'polr.example.com'
api_key = '1234567890abcdef'

# Example url to shorten
long_url = 'https://some.long.example.com/long/url'

# Create PolrApi instance
api = PolrApi(server_url, api_key)

# Make short urls
shorted_url = api.shorten(long_url)
custom_url = api.shorten(long_url, custom=CUSTOM_ENDING)

# Given a short url ending, find full url and stats:
lookup_dict = api.lookup(SHORT_URL_ENDING)
full_url = lookup_dict.get('long_url')
n_clicks = lookup_dict.get('clicks')

# Secret urls have an additional key after the short url ending
# E.g the format <polr root folder> / SHORT_URL_ENDING / URL_KEY:
secret_url = api.shorten(long_url, is_secret=True)
# Secret lookups require url_key:
secret_lookup = api.lookup(SHORT_URL_ENDING, url_key=URL_KEY)
```

2.2 Advanced Usage

This section is more thorough than the one above, and covers the various errors and edge cases you might encounter with the Polr Project API.

The following examples assume the Polr Project to be installed on a server at *https://ti.ny*, and that a valid API_KEY is stored in a separate module `my_secrets`.

2.2.1 Set up API

This is how the API would be set up given the aforementioned (and arbitrary) assumptions:

```
from mypolr import PolrApi, exceptions
from my_secrets import api_key

# Example url to shorten
long_url = 'https://stackoverflow.com/questions/tagged/python'

# Your api server url
server_url = 'https://ti.ny'

# Create PolrApi instance
api = PolrApi(server_url, api_key)
```

2.2.2 Shorten long URLs

Given a long url, the *PolrApi.shorten*-method produces a short url on the form `https://ti.ny / URL_ENDING`:

```
try:
    # Generate a short url with automatic mapping
    automatic_url = api.shorten(long_url)
    # Generate a short url with the ending 'soPython'
    custom_url = api.shorten(long_url, custom_ending='soPython')

    print(automatic_url)    # E.g. https://ti.ny/5N3f8
    print(custom_url)       # E.g. https://ti.ny/soPython
except exceptions.UnauthorizedKeyError:
    print('API_KEY invalid or inactive.')
except exceptions.CustomEndingUnavailable:
    print('Custom ending is already in use: choose another.')
except exceptions.QuotaExceededError:
    print('User account associated with API_KEY has exceeded their quota.')
except exceptions.ServerOrConnectionError:
    print('Check server and/or connection status.')
except exceptions.BadApiRequest:
    print('Something was wrong with the request to server.')
except exceptions.BadApiResponse:
    print('Response from server was not valid JSON.')
```

2.2.3 Lookup short URLs

The *PolrApi.lookup*-method accepts either a short url ending, or a full short url, and returns `False` if no url is found, or returns a dictionary of info about the link.

```
try:
    # Lookup short url to get info
    url_info = api.lookup('https://ti.ny/soPython')
    url_info = api.lookup('soPython')
    if url_info is False:
        print('No url found with that ending.')
    else:
        print('Long url is: {}'.format(url_info.get('long_url')))
except exceptions.UnauthorizedKeyError:
    print('API_KEY invalid or inactive.')
except exceptions.ServerOrConnectionError:
    print('Check server and/or connection status.')
except exceptions.BadApiRequest:
    print('Something was wrong with the request to server.')
except exceptions.BadApiResponse:
    print('Response from server was not valid JSON.')
```

Lookup result

Response of a successful lookup is a dictionary á la something like this:

```
{
    'clicks': 42,
    'created_at':
        {
            'date': '2017-12-03 00:40:45.000000',
            'timezone': 'UTC',
            'timezone_type': 3
        },
    'long_url': 'https://stackoverflow.com/questions/tagged/python',
    'updated_at':
        {
            'date': '2017-12-03 00:40:45.000000',
            'timezone': 'UTC',
            'timezone_type': 3
        }
}
```

2.2.4 Secret URLs

Secret urls differ from normal short urls in the way that they have the form `https://ti.ny / URL_ENDING / URL_KEY`. The additional part, URL_KEY, is required as a parameter when doing lookup of secret urls.

```
# Working with secret urls
secret_long_url = 'https://stackoverflow.com/questions/tagged/cryptography'

# Can still use both automatic or custom mapping
secret_url = api.shorten(secret_long_url, is_secret=True)
secret_custom_url = api.shorten(secret_long_url, custom_ending='soSecret', is_
↪secret=True)
```

```
print(secret_url)           # E.g. https://ti.ny/gztns/bXL2
print(secret_custom_url)    # E.g. https://ti.ny/soSecret/F3iH

try:
    secret_url_info = api.lookup('soPython', url_key='F3iH')
except exceptions.UnauthorizedKeyError:
    print('Your URL_KEY is wrong, or the API_KEY is invalid.')
```

Note: The `exceptions.UnauthorizedKeyError` in the previous example is the sole catch in order to simplify the example about secret lookups, but as seen in *Lookup short URLs* above, this isn't the only exception that could be raised.

2.2.5 Ignoring Errors

The *exceptions.no_raise*-decorator has been applied to both *PolrApi.shorten_no_raise* and *PolrApi.lookup_no_raise*, and will act as their corresponding normal methods, but will return `None` instead of raising **module** exceptions upon errors.

The *PolrApi.lookup_no_raise*-method still returns `False` when no url is found (if no error occurs).

```
# Use the _no_raise-methods to return None instead of exceptions as above
short_url = api.shorten_no_raise(long_url)
url_info = api.lookup_no_raise('soPython')

if short_url is None:
    print('There was an error with the url shortening process.')

if url_info is False:
    print('No url with that ending.')
elif url_info is None:
    print('There was an error with the url lookup process.')
```

Warning: Even though the use of **_no_raise*-methods allows for easy check of failure/success, there is no feedback of what went wrong upon failure.

Note: The **_no_raise*-methods will still raise *other* exceptions, and **ONLY** errors derived from *MypolrError* will instead return `None`.

2.3 CLI usage

CLI-support was added in *version 1.3* and is supported for Python >= 3.4.

2.3.1 Examples

Assuming Polr Project is intalled on *https://ti.ny* and that your API_KEY is *abcdef1234567890*, below is a few examples. Read full description of the interface further down.

Basic example, performs the default *PolrApi.shorten* action.

```
python -m mypolr http://some.long.example.com --server https://ti.ny --key␣
↪abcdef1234567890
```

Set `-l`/`--lookup` option to perform *PolrApi.lookup* action.

```
python -m mypolr https://ti.ny/5Bn8V --lookup
```

In the previous example, no server configuration values were used. They, including the key, can be saved **in plaintext** to *~/.mypolr/config.ini* with the `--save` option. This will load the saved values if not presented upon invocation.

```
python -m mypolr --server https://ti.ny --key abcdef1234567890 --save
```

Clear the *config.ini*-file with the `--clear` option.

```
python -m mypolr --clear
```

2.3.2 CLI description

Interacts with the Polr Project's API.

User Guide and documentation: https://mypolr.readthedocs.io

```
usage: mypolr [-h] [-v] [-s SERVER] [-k KEY] [--api-root API_ROOT] [-c CUSTOM]
              [--secret] [-l] [--save] [--clear]
              [url]
```

Positional Arguments

url The url to process.

Named Arguments

-v, --version Print version and exit.

Default: False

API server arguments

Use these for configure the API. Can be stored locally with –save.

-s, --server Server hosting the API.

-k, --key API_KEY to authenticate against server.

--api-root API endpoint root.

Default: "/api/v2/"

Action options

Configure the API action to use.

-c, --custom Custom short url ending.

--secret Set option if using secret url.

Default: False

-l, --lookup Perform lookup action instead of shorten action.

Default: False

Manage credentials

Use these to save, delete or update SERVER, KEY and/or API_ROOT locally in ~/.mypolr/config.ini.

--save Save configuration (including credentials) in plaintext(!).

Default: False

--clear Clear configuration.

Default: False

NOTE: if configurations are saved, they are stored as plain text on disk, and can be read by anyone with access to the file.

CHAPTER 3

Class Specifications

This section is generated with the Sphinx autodoc extension.

3.1 PolrApi

This file defines the main component of the Mypolr package: the *PolrApi* class.

class mypolr.polr_api.**PolrApi**(*api_server*, *api_key*, *api_root='/api/v2/'*)

Url shorter instance that stores server and API key

Parameters

- **api_server** (*str*) – The url to your server with Polr Project installed.
- **api_key** (*str*) – The API key associated with a user on the server.
- **api_root** (*str*) – API root endpoint.

lookup(*lookup_url*, *url_key=None*)

Looks up the url_ending to obtain information about the short url.

If it exists, the API will return a dictionary with information, including the long_url that is the destination of the given short url URL.

The lookup object looks like something like this:

```
{
    'clicks': 42,
    'created_at':
        {
            'date': '2017-12-03 00:40:45.000000',
            'timezone': 'UTC',
            'timezone_type': 3
        },
    'long_url': 'https://stackoverflow.com/questions/tagged/python',
    'updated_at':
```

```
        {
            'date': '2017-12-24 13:37:00.000000',
            'timezone': 'UTC',
            'timezone_type': 3
        }
}
```

Parameters

- **lookup_url** (*str*) – An url ending or full short url address
- **url_key** (*str or None*) – optional URL ending key for lookups against secret URLs

Returns Lookup dictionary containing, among others things, the long url; or None if not existing

Return type dict or None

lookup_no_raise(**args*, ***kwargs*)
Calls *PolrApi.lookup(*args, **kwargs)* but returns *None* instead of raising module errors.

shorten(*long_url*, *custom_ending=None*, *is_secret=False*)
Creates a short url if valid

Parameters

- **long_url** (*str*) – The url to shorten.
- **custom_ending** (*str or None*) – The custom url to create if available.
- **is_secret** (*bool*) – if not public, it's secret

Returns a short link

Return type str

shorten_no_raise(**args*, ***kwargs*)
Calls *PolrApi.shorten(*args, **kwargs)* but returns *None* instead of raising module errors.

3.2 Exceptions

All Mypolr exceptions are inherited from *MypolrError* and are defined in this file.

exception mypolr.exceptions.**BadApiRequest**
Raised when a request is malformed or otherwise is not understandable by server.

exception mypolr.exceptions.**BadApiResponse**(*msg='Cannot interpret API response: invalid JSON.'*)
Raised when a response is malformed and cannot be interpreted as valid JSON.

exception mypolr.exceptions.**CustomEndingUnavailable**(*custom_ending*)
Raised when a custom ending is in use and therefore cannot be created.

Parameters **custom_ending** (*str*) – the custom ending that was unavailable

exception mypolr.exceptions.**DebugTempWarning**
Temporary Warning that should be removed

exception mypolr.exceptions.**MypolrError**
Base class for all module exceptions

exception mypolr.exceptions.**QuotaExceededError**
Admins may assign quotas to users, and this is raised when it's exceeded and service stopped.

exception mypolr.exceptions.**ServerOrConnectionError**(*caused=None*)
Raised when there is a timeout, internal server error, or any other connection error.

exception mypolr.exceptions.**UnauthorizedKeyError**(*msg=None*)
Raised when an invalid key has been used in a request.

This refers either to: the API_KEY used in all endpoints, or the URL_KEY optionally used at the lookup endpoint.

Parameters **msg** –

mypolr.exceptions.**no_raise**(*f*)
Decorator/wrapper function to force return None instead of raising module exceptions.

Exceptions that can be ignored are found in mypolr.exceptions.

3.3 Testing

3.3.1 Use tox with conda

See: *Working with Windows, conda, tox*.

3.3.2 ResponseErrorMap

Eases the burden of making repeated calls to the same url for testing when using `pytest` and `responses`.

class test_mypolr.**ResponseErrorMap**(*endpoint*, *response_args=None*, *errors=None*, *common_kwargs=None*)
Maps sets of `responses.add()`-arguments to expected exceptions.

This works with pytest and responses.

Either pass mappings to initializer, or add them as pairs with `add(response_args, errors)`:

```
response_args = [
    dict(status=401, json=dict(error='please authorize')),
    dict(status=500, json=dict(error='internal error')),
    dict(body=requests.RequestException()),
    dict(body=ValueError())
]

errors = [
    UnauthorizedKeyError,
    ServerOrConnectionError,
    ServerOrConnectionError,
    BadApiResponse,
]

rmap = ResponseErrorMap(response_args, errors)
```

which is the equivalent of doing this:

```
rmap = ResponseErrorMap()
rmap.add(dict(status=401, json=dict(error='please authorize')),␣
↪UnauthorizedKeyError)
rmap.add(dict(status=500, json=dict(error='internal error')),␣
↪ServerOrConnectionError)
rmap.add(dict(body=requests.RequestException()), ServerOrConnectionError)
rmap.add(dict(body=ValueError()), BadApiResponse)
```

After (and **only** after) mappings have been added, the `make_error_tests()` can be called. E.g.:

```
rmap.make_error_tests(my_api.action, 'foo', 42, dict(user='Alice', pass='pass123
↪'))
```

This will:

- add all `response_args` entries with `responses.add(*args)`, and then
- call the `my_api.action()`-method with given arguments for each entry in the `ResponseErrorMap`, but in a `pytest.raises()`-context, like so:

```
for error in self.errors:
    with pytest.raises(error):
        my_api.action('foo', 42, dict(user='Alice', pass='pass123'))
```

Parameters

- **response_args** (*list of dictionaries or None*) – list of dictionaries that will be the arguments for the given test
- **errors** (*list or None*) – list of exceptions that should be raised given when the corresponding response from response_args is used.
- **common_kwargs** (*dict or None*) –

add(*response_kwargs*, *error*)

Parameters

- **response_kwargs** (*dict or None*) – a dictionary of arguments to `response.add()`
- **error** (*type(Exception)*) – the error that pytest should expect with `pytest.raises(error)`.

Returns None

CHAPTER 4

Testing

4.1 Pytest and tox

This project is set up to use pytest and tox, and all tests are in the */tests*-folder.

Note: pytest is NOT compatible with Python 3.3.

Warning: Using Anaconda and tox on Windows is likely to result in a "Error: InterpreterNotFound" message and fail. If that is the case, see *Working with Windows, conda, tox* on how to make it work.

4.1.1 Test in one environment

To run tests for the current Python environment, simply invoke pytest in project root, or pass `test` as an option argument to `setup.py`.

Examples:

```
C:\dev\mypolr> pytest
```

```
C:\dev\mypolr> python setup.py test
```

4.1.2 Multiple versions

Tests can be run for multiple python versions in separate *virtualenv*s using tox. Its setup is defined in the *tox.ini*, and will run tests in separate environments for:

- Python 2.7
- Python 3.4

- Python 3.5
- Python 3.6

These need to be created first using `virtualenv` or `conda`. (Keep reading.)

All versions

To run tests in all the Python environments, simply invoke tox in project root (after the Python environments are created).

Example:

```
C:\dev\mypolr> tox
```

Specific versions

To run on only a subset (or a single one) of the configured environments, you can use the `-e ENV[,ENV,...]` option.

E.g., to only run tests in environments with Python version 2.7 and 3.6:

```
C:\dev\mypolr>tox -e py27,py36
```

Read more about how to integrate tox and pytest.

4.1.3 Working with Windows, conda, tox

Note: Using `tox` and `conda` (miniconda or Anaconda) is at first glance **not a good match**, or at least on Windows. Tox expects to work from a *virtualenv* and not a *conda environment*. However, it's possible with a few, simple steps:

Read the tips below if you're using conda to manage Python environments **and** have problems with `InterpreterNotFound`-errors when attempting to run tox.

To use tox and conda on Windows, the following recommendations apply:

1. For tox to find your environments, consider to **either**:
   - Install environments in `C:\PythonXY`, where X and Y is major and minor version, respectively; **or**
   - Make a symlink from `C:\PythonXY` to your real path. E.g.: `mklink /J C:\Python27 C:\Anaconda3\envs\myPy27env`
2. In the environment where you call tox: install `virtualenv` with conda and *not with pip*. This seems to work well with Anaconda.

Tips 1: Use the `--prefix, -p`-option to define the location of new environments:

- Use `conda install -p C:\PythonXY python=X.Y` to create an environment called `PythonXY` in the location `C:\PythonXY`. (No symlink creation is needed.)
- Use `conda install -p C:\path\to\myenv python=X.Y` to create an environment called `myenv` in the location `C:\path\to\myenv`. (Symlinks should be made.)

Tips 2: Add the `--yes` option to prevent conda from asking confirmation upon creating environments.

Tips 3: If your Anaconda installation is on a different drive than C, e.g. *E:\Anaconda3*, then environments will be installed in the *E:\Anaconda3\envs*-directory if your current working drive is E. This allows you to create envs in the same drive as the rest of Anaconda without the need to use the `--prefix` option.

Fast and easy fix

The *tests/tox_with_conda.py*-file is a utility for making the steps above with a single call.

The `ToxEnvMatcher`-class can be used from Python to create environments and set up the needed symlinks, but it's also possible to use the file from command line.

Examples of use in Python:

```
my_envs = join('E:\\', 'Anaconda3', 'envs')
tem = ToxEnvMatcher(my_envs)
for version in '27,34,35,36'.split(','):
    tem.make(version)
```

Examples of use from cmd.exe:

```
E:\dev\mypolr\tests> tox_with_conda.py E:\Anaconda3\envs 27 34 35 36
```

Environment prefix (defaults to *py*) can be overridden with -p/–env_prefix options:

```
E:\dev\mypolr\tests> python tox_with_conda.py E:\Anaconda3\envs 27 34 35 36 -p Python
```

This will create new environments in `E:\Anaconda3\envs\PythonXY` instead of `E:\Anaconda3\envs\pyXY`

If, for some reason you need to, it's possible to use the `-b`/`--base` option to override the default base location (`C:\Python`):

```
E:\dev\mypolr\tests> tox_with_conda.py E:\Anaconda3\envs 27 34 35 36 --base D:\Python
```

Note: The *tox_with_conda.py*-file has been uploaded to a repository of its own on https://github.com/fauskanger/tox_with_conda and can also be installed with pip:

```
pip install tox_with_conda
```

If installed with pip, then instead of

```
python tox_with_conda.py ...
```

use

```
python -m tox_with_conda ...
```

4.2 Travis CI

Current build and test status:

The *.travis.yml*-file defines the Travis CI setup for this project. When new code has been pushed to the git repository, Travis CI will automatically pull the updates. Then it will build and run tests for multiple versions of Python. The process can be monitored here.

Warning: Travis continuous integration is not a replacement for running tests locally before committing changes or making pull requests.

CHAPTER 5

License

MIT License

Copyright (c) 2017 Thomas Fauskanger

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

t

Index