Wespe Documentation

Release 0.2.4

Lucas Lira Gomes

Sep 15, 2020

Contents

CHAPTER 1

Wespe - Batching ad tech providers' operations for humans

1.1 Abstract

What is *Wespe*?

Wespe is a Python API for batching requests when interfacing AdTech providers (e.g. adwords, facebook business). The motivation behind *Wespe* is to provide a simple and consistent interface for batching requests. Currently it only supports Facebook Business. Other providers will be added in the future.

Read the docs: http://wespe.readthedocs.io/en/latest/

1.2 Installation

Wespe supports python `3.6+`. It may also work on pypy, cython, and jython, but is not being tested for these versions.

To install *Wespe* run the following command:

```
pip install wespe
```

1.3 Usage - Facebook Business

All steps from now on will assume you've already set the default api connection using facebook_busines. It's also possible to set one on the fly by providing the api kwarg in FacebookBatchUploader's constructor.

```
from wespe.batch_uploaders import FacebookBatchUploader

# There is no request limit. If necessary Wespe will coordinate the execution of␣
↪multiple FacebookAdsApiBatch
```

(continues on next page)

(continued from previous page)

```
# instances.
batch_uploader = FacebookBatchUploader(requests)

try:
    batch_uploader.execute()
except BatchExecutionError:
    for error in batch_uploader.errors:
        # See FacebookBatchRequestError for more info on what you can do
        pass

for response in batch_uploader.responses:
    # See FacebookBatchResponse for more info on what you can do
    pass
```

1.4 License

Crafted with in Berlin.

CHAPTER 2

Installation

2.1 Stable release

To install Wespe, run this command in your terminal:

```
$ pip install wespe
```

This is the preferred method to install Wespe, 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 Wespe can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/x8lucas8x/wespe
```

Or download the tarball:

```
$ curl  -OL https://github.com/kayak/wespe/tarball/master
```

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

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

CHAPTER 3

Usage - Facebook Business

All steps from now on will assume you've already set the default api connection using facebook_busines. It's also possible to set one on the fly by providing the api kwarg in FacebookBatchUploader's constructor.

```
from wespe.batch_uploaders import FacebookBatchUploader

# There is no request limit. If necessary Wespe will coordinate the execution of
↪multiple FacebookAdsApiBatch
# instances.
batch_uploader = FacebookBatchUploader(requests)

try:
    batch_uploader.execute()
except BatchExecutionError:
    for error in batch_uploader.errors:
        # See FacebookBatchRequestError for more info on what you can do
        pass

for response in batch_uploader.responses:
    # See FacebookBatchResponse for more info on what you can do
    pass
```

CHAPTER 4

wespe

4.1 wespe package

4.1.1 Subpackages

wespe.batch_uploaders package

Submodules

wespe.batch_uploaders.facebook module

wespe.batch_uploaders.requests module

class wespe.batch_uploaders.requests.**BaseRequestError**(*description: str*, *is_transient: bool*, *data: dict*)

Bases: object

data

Returns the entire error payload response.

Returns a dictionary.

description

Returns an error description.

Returns a string.

is_transient

Returns True when the error is likely to be caused by a temporary issue (e.g. network issue). This should be used as a way to identify which requests can be retried without further modifications.

Returns a boolean.

class wespe.batch_uploaders.requests.**BaseResponse**(*data: dict*)
Bases: object

data
Returns the entire payload response.

Returns a dictionary.

wespe.batch_uploaders.retries module

Module contents

4.1.2 Submodules

4.1.3 wespe.exceptions module

exception wespe.exceptions.**BaseException**
Bases: Exception

All errors specific to to this library will be subclassed from BaseException.

exception wespe.exceptions.**BatchExecutionError**
Bases: OSError, *wespe.exceptions.BaseException*

exception wespe.exceptions.**InvalidValueError**
Bases: ValueError, *wespe.exceptions.BaseException*

exception wespe.exceptions.**NoFacebookRequestProvidedError**
Bases: *wespe.exceptions.InvalidValueError*

exception wespe.exceptions.**TooManyRequestsPerBatchError**
Bases: *wespe.exceptions.InvalidValueError*

4.1.4 Module contents

Top-level package for Wespe.

CHAPTER 5

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:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/kayak/wespe/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.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/kayak/wespe/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 :)

5.2 Get Started!

Ready to contribute? Here's how to set up *wespe* for local development.

1. Fork the *wespe* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/wespe.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 wespe
$ cd wespe/
$ python setup.py develop
```

4. Install the commit hooks, which includes an auto formatter:

```
$ pre-commit install
```

5. Create a branch for local development:

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

   Now you can make your changes locally.

6. When you're done making changes, check that your changes pass for all python versions that we support, including flake8 rules, by running tox:

```
$ tox
```

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.

5.3 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 3.4, 3.5 and 3.6. Check https://travis-ci.org/x8lucas8x/wespe/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_wespe
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed. Then run:

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

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Python Module Index

W

Index