Diablo 3 API Documentation

Release 0.1.0

David Lewis

December 27, 2015

Contents

Contents:

CHAPTER 1

Diablo 3 API

A Python API Wrapper for the Diablo 3 API

- Free software: BSD license
- Documentation: https://diablo3api.readthedocs.org.

1.1 Features

- Provides an API wrapper for the current Diablo 3 API Resources.
- Works with Python 2.6, 2.7, 3.3, 3.4.

CHAPTER 2

Installation

At the command line:

```
$ easy_install diablo3api
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv diablo3api
$ pip install diablo3api
```

CHAPTER 3

Usage

To use Diablo 3 API in a project:

```
from diablo3api import Diablo3API

api = Diablo3API()
api.profile.get("battletag-1234")
```

CHAPTER 4

Resources

The following Resources are supported by the Diablo3API wrapper

4.1 Career Profile

The Profile resource takes one argument, the Battletag of the user in the form of Battletag-1234

```
from diablo3api import Diablo3API

api = Diablo3API()
api.profile.get("battletag-1234")
```

4.2 Hero Profile

The Hero Profile resource takes 2 arguments, the Battletag and the Hero ID. The Career Profile resource will list all of the heroes for a given profile.

```
from diablo3api import Diablo3API

api = Diablo3API()
api.profile.hero.get("battletag-1234", 5678)
```

4.3 Item Information

The Item Information resource takes 1 argument, the Item ID. The Hero Profile will list all of the items equipped by a Hero.

```
from diablo3api import Diablo3API

api = Diablo3API()
api.item.get("item-id")
```

4.4 Follower Information

The Follower Information resource takes 1 argument, the Follower Type. The only valid Follower types are "templar", "scoundrel", and "enchantress".

```
from diablo3api import Diablo3API

api = Diablo3API()
api.follower.get("templar")
```

Alternatively there are specific methods for each of the followers.

```
api.follower.templar()
api.follower.scoundrel()
api.follower.enchantress()
```

4.5 Artisan Information

The Artisan Information resource takes 1 argument, the Artisan Type. The only valid Artisan types are "blacksmith", "jeweler", and "mystic".

```
from diablo3api import Diablo3API

api = Diablo3API()
api.artisan.get("blacksmith")
```

Alternatively there are specific methods for each of the artisans.

```
api.artisan.blacksmith()
api.artisan.jeweler()
api.artisan.mystic()
```

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/menglewis/diablo3api/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" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

5.1.4 Write Documentation

Diablo 3 API could always use more documentation, whether as part of the official Diablo 3 API 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/menglewis/diablo3api/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 *diablo3api* for local development.

1. Fork the *diablo3api* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/diablo3api.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 diablo3api
$ cd diablo3api/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b 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 flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 diablo3api tests
$ python setup.py test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

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 2.6, 2.7, 3.3, and 3.4, and for PyPy. Check https://travis-ci.org/menglewis/diablo3api/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_diablo3api
```

CHAPTER 6

Credits

6.1 Development Lead

- David Lewis <meng.lewis@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

CHAPTER 8

0.1.0 (2014-12-15)

- First release on PyPI.

CHAPTER 9

Indices and tables

- genindex
- modindex
- search