Python Boilerplate Documentation

Release 0.1

Ricardo Bánffy

January 26, 2017

Contents

Contents:

CHAPTER 1

appengine-fixture-loader

A simple way to load Django-like fixtures into the local development datastore, originally intended to be used by testable_appengine.

1.1 Installing

For the less adventurous, Appengine-Fixture-Loader is available on PyPI at https://pypi.python.org/pypi/Appengine-Fixture-Loader.

1.2 Single-kind loads

Let's say you have a model like this:

```
class Person(ndb.Model):
    """Our sample class"""
    first_name = ndb.StringProperty()
    last_name = ndb.StringProperty()
    born = ndb.DateTimeProperty()
    userid = ndb.IntegerProperty()
    thermostat_set_to = ndb.FloatProperty()
    snores = ndb.BooleanProperty()
    started_school = ndb.DateProperty()
    sleeptime = ndb.TimeProperty()
    favorite_movies = ndb.JsonProperty()
    processed = ndb.BooleanProperty(default=False)
```

If you want to load a data file like this:

```
[
    {
        "__id__": "jdoe",
        "born": "1968-03-03T00:00:00",
        "first_name": "John",
        "last_name": "Doe",
        "favorite_movies": [
            "2001",
            "The Day The Earth Stood Still (1951)"
        ],
        "snores": false,
        "sleeptime": "23:00",
```

```
        "started_school": "1974-02-15",
        "thermostat_set_to": 18.34,
        "userid": 1
    },

...

    {
        "born": "1980-05-25T00:00:00",
        "first_name": "Bob",
        "last_name": "Schneier",
        "favorite_movies": [
            "2001",
            "Superman"
        ],
        "snores": true,
        "sleeptime": "22:00",
        "started_school": "1985-08-01",
        "thermostat_set_to": 18.34,
        "userid": -5
    }
]
```

All you need to do is to:

```
from appengine_fixture_loader.loader import load_fixture
```

and then:

```
loaded_data = load_fixture('tests/persons.json', kind=Person)
```

In our example, *loaded_data* will contain a list of already persisted Person models you can then manipulate and persist again.

The *__id__* attribute, when defined, will save the object with that given id. In our case, the key to the first object defined will be a *ndb.Key('Person', 'jdoe')*. The key may be defined on an object by object base - where the *__id__* parameter is omitted, an automatic id will be generated - the key to the second one will be something like *ndb.Key('Person', 1)*.

1.3 Multi-kind loads

It's convenient to be able to load multiple kinds of objects from a single file. For those cases, we provide a simple way to identify the kind of object being loaded and to provide a set of models to use when loading the objects.

Consider our original example model:

```
class Person(ndb.Model):
    """Our sample class"""
    first_name = ndb.StringProperty()
    last_name = ndb.StringProperty()
    born = ndb.DateTimeProperty()
    userid = ndb.IntegerProperty()
    thermostat_set_to = ndb.FloatProperty()
    snores = ndb.BooleanProperty()
    started_school = ndb.DateProperty()
    sleeptime = ndb.TimeProperty()
    favorite_movies = ndb.JsonProperty()
    processed = ndb.BooleanProperty(default=False)
```

and let's add a second one:

```
class Dog(ndb.Model):
    """Another sample class"""
    name = ndb.StringProperty()
```

Now, if we wanted to make a single file load objects of the two kinds, we'd need to use the *__kind__* attribute in the JSON:

```
[
    {
        "__kind__": "Person",
        "born": "1968-03-03T00:00:00",
        "first_name": "John",
        "last_name": "Doe",
        "favorite_movies": [
            "2001",
            "The Day The Earth Stood Still (1951)"
        ],
        "snores": false,
        "sleeptime": "23:00",
        "started_school": "1974-02-15",
        "thermostat_set_to": 18.34,
        "userid": 1
    },
    {
        "__kind__": "Dog",
        "name": "Fido"
    }
]
```

And, to load the file, we'd have to:

```
from appengine_fixture_loader.loader import load_fixture
```

and:

```
loaded_data = load_fixture('tests/persons_and_dogs.json',
                           kind={'Person': Person, 'Dog': Dog})
```

will result in a list of Persons and Dogs (in this case, one person and one dog).

1.4 Multi-kind, multi-level loads

Anther common case is having hierarchies of entities that you want to reconstruct for your tests.

Using slightly modified versions of our example classes:

```
class Person(ndb.Model):
    """Our sample class"""
    first_name = ndb.StringProperty()
    last_name = ndb.StringProperty()
    born = ndb.DateTimeProperty()
    userid = ndb.IntegerProperty()
    thermostat_set_to = ndb.FloatProperty()
    snores = ndb.BooleanProperty()
    started_school = ndb.DateProperty()
    sleeptime = ndb.TimeProperty()
```

```
    favorite_movies = ndb.JsonProperty()
    processed = ndb.BooleanProperty(default=False)
    appropriate_adult = ndb.KeyProperty()
```

and:

```
class Dog(ndb.Model):
    """Another sample class"""
    name = ndb.StringProperty()
    processed = ndb.BooleanProperty(default=False)
    owner = ndb.KeyProperty()
```

And using *__children__[attribute_name]__* like meta-attributes, as in:

```
[
    {
        "__kind__": "Person",
        "born": "1968-03-03T00:00:00",
        "first_name": "John",
        "last_name": "Doe",

        ...

        "__children__appropriate_adult__": [
            {
                "__kind__": "Person",
                "born": "1970-04-27T00:00:00",

                ...

                "__children__appropriate_adult__": [
                    {
                        "__kind__": "Person",
                        "born": "1980-05-25T00:00:00",
                        "first_name": "Bob",

                        ...

                        "userid": 3
                    }
                ]
            }
        ]
    },
    {
        "__kind__": "Person",
        "born": "1999-09-19T00:00:00",
        "first_name": "Alice",

        ...

        "__children__appropriate_adult__": [
            {
                "__kind__": "Person",

                ...

                "__children__owner__": [
                    {
```

```
                        "__kind__": "Dog",
                        "name": "Fido"
                    }
                ]
            }
        ]
    }
]
```

you can reconstruct entire entity trees for your tests.

1.5 Parent/Ancestor-based relationships with automatic keys

It's also possible to set the *parent* by using the *__children__* attribute.

For our example classes, importing:

```
[
    {
        "__kind__": "Person",
        "first_name": "Alice",

        ...

        "__children__": [
            {
                "__kind__": "Person",
                "first_name": "Bob",
                ...

                "__children__owner__": [
                    {
                        "__kind__": "Dog",
                        "name": "Fido"
                    }
                ]
            }
        ]
    }
]
```

should be equivalent to:

```
alice = Person(first_name='Alice')
alice.put()
bob = Person(first_name='Bob', parent=alice)
bob.put()
fido = Dog(name='Fido', parent=bob)
fido.put()
```

You can then retrieve fido with:

```
fido = Dog.query(ancestor=alice.key).get()
```

CHAPTER 2

Development

There are two recommended ways to work on this codebase. If you want to keep one and only one App Engine SDK install, you may clone the repository and run the tests by:

```
$ PYTHONPATH=path/to/appengine/library python setup.py test
```

Alternatively, this project contains code and support files derived from the testable_appengine project. Testable_appengine was conceived to make it easier to write (and run) tests for Google App Engine applications and to hook your application to Travis CI. In essence, it creates a virtualenv and downloads the most up-to-date SDK and other support tools into it. To use it, you run *make*. Calling *make help* will give you a quick list of available make targets:

```
$ make venv
Running virtualenv with interpreter /usr/bin/python2
New python executable in /export/home/ricardo/projects/appengine-fixture-loader/.env/bin/python2
Also creating executable in /export/home/ricardo/projects/appengine-fixture-loader/.env/bin/python
(...)
'/export/home/ricardo/projects/appengine-fixture-loader/.env/bin/run_tests.py' -> '/export/home/rica
'/export/home/ricardo/projects/appengine-fixture-loader/.env/bin/wrapper_util.py' -> '/export/home/r
$ source .env/bin/activate
(.env) $ nosetests
..............
----------------------------------------------------------------------
Ran 14 tests in 2.708s

OK
```

CHAPTER 3

Installation

At the command line:

```
$ easy_install appengine_fixture_loader
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv appengine_fixture_loader
$ pip install appengine_fixture_loader
```

CHAPTER 4

Usage

To use App Engine Fixture Loader in a project:

```
import appengine_fixture_loader
```

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/rbanffy/appengine-fixture-loader/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

App Engine Fixture Loader could always use more documentation, whether as part of the official App Engine Fixture Loader 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/rbanffy/appengine-fixture-loader/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 *appengine-fixture-loader* for local development.

1. Fork the *appengine-fixture-loader* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/appengine-fixture-loader.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 appengine-fixture-loader
$ cd appengine-fixture-loader/
$ 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 appengine-fixture-loader 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/rbanffy/appengine-fixture-loader/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_appengine-fixture-loader
```

CHAPTER 6

Credits

6.1 Development Lead

- Ricardo Bánffy <rbanffy@gmail.com>

6.2 Contributors

John Del Rosario (john2x)

CHAPTER 7

History

CHAPTER 8

0.1.0 (2014-10-13)

- First release on GitHub.

CHAPTER 9

0.1.1 (2014-12-4)

- Add support for multi-kind JSON files

CHAPTER 10

0.1.2 (2014-12-4)

- Minor fixes

0.1.3 (2014-12-5)

- Added support for PropertyKey-based child entities

CHAPTER 12

0.1.4 (2015-2-4)

- Fixed bug in which post-processor was called on every property change
- Added section on development to README.rst

CHAPTER 13

0.1.5 (2015-2-11)

- Added __*children*__ support
- Added manual key definition through the __*id*__ attribute

CHAPTER 14

0.1.6 (2015-8-30)

- Builds if you don't have *curl* installed
- Minor documentation improvements

CHAPTER 15

0.1.7 (2015-11-3)

- Syntax highlighting on the documentation
- Coverage analysis using Coveralls

CHAPTER 16

0.1.8 (2016-02-05)

- New resources/Makefile

CHAPTER 17

0.1.9 (2016-12-19)

- Replace pep8 with pycodestyle
- Update current SDK version detection to latest version

CHAPTER 18

Indices and tables

- genindex
- modindex
- search