Flatson Documentation

Release 0.1.0

ScrapingHub

June 09, 2016

Contents

1 Flatson **3**
1.1 Features . . . 3
1.2 Next Steps . . . 4

2 Installation **5**

3 Using Flatson **7**
3.1 Walk-through with an example . . . 7
3.2 Array serialization . . . 8

4 Contributing **9**
4.1 Types of Contributions . . . 9
4.2 Get Started! . . . 10
4.3 Pull Request Guidelines . . . 10
4.4 Tips . . . 11

5 Credits **13**
5.1 Development Lead . . . 13
5.2 Contributors . . . 13

6 History **15**

7 0.1.0 (2015-09-25) **17**

8 Indices and tables **19**

Contents:

CHAPTER 1

Flatson

Flatson emerged at Scrapinghub from the need to export huge JSON-like datasets into **flat** CSV-like tables. Flatson is particularly useful to handle really huge datasets, because it doesn't load all the data in memory at once.

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

1.1 Features

- Flattens Python dictionaries using a JSON schema
- Supports per-field configuration via the schema

Usage:

```
>>> from flatson import Flatson
>>> schema = {
        "$schema": "http://json-schema.org/draft-04/schema",
        "type": "object",
        "properties": {
            "name": {"type": "string"},
            "age": {"type": "number"},
            "address": {
                "type": "object",
                "properties": {"city": {"type": "string"}, "street": {"type": "string"}}
            },
            "skills": {"type": "array", "items": {"type": "string"}}
        }
    }
>>> sample = {
            "name": "Claudio", "age": 42,
            "address": {"city": "Paris", "street": "Rue de Sevres"},
            "skills": ["hacking", "soccer"]}
>>> f = Flatson(schema)
>>> f.fieldnames
['address.city', 'address.street', 'age', 'name', 'skills']
>>> f.flatten(sample)
['Paris', 'Rue de Sevres', 42, 'Claudio', '["hacking","soccer"]']
```

You can get a dict with the field names order preserved:

```
>>> f.flatten_dict(sample)
OrderedDict([('address.city', 'Paris'), ('address.street', 'Rue de Sevres'), ('age', 42), ('name', '
```

You can also configure array serialization behavior through the schema (default JSON):

```
>>> schema = {
        "$schema": "http://json-schema.org/draft-04/schema",
        "type": "object",
        "properties": {
            "name": {"type": "string"},
            "skills": {
                "type": "array",
                "items": {"type": "string"},
                "flatson_serialize": {"method": "join_values"},
            }
        }
    }
>>> f = Flatson(schema)
>>> f.flatten({"name": "Salazar", "skills": ["hacking", "socker", "partying"]})
['Salazar', 'hacking,socker,partying']
```

1.2 Next Steps

Read more on *how to use Flatson*, check out the Github Repo and feel free to send Issues or PRs. =)

CHAPTER 2

Installation

At the command line:

```
$ easy_install flatson
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv flatson
$ pip install flatson
```

CHAPTER 3

Using Flatson

Using Flatson is simple, you just need some JSON data to flatten and its JSON schema. Flatson will use the information from the schema to understand the structure of your object, which makes the flattening easier and more predictable.

Note: If you don't have the JSON schema for the data you want to flatten, you can use a tool to generate a JSON schema for your data, like Skinfer or http://jsonschema.net.

3.1 Walk-through with an example

Say you have the following JSON schema in a file named `schemafile.json`:

```
{
    "$schema": "http://json-schema.org/draft-04/schema",
    "type": "object",
    "properties": {
        "name": {"type": "string"},
        "age": {"type": "number"},
        "address": {
            "type": "object",
            "properties": {"city": {"type": "string"}, "street": {"type": "string"}}
        },
        "skills": {"type": "array", "items": {"type": "string"}}
    }
}
```

You can instantiate the `Flatson` class for this schemafile like this:

```
>>> from flatson import Flatson
>>> f = Flatson.fromschemafile('schemafile.json')
>>> f.fieldnames
['address.city', 'address.street', 'age', 'name', 'skills']
```

Note how Flatson has inferred the flattened field names, which you can access through the `fieldnames` property.

Let's test it with some sample data:

```
>>> sample = {
            "name": "Claudio", "age": 42,
            "address": {"city": "Paris", "street": "Rue de Sevres"},
            "skills": ["hacking", "soccer"]}
```

```
>>> f.flatten(sample)
['Paris', 'Rue de Sevres', 42, 'Claudio', '["hacking","soccer"]']
```

There are a couple of things to note here:

1. the `flatten()` method simply returns a list of simple objects
2. the array is by default serialized as a JSON string

Note: Array serialization is *a topic apart*, for now it suffices to say that if you don't like this default behavior, there are other options you can configure through the schema, you can even register your own serialization methods if you like.

Say you actually want a Python dict instead of a list, no worries, just use `flatten_dict()`:

```
>>> f.flatten_dict(sample)
OrderedDict([('address.city', 'Paris'), ('address.street', 'Rue de Sevres'), ('age', 42), ('name', 'C
```

Note that this returns an OrderedDict instead of a traditional Python dict: this has the advantage of preserving the same field ordering of the the list returned by the `flatten()` method.

3.2 Array serialization

TODO: write about array serialization here, point to design decisions, list available methods

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/scrapinghub/flatson/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

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

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/flatson.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 flatson
$ cd flatson/
$ 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 flatson 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.

4.3 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/scrapinghub/flatson/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

CHAPTER 5

Credits

5.1 Development Lead

- ScrapingHub <info@scrapinghub.com>

5.2 Contributors

- Elias Dorneles <elias@scrapinghub.com>
- Bernardo Botella <bernardo@scrapinghub.com>

CHAPTER 6

History

CHAPTER 7

0.1.0 (2015-09-25)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search