fwissr Documentation

Release 1.0.5

Pierre Baillet

December 05, 2013

Contents

Contents:

CHAPTER 1

fwissr

fwissr-python is a registry configuration tool, compatible with fwissr, its ruby counterpart. Made by fotonauts.

- Free software: MIT license
- Documentation: http://fwissr.rtfd.org.

1.1 Install

```
$ [sudo] pip install fwissr
```

1.2 Usage

Create the main `fwissr.json` configuration file in either `/etc/fwissr/` or `~/.fwissr/` directory:

```
{
  "foo" : "bar",
  "horn" : { "loud" : true, "sounds": [ "TUuuUuuuu", "tiiiiiiIIiii" ] }
}
```

In your application, you can access `fwissr`'s global registry that way:

```
from fwissr.fwissr import Fwissr
Fwissr['/foo']
# u'bar'
Fwissr['/horn']
# {u'sounds': [u'TUuuUuuuu', u'tiiiiiiIIiii'], u'loud': True}
Fwissr['/horn/loud']
# True
Fwissr['/horn/sounds']
# [u'TUuuUuuuu', u'tiiiiiiIIiii']
```

In bash you can call the *fwissr* tool:

```
$ fwissr /foo
bar

# json output
```

```
$ fwissr -j /horn
{"loud": true, "sounds": ["TUuuUuuuu", "tiiiiiiIIiii"]}

# pretty print json output
$ fwissr -j -p /horn
{
    "loud": true,
    "sounds": [
        "TUuuUuuuu",
        "tiiiiiiIIiii"
    ]
}

# dump all registry with pretty print json output
# NOTE: yes, that's the same as 'fwissr -jp /'
$ fwissr --dump -jp
{
  "horn": {
    "loud": true,
    "sound": [
      "TUuuUuuuu",
      "tiiiiiiIIiii"
    ]
  }
}
```

1.3 Additional configuration file

Provide additional configuration files with the `fwissr_sources` setting in `fwissr.json`:

```
{
  "fwissr_sources": [
    { "filepath": "/etc/my_app.json" }
  ]
}
```

The settings for that configuration will be prefixed with the file name.

For example, with that `/etc/my_app.json`:

```
{ "foo": "bar", "bar": "baz" }
```

the settings can be accessed that way:

```
from fwissr.fwissr import Fwissr

Fwissr['/my_app']
#=> { "foo": "bar", "bar": "baz" }

Fwissr['/my_app/foo']
#=> "bar"

Fwissr['/my_app/bar']
#=> "baz"
```

You can bypass that behaviour with the `top_level` setting:

```
{
  "fwissr_sources": [
    { "filepath": "/etc/my_app.json", "top_level": true }
  ]
}
```

With the `top_level` setting activated the configuration settings are added to registry root:

```
from fwissr.fwissr import Fwissr

Fwissr['/']
#=> { "foo": "bar", "bar": "baz" }

Fwissr['/foo']
#=> "bar"

Fwissr['/bar']
#=> "baz"
```

Note that you can provide `.json` and `.yaml` configuration files.

1.4 Directory of configuration files

If the `filepath` setting is a directory, then all `.json` and `.yaml` files in that directory (but NOT in subdirectories) will be imported in the global registry:

```
{
  "fwissr_sources": [
    { "filepath": "/mnt/my_app/conf/" },
  ],
}
```

With `/mnt/my_app/conf/database.yaml`:

```
production:
  adapter: mysql2
  encoding: utf8
  database: my_app_db
  username: my_app_user
  password: my_app_pass
  host: db.my_app.com
```

and `/mnt/my_app/conf/credentials.json`:

```
{ "key": "i5qw64816c", "code": "448e4wef161" }
```

the settings can be accessed that way:

```
from fwissr.fwissr import Fwissr

Fwissr['/database']
#=> { "production": { "adapter": "mysql2", "encoding": "utf8", "database": "my_app_db", "username":

Fwissr['/database/production/host']
#=> "db.my_app.com"

Fwissr['/credentials']
#=> { "key": "i5qw64816c", "code": "448e4wef161" }
```

```
Fwissr['/credentials/key']
#=> "i5qw64816c"
```

1.5 File name mapping to setting path

Use dots in file name to define a path for configuration settings.

For example:

```
{
  "fwissr_sources": [
    { "filepath": "/etc/my_app.database.slave.json" }
  ]
}
```

with that `/etc/my_app.database.slave.json`:

```
{ "host": "db.my_app.com", "port": "1337" }
```

The settings can be accessed that way:

```
from fwissr.fwissr import Fwissr

Fwissr['/my_app/database/slave/host']
#=> "db.my_app.com"

Fwissr['/my_app/database/slave/port']
#=> "1337"
```

1.6 Mongodb source

You can define a mongob collection as a configuration source:

```
{
  "fwissr_sources": [
    { "mongodb": "mongodb://db1.example.net/my_app", "collection": "config" }
  ]
}
```

Each document in the collection is a setting for that configuration.

The `_id` document field is the setting key, and the `value` document field is the setting value.

For example:

```
> db["my_app.stuff"].find()
{ "_id" : "foo", "value" : "bar" }
{ "_id" : "database", "value" : { "host": "db.my_app.com", "port": "1337" } }
```

```
from fwissr.fwissr import Fwissr

Fwissr['/my_app/stuff/foo']
#=> "bar"

Fwissr['/my_app/stuff/database']
```

```
#=> { "host": "db.my_app.com", "port": "1337" }

Fwissr['/my_app/stuff/database/port']
#=> "1337"
```

As with configuration files you can use dots in collection name to define a path for configuration settings. The `top_level` setting is also supported to bypass that behaviour. Note too that the `fwissr` collection is by default a `top_level` configuration (as the `/etc/fwissr/fwissr.json` configuration file).

1.7 Refreshing registry

Enable registry auto-update with the *refresh* source setting.

For example:

```
{
  "fwissr_sources": [
    { "filepath": "/etc/my_app/my_app.json" },
    { "filepath": "/etc/my_app/stuff.json", "refresh": true },
    { "mongodb": "mongodb://db1.example.net/my_app", "collection": "production" },
    { "mongodb": "mongodb://db1.example.net/my_app", "collection": "config", "refresh": true }
  ]
}
```

The `/etc/my_app/my_app.json` configuration file and the `production` mongodb collection are read only once, whereas the settings holded by the `/etc/my_app/stuff.json` configuration file and the `config` mongodb collection are expired periodically and re-fetched.

The default freshness is 30 seconds, but you can change it with the `fwissr_refresh_period` setting:

```
{
  "fwissr_sources": [
    { "filepath": "/etc/my_app/my_app.json" },
    { "filepath": "/etc/my_app/stuff.json", "refresh": true },
    { "mongodb": "mongodb://db1.example.net/my_app", "collection": "production" },
    { "mongodb": "mongodb://db1.example.net/my_app", "collection": "config", "refresh": true }
   ],
  "fwissr_refresh_period": 60
}
```

The refresh is done periodically in a thread:

```
from fwissr.fwissr import Fwissr
import time

Fwissr['/stuff/foo']
#=> "bar"

# > Change '/etc/my_app/stuff.json' file by setting: {"foo":"baz"}

# Wait 2 minutes
time.sleep(120)

# The new value is now in the registry
Fwissr['/stuff/foo']
#=> "baz"
```

1.8 Create a custom registry

`fwissr` is intended to be easy to setup: just create a configuration file and that configuration is accessible via the global registry. But if you need to, you can create your own custom registry:

```
from fwissr.fwissr import Fwissr
from fwissr.registry import Registry
from fwissr.source.source import Source
# create a custom registry
registry = Registry(refresh_period=20)

# add configuration sources to registry
registry.add_source(Source.from_settings({ 'filepath': '/etc/my_app/my_app.json' }))
registry.add_source(Source.from_settings({ 'filepath': '/etc/my_app/stuff.json', 'refresh': true }))
registry.add_source(Source.from_settings({ 'mongodb': 'mongodb://db1.example.net/my_app', 'collectio
registry.add_source(Source.from_settings({ 'mongodb': 'mongodb://db1.example.net/my_app', 'collectio

registry['/stuff/foo']
#=> 'bar'
```

1.9 Create a custom source

Currently `fwissr.source.file.File` and `fwissr.source.mongodb.Mongodb` are the two kinds of possible registry sources, but you can define your own source:

TODO

1.10 Credits

The Fotonauts team: http://www.fotopedia.com

CHAPTER 2

Installation

At the command line:

```
$ easy_install fwissr
```

Or, if you have virtualenvwrapper installed:

```
$ mkvirtualenv fwissr
$ pip install fwissr
```

CHAPTER 3

Usage

To use fwissr in a project:

```
import fwissr
```

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/fotonauts/fwissr-python/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

fwissr could always use more documentation, whether as part of the official fwissr 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/fotonauts/fwissr-python/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 *fwissr-python* for local development.

1. Fork the *fwissr-python* repo on GitHub.
2. Clone your fork locally:

   ```
   $ git clone git@github.com:your_name_here/fwissr-python.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 fwissr-python
   $ cd fwissr-python/
   $ 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 fwissr 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, and 3.3, and for PyPy. Check https://travis-ci.org/fotonauts/fwissr-python/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_fwissr
```

CHAPTER 5

Credits

5.1 Development Lead

- Aymerick Jéhanne <aymerick@jehanne.org>

5.2 Contributors

- Pierre Baillet <pierre@baillet.name>: Port to python

History

6.1 1.0.5 (2013-12-05)

- Command line behaves as ruby counterpart
- Fix for broken url parsing in Python for MongoDB urls with a query parameter

6.2 1.0.0 (2013-12-02)

- First public release on Github

6.3 0.1.1 (2013-11-04)

- Dictionary provided by API is now Read only

6.4 0.1.0 (2013-11-01)

- First release on PyPI.

CHAPTER 7

Indices and tables

- *genindex*
- *modindex*
- *search*