epochs Documentation

Release 0.2.0

Michael Galloy

Feb 28, 2021

Contents:

CHAPTER 1

epochs

Python package to handle configuration files specifying values changing over time

- Free software: BSD license
- Documentation: https://epochs.readthedocs.io.

1.1 Features

The main features of epochs are:

- Parse date-based configuration files and retrieve values based on datetime.
- Validate configuration files, both normal and epoch date-based ones, against a specification.

For example, for a configuration file, `epochs.cfg`, such as:

```
[2019-04-09 20:27:15]
value   : 3

[2019-04-09 22:31:01]
value   : 5
```

The dates can be anything parsed by dateutil.parser.parse. Then, epochs can retrieve the correct value from the config file corresponding to a given date:

```
>>> import epochs
>>> ep = epochs.parse('epochs.cfg')
>>> value = ep.get('value', datetime='2019-04-09 21:55:45')
>>> print(value)
```

(continues on next page)

(continued from previous page)

```
3
>>> value = ep.get('value', datetime='2019-04-09 23:15:40')
>>> print(value)
5
```

The "correct" value is the one specified in the earliest section of the configuration file with a date on or before the given date.

Below is an example specification for a configuration file:

```
[city]
name    : required=True, type=str
streets : required=True, type=List[str]
temp    : required=False, type=float, default=0.0
```

And an example configuration file following this specification:

```
[city]
name    : Boulder
streets : [Broadway, Baseline, Valmont]
```

Then to parse the configuration file with its specification:

```
>>> import epochs
>>> cf = epochs.parse('example.cfg', spec='spec.cfg')
>>> name = cf.get('name', section='city')
>>> print(name)
Boulder
>>> streets = cf.get('streets', section='city')
>>> print(steets)
['Broadway', 'Baseline', 'Valmont']
>>> temp = cf.get('temp', section='city')
>>> print(temp, type(temp))
0.0 <class 'float'>
```

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install epochs, run this command in your terminal:

```
$ pip install epochs
```

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

You can either clone the public repository:

```
$ git clone git://github.com/mgalloy/epochs
```

Or download the tarball:

```
$ curl  -OL https://github.com/mgalloy/epochs/tarball/master
```

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

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

CHAPTER 3

Usage

To use epochs in a project:

```
import epochs
```

CHAPTER 4

Specification files

Specification files provide a specification for each option in a configuration file. This specification allows:

1. the `get` method to automatically return the value as the correct type
2. the `get` method to return a default value if an optional value was not given in the configuration file
3. the configuration file(s) to be validated against the specification to ensure all required options are provided and no extra options are given.

For example, a specification file might look like the following:

```
[logging]
basedir       : type=str
level         : type=str, default=DEBUG
rotate        : type=bool, default=YES
max_version   : type=int, default=9
max_width     : type=int, default=90

[level1]
wavelengths   : type=List[float], default=[]
wavetypes     : type=List[str], default="[1074, 1079, 1083]"
```

The configuration file that uses this specification might be like:

```
[logging]
basedir       : /Users/mgalloy/data
level         : DEBUG
rotate        : NO
max_version   : 3

[level1]
wavelengths   : [1074.7, 1079.8, 1083.0]
```

The possible attributes for an option are listed below.

required whether the option is required; allowed values (case-insensitive) for required are “True”, “YES”, or “1” and “False”, “NO”, or “0” for not required

type Python type: `str` (default), `int`, `float`, `bool`/`boolean` or `List[]` of one of the scalar types i.e., `List[int]`

default default value if the option is not specified

Specification files for epoch configuration files use only the `DEFAULT` section whereas specification files for standard configuration files mirror the same sections as their configuration files.

CHAPTER 5

epochs

5.1 epochs package

5.1.1 Submodules

5.1.2 epochs.configparser module

Module defining parsers. The `ConfigParser` is an extended `configparser` that can use a specification file to define types/default values for the config files options, as well as to validate the file against. The `EpochParser` is a parser which organizes options by date. An option value is found by matching the option in the section with the latest datetime before the given datetime.

class `epochs.configparser.`**ConfigParser**(*spec_filename: str = None*, ***kwargs*)
Bases: `configparser.ConfigParser`

ConfigParser subclass which can verify a config file against a specification and uses types/defaults from the specification.

get(*section: str*, *option: str*, *raw: bool = False*, *use_spec: bool = True*, ***kwargs*) → OptionValue
Get an option using the type and default from the specification file

Parameters

- **section** (*str*) – section name
- **option** (*str*) – option name
- **raw** (*bool*) – set to True to not interpolate
- **use_spec** (*bool*) – set to False to not use the specification

is_valid(*allow_extra_options: bool = False*) → bool
Verify that the `ConfigParser` matches the specification. A `ConfigParser` without a spec is automatically valid.

write(*file: FileType*, *space_around_delimiters: bool = True*) → None
Write config file to a file-like object

Parameters

- **file** (*FileType*) – file-like object to write to
- **space_around_delimiters** (*bool*) – whether to put spaces around the delimiter, i.e., ":" or "="

class epochs.configparser.**EpochConfigParser**(*spec_filename: str = None*, ***kwargs*)
Bases: object

EpochConfigParser parses config files with dates as section name. Retrieving an option for a given date returns the option value on the date closest, but before, the given date.

date

formats

get(*option: str*, *date: DateValue = None*, *raw: bool = False*, ***kwargs*) → OptionValue
Get an option using the type and default from the specification file

Parameters

- **option** (*str*) – option name
- **date** (*FileValue*) – date as a string or datetime.datetime
- **raw** (*bool*) – set to True is disable interpolation

is_valid(*allow_extra_options: bool = False*) → bool
Verify that the *EpochParser* matches the specification. A *configparser* without a spec is automatically valid.

read(*files*)
Attempt to read and parse an iterable of filenames, returning a list of filenames which were successfully parsed.

write(*file: FileType*, *space_around_delimiters: bool = True*) → None
Write config file to a file-like object

Parameters

- **file** (*FileType*) – file-like object to write to
- **space_around_delimiters** (*bool*) – whether to put spaces around the delimiter, i.e., ":" or "="

class epochs.configparser.**OptionSpec**
Bases: tuple

Specification for an option

default

list

required

type

5.1.3 Module contents

Top-level package for epochs.

CHAPTER 6

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:

6.1 Types of Contributions

6.1.1 Report Bugs

Report bugs at https://github.com/mgalloy/epochs/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.

6.1.2 Fix Bugs

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

6.1.3 Implement Features

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

6.1.4 Write Documentation

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

6.1.5 Submit Feedback

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

6.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/epochs.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 epochs
$ cd epochs/
$ 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 epochs tests
$ python setup.py test or 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.

6.3 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 3.5 and 3.6, and for PyPy. Check https://travis-ci.org/mgalloy/epochs/pull_requests and make sure that the tests pass for all supported Python versions.

6.4 Tips

To run a subset of tests:

```
$ py.test tests/test_parser.py
```

6.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

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

Travis will then deploy to PyPI if tests pass.

CHAPTER 7

Credits

7.1 Development Lead

- Michael Galloy <mgalloy@gmail.com>

7.2 Contributors

None yet. Why not be the first?

CHAPTER 8

History

8.1 0.1.0 (2019-04-12)

- initial creation

8.2 0.1.1 (2019-04-19)

- basic config parser with specification file

8.3 0.1.2 (2019-04-19)

- better repr and str representations

8.4 0.2.0 (2019-04-22)

- basic epoch parser

CHAPTER 9

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

Index