marshmallow-configparser Documentation

Release 0.3.3

Tomislav Adamic

Oct 04, 2017

Contents

Ever wanted to load plain `.ini` config files and then validate loaded config?

Ever wanted to load config from multiple locations (`/etc/appconfig.conf`, `~/.appconfig.conf`) into single object and then validate that?

Worry no more!

Python's ConfigParser met marshmallow and now they get along just fine - without any JSON in sight to spoil the fun.

CHAPTER 1

Installation

```
pip install marshmallow_configparser
```

CHAPTER 2

Example

Having config file `/tmp/example_config.conf` looking like this:

```
[Section1]
option1 = mandatory string
option2 = optional string
option3 = 42
option4 = 24

[Section2]
option1 = mandatory string
option2 = optional string
option3 = 42
option4 = 24
```

And wanting to load it into our config object:

```
class ConfigObject(object):
    MANDATORY_STRING1 = None
    OPTIONAL_STRING1 = None
    MANDATORY_INTEGER1 = None
    OPTIONAL_INTEGER1 = None
    MANDATORY_STRING2 = None
    OPTIONAL_STRING2 = None
    MANDATORY_INTEGER2 = None
    OPTIONAL_INTEGER2 = None
```

We can define marshmallow schema:

```
from marshmallow.validate import Range

from marshmallow_configparser import (ConfigBoolean, ConfigInteger,
                                      ConfigParserSchema, ConfigString,
                                      IsNotBlank)

class ConfigSchema(ConfigParserSchema):
```

```
    class Meta:
        model = ConfigObject

    MANDATORY_STRING1 = ConfigString(
        section='Section1', load_from='option1', dump_to='option1',
        validate=[IsNotBlank()]
    )
    OPTIONAL_STRING1 = ConfigString(
        section='Section1', load_from='option2', dump_to='option2',
    )
    MANDATORY_INTEGER1 = ConfigInteger(
        section='Section1', load_from='option3', dump_to='option3',
        validate=[Range(min=24, max=42)]
    )
    OPTIONAL_INTEGER1 = ConfigInteger(
        section='Section1', load_from='option4', dump_to='option4',
    )

    MANDATORY_STRING2 = ConfigString(
        section='Section2', load_from='option1', dump_to='option1',
        validate=[IsNotBlank()]
    )
    OPTIONAL_STRING2 = ConfigString(
        section='Section2', load_from='option2', dump_to='option2',
    )
    MANDATORY_INTEGER2 = ConfigInteger(
        section='Section2', load_from='option3', dump_to='option3',
        validate=[Range(min=24, max=42)]
    )
    OPTIONAL_INTEGER2 = ConfigInteger(
        section='Section2', load_from='option4', dump_to='option4',
    )
```

Which can then load and validate our config:

```
schema = ConfigSchema()
obj, errors = schema.load(['/tmp/example_config.conf'])
```

In the end we have:

```
obj.__dict_

{'MANDATORY_INTEGER1': 42,
 'MANDATORY_INTEGER2': 42,
 'MANDATORY_STRING1': 'mandatory string',
 'MANDATORY_STRING2': 'mandatory string',
 'OPTIONAL_INTEGER1': 24,
 'OPTIONAL_INTEGER2': 24,
 'OPTIONAL_STRING1': 'optional string',
 'OPTIONAL_STRING2': 'optional string'}
```

Instead of using convenience classes like `ConfigString`, there are also classes in `marshmallow_configparser.fields` module that expose full marshmallow API. Check the docs for details.

CHAPTER 3

Documentation

http://marshmallow-configparser.readthedocs.io/en/latest/index.html

Tables of contents and indices

API reference

marshmallow_configparser

class **Boolean** (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Boolean

class **ConfigParserFieldMixin**
Bases: object

dump_to

load_from

class **Date** (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Date

class **DateTime** (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.DateTime

class **Decimal** (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Decimal

class **Dict** (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Dict

class Email (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Email

class Float (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Float

class FormattedString (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.FormattedString

class Function (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Function

class Integer (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Integer

class List (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.List

class LocalDateTime (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.LocalDateTime

class Method (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Method

class Number (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Number

class String (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.String

class Time (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Time

class TimeDelta (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.TimeDelta

class UUID (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.UUID

class Url (*section*, **args*, ***kwargs*)
Bases: *marshmallow_configparser.fields.ConfigParserFieldMixin*, marshmallow.fields.Url

class ConfigParserSchema (*extra=None*, *only=()*, *exclude=()*, *prefix=u''*, *strict=None*, *many=False*, *context=None*, *load_only=()*, *dump_only=()*, *partial=False*)
Bases: marshmallow.schema.Schema

OPTIONS_CLASS
alias of *ModelOpts*

dump(*obj*)
Dump object to list of strings representing INI file.

dumps(*obj*)
Dump object to string representing INI file.

load(*config_files*)
Load configuration from list of config file paths.

loads(*ini_file_data*)
Load configuration from string representing INI file.

make_config_object(*data*)

opts = <marshmallow_configparser.schema.ModelOpts object>

class **ModelOpts**(*meta*, ***kwargs*)
Bases: marshmallow.schema.SchemaOpts

is_blank(*line*)

class **IsNotBlank**
Bases: marshmallow.validate.Validator

Validator which succeeds if the value passed is not blank string.

error = None

class **IsNotNone**
Bases: marshmallow.validate.Validator

Validator which succeeds if the value passed is not blank string.

error = None

class **ConfigBoolean**(*section*, *default=None*, *attribute=None*, *load_from=None*, *dump_to=None*, *error=None*, *error_messages=None*, ***metadata*)
Bases: *marshmallow_configparser.fields.Boolean*

marshmallow.fields.Field that ensures:

- option is always present in deserialized data

This is pretty similar to *Boolean*, except this one assumes defaults for some of *Integer* attributes.

class **ConfigInteger**(*section*, *default=None*, *attribute=None*, *load_from=None*, *dump_to=None*, *error=None*, *error_messages=None*, *as_string=False*, *validate=None*, ***metadata*)
Bases: *marshmallow_configparser.fields.Integer*

marshmallow.fields.Field that ensures:

- option is always present in deserialized data
- deserialized value is either the one read from config file or one declared in `default`
- allows None as deserialized value (*Integer* raises ValidationError('Not a valid integer.'))

This is pretty similar to *Integer*, except this one assumes defaults for some of *Integer* attributes and changes treating of None values.

class **ConfigString**(*section*, *default=''*, *attribute=None*, *load_from=None*, *dump_to=None*, *error=None*, *load_only=False*, *dump_only=False*, *error_messages=None*, *validate=None*, ***metadata*)
Bases: *marshmallow_configparser.fields.String*

marshmallow.fields.Field that ensures:

- option is always present in deserialized data

- deserialized value is either the one read from config file or one declared in `default`

This is pretty similar to *`String`*, except this one assumes defaults for some of *`String`* attributes.

License

marshmallow_configparser is licensed under terms of MIT License

```
Copyright (c) 2017 Tomislav Adamic

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Development

Preparing development environment

Create new virtual environment

```
cd path/to/cloned/repo/marshmallow_configparser
python3 -m venv .venv
source .venv/bin/activate
pip install -u pip wheel
```

Installing in develop mode

```
python setup.py develop
```

Later, to uninstall it

```
python setup.py develop --uninstall
```

To install extra packages useful in development

```
pip install -e .[dev]
```

Running tests

```
py.test
```

or to get more verbose output

```
py.test -p no:sugar --spec
```

or to generate tests coverage

```
py.test --cov=marshmallow_configparser --cov-report=html
```

and finally, tests can be run with tox

```
tox
```

Note, to combine the coverage data from all the tox environments run:

Windows	`set PYTEST_ADDOPTS=--cov-append` `tox`
Other	`PYTEST_ADDOPTS=--cov-append tox`

Running under PyPy3

```
wget https://bitbucket.org/pypy/pypy/downloads/pypy3-v5.8.0-linux64.tar.bz2
tar -xvjf pypy3-v5.8.0-linux64.tar.bz2
virtualenv -p pypy3-v5.8.0-linux64/bin/pypy3 .venvpypy
source .venvpypy/bin/activate
pip install -U pip wheel
```

Profiling

Use IPython shell to generate profiling data

```
%prun -D program.prof [mover.move(d) for d in moves_cycle]
```

After that, it is viewable by either Snakeviz

```
snakeviz program.prof
```

or as call graph through KCacheGrind

```
pyprof2calltree -i program.prof
kcachegrind program.prof.log
```

Uploading to PyPI

```
pip install -U twine
```

Prepare ~/.pypirc

```
[distutils]
index-servers=
    pypi
    pypitest

[pypitest]
repository = https://test.pypi.org/legacy/
username = <username>
password = <password>

[pypi]
username = <username>
password = <password>
```

Create dist

```
python setup.py sdist bdist_wheel
```

An upload it

```
twine upload -r pypitest dist/*
```

Changelog

0.3.3 (2017-09-20)

- Added attribute to Schema that gives access to underlying config data

0.3.2 (2017-08-07)

- docs cleanup

0.3.1 (2017-08-07)

- repo cleanup

0.3.0 (2017-05-08)

- config validation now fails if there is no config files to read from or they are not readable.

0.2.0 (2017-05-02)

- Added convenience_fields module

0.1.0 (2017-04-30)

- First release on PyPI.
- genindex

- modindex
- search

Python Module Index

m

Index

U