
Twelve Factor

Release 0.2.0-alpha.0

Daniel Knell

Jan 18, 2021

CONTENTS

- 1 Getting Started 3**
 - 1.1 Installation 3
 - 1.2 Quickstart Guide 3
 - 1.3 Schema 4
- 2 Reference 7**
 - 2.1 API 7
- 3 Additional Notes 11**
 - 3.1 Change Log 11
 - 3.2 License 11
- Python Module Index 13**
- Index 15**

twelvefactor is a library that provides utilities for creating a [12 factor](#) application, at present it allows you to parse the processes environment variables into a configuration dictionary.

GETTING STARTED

1.1 Installation

The library can be installed via [Poetry](#) as follows:

```
poetry add twelfefactor
```

Alternatively using [Pipenv](#):

```
pipenv install twelfefactor
```

Or using `pip` directly with:

```
pip install twelfefactor
```

1.2 Quickstart Guide

1.2.1 Flask

```
# wsgi.py
from twelfefactor import config
from flask import Flask

app = Flask(__name__)

app.config.update(config({
    'DEBUG': {
        'type': bool,
        'default': False,
    },
    'SECRET_KEY': str
}))
```

The above will set two config values, the first named `DEBUG` a boolean defaulting to `False`, and the second named `SECRET_KEY` a string which will throw an exception if not set. Both variables will be populated from the processes environment variables.

1.2.2 Django

```
# settings.py
from twelfefactor import config

globals().update(config({
    'DEBUG': {
        'type': bool,
        'default': False,
    },
    'SECRET_KEY': str
}))
```

The above will create two variables, the first named `DEBUG` a boolean defaulting to `False`, and the second named `SECRET_KEY` a string which will throw an exception if not set. Both variables will be populated from the processes environment variables.

1.3 Schema

1.3.1 Example

```
{
    'DEBUG': {
        'type': bool,
        'default': False,
    },
    'SECRET_KEY': str,
    'DATABASE': {
        'key': 'DATABASE_URL',
        'mapper': dj_database_url.parse
    },
    'SOME_SET': {
        'type': set,
        'subtype': int
    }
}
```

1.3.2 Properties

key

The name of the environment variable to look up, this allows you map map values in the environment to differently named configuration variables, it defaults to the name of the configuration variable.

default

A value to use should no environment variable be found, if no default is provided then an error will be thrown.

type

A function to convert the string value to the correct type.

When `list`, `tuple`, or `set` are provided then the value will be interpreted as a comma separated list and interpreted based on the subtype setting.

If no type is set then `str` is assumed.

subtype

A function to convert the string sub-value to the correct type.

If no subtype is set then `str` is assumed.

mapper

A method to post process the value after it has been converted to the correct type, it is the last transformation to be applied and should take the value and transform it into a more suitable configuration value.

A mapper should not be used to instantiate complex classes such as database adapters, these should be instantiated outside of the configuration code.

If no mapper is provided then the value is returned as is.

1.3.3 Shorthand

When only a type is required you can specify just the type instead of a dictionary defining the config value.

The following two examples are identical

```
{
  'DEBUG': bool
}
```

```
{
  'DEBUG': {
    'type': bool
  }
}
```


REFERENCE

2.1 API

class `twelvefactor.Config` (*environ=None*)

Bases: `object`

Config environment parser.

This class allows chosen configuration values to be extracted from the processes environment variables and converted into the relevant types.

```
parser = Config()

config = parser({
    'DEBUG': {
        'type': bool,
        'default': False,
    },
    'SECRET_KEY': str,
})
```

The above will populate the `config` variable with two values, `DEBUG` will be populated with a `bool` from the environment variable of the same name, throwing an exception on invalid values and defaulting to `False` when none is provided, and `SECRET_KEY` will be a `str` and throw a `ConfigError` when no value is found in the environment.

An optional `environ` param can be passed in order to override the environment.

Parameters `environ` (`Optional[Mapping[str, str]]`) – environment dictionary, defaults to `os.environ`

__call__ (*schema*)

Parse the environment according to a schema.

Parameters `schema` (`Mapping[str, Union[Type[Any], SchemaItem]]`) – the schema to parse

Return type `Dict[str, Any]`

Returns a dictionary of config values

get (*key, default=<object object>, type_=<class 'str'>, subtype=<class 'str'>, mapper=None*)

Parse a value from an environment variable.

```
>>> os.environ['FOO']
<<< '12345'
```

(continues on next page)

(continued from previous page)

```
>>>
>>> os.environ['BAR']
<<< '1,2,3,4'
>>>
>>> 'BAZ' in os.environ
<<< False
>>>
>>> parser = Config()
>>> parser.get('FOO', type_=int)
<<< 12345
>>>
>>> parser.get('BAR', type_=list, subtype=int)
<<< [1, 2, 3, 4]
>>>
>>> parser.get('BAZ', default='abc123')
<<< 'abc123'
>>>
>>> parser.get('FOO', type_=int, mapper=lambda x: x*10)
<<< 123450
```

Parameters

- **key** (`str`) – the key to look up the value under
- **default** (`Any`) – default value to return when no value is present
- **type_** (`Type[Any]`) – the type to return
- **subtype** (`Type[Any]`) – subtype for iterator types
- **mapper** (`Optional[Callable[[object], object]]`) – a function to post-process the value with

Return type `Any`

Returns the parsed config value

parse (*value*, *type_*=<class 'str'>, *subtype*=<class 'str'>)

Parse value from string.

Convert value to

```
>>> parser = Config()
>>> parser.parse('12345', type_=int)
<<< 12345
>>>
>>> parser.parse('1,2,3,4', type_=list, subtype=int)
<<< [1, 2, 3, 4]
```

Parameters

- **value** (`str`) – string
- **type_** (`Type[Any]`) – the type to return
- **subtype** (`Type[Any]`) – subtype for iterator types

Return type `Any`

Returns the parsed config value

class `twelvefactor.ConfigError`

Bases: `Exception`

Exception to throw on configuration errors.

`twelvefactor.SchemaItem`

A type annotation for the definition of a single item in a the schema.

`twelvefactor.Schema`

A type annotation for the *Schema*.

`twelvefactor.config`

An instance of *Config*.

ADDITIONAL NOTES

3.1 Change Log

Here you can see the full list of changes.

3.1.1 Unreleased

- Removed Python < 3.6 support
- Added type hints
- Changed build tool to poetry
- Added CI config for python 3.7, 3.8-dev, and nightly
- Improved test suite
- Added code coverage monitoring via coveralls

3.1.2 Version 0.1.2

Released on 2016-08-11

- First public preview release.

3.2 License

twelvefactor is licensed under the MIT license. Basically, you can do whatever you want as long as you include the original copyright and license notice in any copy of the software/source.

3.2.1 MIT License

Copyright (c) 2016 Daniel Knell, <http://danielknell.co.uk>

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.

PYTHON MODULE INDEX

t

twelvefactor, ??

Symbols

`__call__()` (*twelvefactor.Config method*), 7

C

`Config` (*class in twelvefactor*), 7

`config` (*in module twelvefactor*), 9

`ConfigError` (*class in twelvefactor*), 8

G

`get()` (*twelvefactor.Config method*), 7

M

`module`
 twelvefactor, 1, 7

P

`parse()` (*twelvefactor.Config method*), 8

S

`Schema` (*in module twelvefactor*), 9

`SchemaItem` (*in module twelvefactor*), 9

T

`twelvefactor`
 module, 1, 7