Django-environ Documentation

Release 0.4.4

joke2k

Jul 16, 2019

Contents

1 Django-environ **3**
- 1.1 Feature Support . . . 4
- 1.2 Installation . . . 5
- 1.3 Documentation . . . 5
- 1.4 Supported types . . . 5
- 1.5 Tips . . . 7
- 1.6 Nested lists . . . 7
- 1.7 Multiline value . . . 8
- 1.8 Proxy value . . . 8
- 1.9 Multiple env files . . . 8

2 Tests **9**
- 2.1 How to Contribute . . . 9
- 2.2 License . . . 9
- 2.3 Changelog . . . 9
- 2.4 Credits . . . 10

3 environ.Env **11**

4 environ.Path **15**

5 Indices and tables **17**

Python Module Index **19**

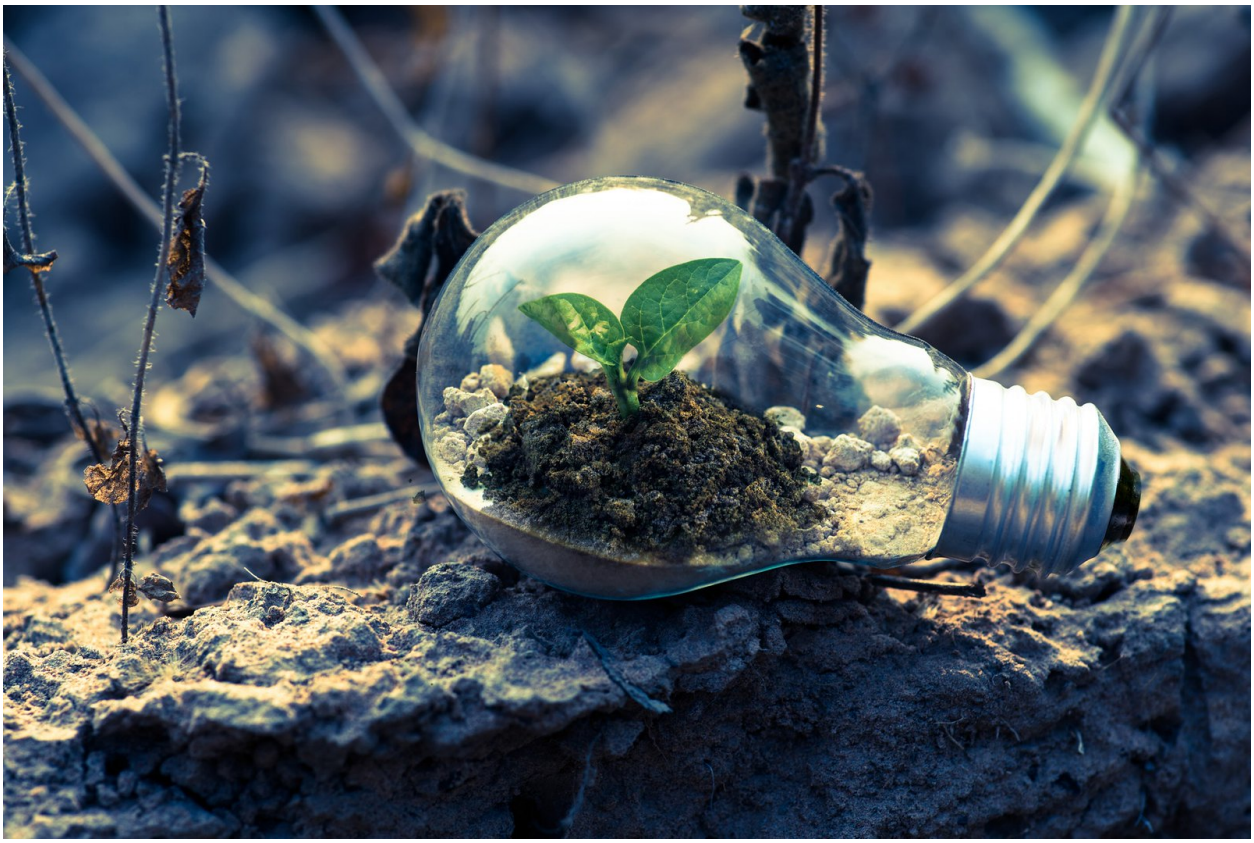
Index **21**

Django-environ allows you to utilize 12factor inspired environment variables to configure your Django application.

CHAPTER 1

Django-environ

django-environ allows you to use Twelve-factor methodology to configure your Django application with environment variables.



```
import environ
env = environ.Env(
    # set casting, default value
    DEBUG=(bool, False)
)
# reading .env file
environ.Env.read_env()

# False if not in os.environ
DEBUG = env('DEBUG')

# Raises django's ImproperlyConfigured exception if SECRET_KEY not in os.environ
SECRET_KEY = env('SECRET_KEY')

# Parse database connection url strings like psql://user:pass@127.0.0.1:8458/db
DATABASES = {
    # read os.environ['DATABASE_URL'] and raises ImproperlyConfigured exception if␣
↪not found
    'default': env.db(),
    # read os.environ['SQLITE_URL']
    'extra': env.db('SQLITE_URL', default='sqlite:////tmp/my-tmp-sqlite.db')
}

CACHES = {
    # read os.environ['CACHE_URL'] and raises ImproperlyConfigured exception if not␣
↪found
    'default': env.cache(),
    # read os.environ['REDIS_URL']
    'redis': env.cache('REDIS_URL')
}
```

See the similar code, sans django-environ.

```
     _ _                                                    _
    | (_)                                                  (_)
  __| |_  __ _ _ __    __ _  ___ _______ ___ _ _____   ___ _ __ ___  _ __
 / _` | |/ _` | '_ \ / _` |/ _ \_______/ _ \ '_ \ V /| | '__/ _ \| '_ \
| (_| | | (_| | | | | (_| | (_) |     |  __/ | | \ V /| | | | (_) | | | | |
 \__,_| |\__,_|_| |_|\__, |\___/       \___|_| |_|\_/ |_|_|  \___/|_| |_|
     _/ |             __/ |
    |__/             |___/
```

The idea of this package is to unify a lot of packages that make the same stuff: Take a string from `os.environ`, parse and cast it to some of useful python typed variables. To do that and to use the 12factor approach, some connection strings are expressed as url, so this package can parse it and return a `urllib.parse.ParseResult`. These strings from `os.environ` are loaded from a *.env* file and filled in `os.environ` with `setdefault` method, to avoid to overwrite the real environ. A similar approach is used in Two Scoops of Django book and explained in 12factor-django article.

Using django-environ you can stop to make a lot of unversioned `settings_*.py` to configure your app. See cookiecutter-django for a concrete example on using with a django project.

1.1 Feature Support

- Fast and easy multi environment for deploy

- Fill `os.environ` with .env file variables
- Variables casting (see *Supported types* below)
- Url variables exploded to django specific package settings

Django-environ officially supports Django 1.8 ~ 2.0.

1.2 Installation

```
$ pip install django-environ
```

NOTE: No need to add it to INSTALLED_APPS.

Then create a `.env` file:

```
DEBUG=on
SECRET_KEY=your-secret-key
DATABASE_URL=psql://urser:un-githubbedpassword@127.0.0.1:8458/database
SQLITE_URL=sqlite:///my-local-sqlite.db
CACHE_URL=memcache://127.0.0.1:11211,127.0.0.1:11212,127.0.0.1:11213
REDIS_URL=rediscache://127.0.0.1:6379/1?client_class=django_redis.client.
↪DefaultClient&password=ungithubbed-secret
```

And use it with *settings.py* above. Don't forget to add `.env` in your `.gitignore` (tip: add `.env.example` with a template of your variables).

1.3 Documentation

Documentation is available at RTFD.

1.4 Supported types

- str
- bool
- int
- float
- json
- list (FOO=a,b,c)
- tuple (FOO=(a,b,c))
- dict (BAR=key=val,foo=bar) #environ.Env(BAR=(dict, {}))
- dict (BAR=key=val;foo=1.1;baz=True) #environ.Env(BAR=(dict(value=unicode, cast=dict(foo=float,baz=bool)), {}))
- url
- path (environ.Path)
- **db_url**

  - PostgreSQL: postgres://, pgsql://, psql:// or postgresql://
  - PostGIS: postgis://
  - MySQL: mysql:// or mysql2://
  - MySQL for GeoDjango: mysqlgis://
  - SQLITE: sqlite://
  - SQLITE with SPATIALITE for GeoDjango: spatialite://
  - Oracle: oracle://
  - MSSQL: mssql://
  - PyODBC: pyodbc://
  - Redshift: redshift://
  - LDAP: ldap://
- **cache_url**
  - Database: dbcache://
  - Dummy: dummycache://
  - File: filecache://
  - Memory: locmemcache://
  - Memcached: memcache://
  - Python memory: pymemcache://
  - Redis: rediscache://
- **search_url**
  - ElasticSearch: elasticsearch://
  - Solr: solr://
  - Whoosh: whoosh://
  - Xapian: xapian://
  - Simple cache: simple://
- **email_url**
  - SMTP: smtp://
  - SMTP+SSL: smtp+ssl://
  - SMTP+TLS: smtp+tls://
  - Console mail: consolemail://
  - File mail: filemail://
  - LocMem mail: memorymail://
  - Dummy mail: dummymail://

1.5 Tips

1.5.1 Using unsafe characters in URLs

In order to use unsafe characters you have to encode with `urllib.parse.encode` before you set into `.env` file.

```
DATABASE_URL=mysql://user:%23password@127.0.0.1:3306/dbname
```

See https://perishablepress.com/stop-using-unsafe-characters-in-urls/ for reference.

1.5.2 Multiple redis cache locations

For redis cache, multiple master/slave or shard locations can be configured as follows:

```
CACHE_URL='rediscache://master:6379,slave1:6379,slave2:6379/1'
```

1.5.3 Email settings

In order to set email configuration for django you can use this code:

```
EMAIL_CONFIG = env.email_url(
    'EMAIL_URL', default='smtp://user@:password@localhost:25')

vars().update(EMAIL_CONFIG)
```

1.5.4 SQLite urls

SQLite connects to file based databases. The same URL format is used, omitting the hostname, and using the "file" portion as the filename of the database. This has the effect of four slashes being present for an absolute

file path: `sqlite:////full/path/to/your/database/file.sqlite`.

1.6 Nested lists

Some settings such as Django's `ADMINS` make use of nested lists. You can use something like this to handle similar cases.

```
# DJANGO_ADMINS=John:john@admin.com,Jane:jane@admin.com
ADMINS = [x.split(':') for x in env.list('DJANGO_ADMINS')]

# or use more specific function

from email.utils import getaddresses

# DJANGO_ADMINS=Full Name <email-with-name@example.com>,
↪anotheremailwithoutname@example.com
ADMINS = getaddresses([env('DJANGO_ADMINS')])
```

1.7 Multiline value

You can set a multiline variable value:

```
# MULTILINE_TEXT=Hello\\nWorld
>>> print env.str('MULTILINE_TEXT', multiline=True)
Hello
World
```

1.8 Proxy value

You can set a value prefixed by `$` to use as a proxy to another variable value:

```
# BAR=FOO
# PROXY=$BAR
>>> print env.str('PROXY')
FOO
```

1.9 Multiple env files

It is possible to have multiple env files and select one using environment variables.

Now `ENV_PATH=other-env ./manage.py runserver` uses `other-env` while `./manage.py runserver` uses `.env`.

CHAPTER 2

Tests

```
$ git clone git@github.com:joke2k/django-environ.git
$ cd django-environ/
$ python setup.py test
```

2.1 How to Contribute

1. Check for open issues or open a fresh issue to start a discussion around a feature idea or a bug. There is a Contributor Friendly tag for issues that should be ideal for people who are not very familiar with the codebase yet.
2. Fork the repository on GitHub to start making your changes to the **develop** branch (or branch off of it).
3. Write a test which shows that the bug was fixed or that the feature works as expected.
4. Send a pull request and bug the maintainer until it gets merged and published. :) Make sure to add yourself to Authors file.

2.2 License

This project is licensed under the MIT License - see the License file file for details

2.3 Changelog

See the Changelog file which format is *inspired* by Keep a Changelog.

2.4 Credits

- See Authors file
- 12factor
- 12factor-django
- Two Scoops of Django
- rconradharris / envparse
- kennethreitz / dj-database-url
- migonzalvar / dj-email-url
- ghickman / django-cache-url
- dstufft / dj-search-url
- julianwachholz / dj-config-url
- nickstenning / honcho
- rconradharris / envparse
- Distribute
- modern-package-template

CHAPTER 3

environ.Env

class environ.environ.**Env**(***scheme*)

Provide scheme-based lookups of environment variables so that each caller doesn't have to pass in *cast* and *default* parameters.

Usage::

```
env = Env(MAIL_ENABLED=bool, SMTP_LOGIN=(str, 'DEFAULT'))
if env('MAIL_ENABLED'):
    ...
```

BOOLEAN_TRUE_STRINGS = ('true', 'on', 'ok', 'y', 'yes', '1')

DB_SCHEMES = {'ldap': 'ldapdb.backends.ldap', 'mssql': 'sql_server.pyodbc', 'mysql':

DEFAULT_DATABASE_ENV = 'DATABASE_URL'

CACHE_SCHEMES = {'dbcache': 'django.core.cache.backends.db.DatabaseCache', 'dummycach

DEFAULT_CACHE_ENV = 'CACHE_URL'

EMAIL_SCHEMES = {'consolemail': 'django.core.mail.backends.console.EmailBackend', 'du

DEFAULT_EMAIL_ENV = 'EMAIL_URL'

SEARCH_SCHEMES = {'elasticsearch': 'haystack.backends.elasticsearch_backend.Elasticse

DEFAULT_SEARCH_ENV = 'SEARCH_URL'

__call__(*...*) <==> *x*(*...*)

str(*var*, *default=<NoValue>*, *multiline=False*)

Return type str

bool(*var*, *default=<NoValue>*)

Return type bool

int(*var*, *default=<NoValue>*)

Return type int

float(*var*, *default=<NoValue>*)

Return type float

json(*var*, *default=<NoValue>*)

Returns Json parsed

list(*var*, *cast=None*, *default=<NoValue>*)

Return type list

dict(*var*, *cast=<type 'dict'>*, *default=<NoValue>*)

Return type dict

url(*var*, *default=<NoValue>*)

Return type urlparse.ParseResult

db_url(*var='DATABASE_URL'*, *default=<NoValue>*, *engine=None*)
Returns a config dictionary, defaulting to DATABASE_URL.

Return type dict

cache_url(*var='CACHE_URL'*, *default=<NoValue>*, *backend=None*)
Returns a config dictionary, defaulting to CACHE_URL.

Return type dict

email_url(*var='EMAIL_URL'*, *default=<NoValue>*, *backend=None*)
Returns a config dictionary, defaulting to EMAIL_URL.

Return type dict

search_url(*var='SEARCH_URL'*, *default=<NoValue>*, *engine=None*)
Returns a config dictionary, defaulting to SEARCH_URL.

Return type dict

path(*var*, *default=<NoValue>*, ***kwargs*)

Return type *Path*

classmethod read_env(*env_file=None*, ***overrides*)
Read a .env file into os.environ.

If not given a path to a dotenv path, does filthy magic stack backtracking to find manage.py and then find the dotenv.

http://www.wellfireinteractive.com/blog/easier-12-factor-django/

https://gist.github.com/bennylope/2999704

classmethod db_url_config(*url*, *engine=None*)
Pulled from DJ-Database-URL, parse an arbitrary Database URL. Support currently exists for PostgreSQL, PostGIS, MySQL, Oracle and SQLite.

SQLite connects to file based databases. The same URL format is used, omitting the hostname, and using the "file" portion as the filename of the database. This has the effect of four slashes being present for an absolute file path:

```
>>> from environ import Env
>>> Env.db_url_config('sqlite:////full/path/to/your/file.sqlite')
{'ENGINE': 'django.db.backends.sqlite3', 'HOST': '', 'NAME': '/full/path/to/
↪your/file.sqlite', 'PASSWORD': '', 'PORT': '', 'USER': ''}
```

(continues on next page)

(continued from previous page)

```
>>> Env.db_url_config('postgres://uf07k1i6d8ia0v:wegauwhgeuioweg@ec2-107-21-
↪253-135.compute-1.amazonaws.com:5431/d8r82722r2kuvn')
{'ENGINE': 'django.db.backends.postgresql', 'HOST': 'ec2-107-21-253-135.
↪compute-1.amazonaws.com', 'NAME': 'd8r82722r2kuvn', 'PASSWORD':
↪'wegauwhgeuioweg', 'PORT': 5431, 'USER': 'uf07k1i6d8ia0v'}
```

classmethod cache_url_config(*url*, *backend=None*)
Pulled from DJ-Cache-URL, parse an arbitrary Cache URL.

Parameters

- **url** –
- **backend** –

Returns

classmethod email_url_config(*url*, *backend=None*)
Parses an email URL.

classmethod search_url_config(*url*, *engine=None*)

get_value(*var*, *cast=None*, *default=<NoValue>*, *parse_default=False*)
Return value for given environment variable.

Parameters

- **var** – Name of variable.
- **cast** – Type to cast return value as.
- **default** – If var not present in environ, return this instead.
- **parse_default** – force to parse default..

Returns Value from environment or default (if set)

classmethod parse_value(*value*, *cast*)
Parse and cast provided value

Parameters

- **value** – Stringed value.
- **cast** – Type to cast return value as.

Returns Casted value

CHAPTER 4

environ.Path

class environ.environ.**Path**(*start=''*, **paths*, ***kwargs*)

Inspired to Django Two-scoops, handling File Paths in Settings.

```
>>> from environ import Path
>>> root = Path('/home')
>>> root, root(), root('dev')
(<Path:/home>, '/home', '/home/dev')
>>> root == Path('/home')
True
>>> root in Path('/'), root not in Path('/other/path')
(True, True)
>>> root('dev', 'not_existing_dir', required=True)
Traceback (most recent call last):
environ.environ.ImproperlyConfigured: Create required path: /home/not_existing_dir
>>> public = root.path('public')
>>> public, public.root, public('styles')
(<Path:/home/public>, '/home/public', '/home/public/styles')
>>> assets, scripts = public.path('assets'), public.path('assets', 'scripts')
>>> assets.root, scripts.root
('/home/public/assets', '/home/public/assets/scripts')
>>> assets + 'styles', str(assets + 'styles'), ~assets
(<Path:/home/public/assets/styles>, '/home/public/assets/styles', <Path:/home/
↪public>)
```

root -> Retrieve absolute path

__call__(**paths*, ***kwargs*)

Retrieve the absolute path, with appended paths

Parameters

- **paths** – List of sub path of self.root
- **kwargs** – required=False

path(**paths*, ***kwargs*)

Create new Path based on self.root and provided paths.

Parameters

- **paths** – List of sub paths
- **kwargs** – required=False

Return type *Path*

file(*name*, **args*, ***kwargs*)
Open a file.

Parameters

- **name** – Filename appended to self.root
- **args** – passed to open()
- **kwargs** – passed to open()

Return type file

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

environ.environ, **??**

Index

Symbols

__call__() (*environ.environ.Env method*), 11
__call__() (*environ.environ.Path method*), 15

B

bool() (*environ.environ.Env method*), 11
BOOLEAN_TRUE_STRINGS (*environ.environ.Env attribute*), 11

C

CACHE_SCHEMES (*environ.environ.Env attribute*), 11
cache_url() (*environ.environ.Env method*), 12
cache_url_config() (*environ.environ.Env class method*), 13

D

DB_SCHEMES (*environ.environ.Env attribute*), 11
db_url() (*environ.environ.Env method*), 12
db_url_config() (*environ.environ.Env class method*), 12
DEFAULT_CACHE_ENV (*environ.environ.Env attribute*), 11
DEFAULT_DATABASE_ENV (*environ.environ.Env attribute*), 11
DEFAULT_EMAIL_ENV (*environ.environ.Env attribute*), 11
DEFAULT_SEARCH_ENV (*environ.environ.Env attribute*), 11
dict() (*environ.environ.Env method*), 12

E

EMAIL_SCHEMES (*environ.environ.Env attribute*), 11
email_url() (*environ.environ.Env method*), 12
email_url_config() (*environ.environ.Env class method*), 13
Env (*class in environ.environ*), 11
environ.environ (*module*), 1

F

file() (*environ.environ.Path method*), 16
float() (*environ.environ.Env method*), 12

G

get_value() (*environ.environ.Env method*), 13

I

int() (*environ.environ.Env method*), 11

J

json() (*environ.environ.Env method*), 12

L

list() (*environ.environ.Env method*), 12

P

parse_value() (*environ.environ.Env class method*), 13
Path (*class in environ.environ*), 15
path() (*environ.environ.Env method*), 12
path() (*environ.environ.Path method*), 15

R

read_env() (*environ.environ.Env class method*), 12

S

SEARCH_SCHEMES (*environ.environ.Env attribute*), 11
search_url() (*environ.environ.Env method*), 12
search_url_config() (*environ.environ.Env class method*), 13
str() (*environ.environ.Env method*), 11

U

url() (*environ.environ.Env method*), 12