
django-djconfig Documentation

Release 0.5.0

Esteban Castro Borsani

Jan 06, 2019

Contents

1	User's Guide	1
1.1	Installation	1
1.2	Usage	2
1.3	Cookbook	5
1.4	Fields	6
2	API Reference	9
2.1	API	9
3	Additional Notes	13
3.1	Changelog	13
3.2	License	15
	Python Module Index	17

1.1 Installation

1.1.1 Requirements

- Python 2.7, 3.4, 3.5, 3.6 or 3.7
- Django 1.11 LTS, 2.0 or 2.1

1.1.2 Pip

```
pip install django-djconfig
```

1.1.3 Configuration

```
# settings.py

INSTALLED_APPS = [
    # ...

    'djconfig',
]

MIDDLEWARE_CLASSES = [
    # ...

    'djconfig.middleware.DjConfigMiddleware',
]

TEMPLATES = [
```

(continues on next page)

(continued from previous page)

```
{
    # ...
    'OPTIONS': {
        'context_processors': [
            # ...
            'djconfig.context_processors.config',
        ],
    },
},
]
```

Note: Use **MIDDLEWARE** instead of **MIDDLEWARE_CLASSES** in Django >= 1.10

Afterwards, run:

```
python manage.py migrate
```

All done.

1.2 Usage

1.2.1 Creating the config form

Tip: Form's field names must be unique across forms, so you should prefix them with the name of your app.

```
# forms.py

from djconfig.forms import ConfigForm

class AppConfigForm(ConfigForm):

    myapp_first_key = forms.BooleanField(initial=True, required=False)
    myapp_second_key = forms.IntegerField(initial=20)
```

1.2.2 Registering the config form

Tip: Read the [djconfig_applications_docs](#)

```
# apps.py

from django.apps import AppConfig

class MyAppConfig(AppConfig):
```

(continues on next page)

(continued from previous page)

```
name = 'myapp'
verbose_name = "Myapp"

def ready(self):
    self.register_config()
    # ...

def register_config(self):
    import djconfig
    from .forms import MyConfigForm

    djconfig.register(MyConfigForm)
```

1.2.3 Accessing the config

```
from djconfig import config

if config.myapp_first_key:
    # ...
```

Accessing the config within templates:

```
# template.html

# ...

{% if config.myapp_first_key %}
    # ...
{% endif %}
```

1.2.4 Editing the config values

```
# views.py

@login_required
def config_view(request):
    if not request.user.is_superuser:
        raise Http404

    if request.method == 'POST':
        form = AppConfigForm(data=request.POST)

        if form.is_valid():
            form.save()
            return redirect('/')
    else:
        form = AppConfigForm()

    return render(request, 'app/configuration.html', {'form': form, })
```

1.2.5 Testing helpers

There is a helper similar to django's `@override_settings` that can be used in tests.

```
# tests.py

from djconfig.utils import override_djconfig

@override_djconfig(myapp_first_key="foo", myapp_second_key="bar")
def test_something(self):
    # ...
```

Calling `djconfig.reload_maybe()` is required when unit testing. For example, it may be called within the test's `setUp` method to run it before each test. The middleware will call this, so it's not needed on integration tests that make use of django's `test Client`.

```
# tests.py

import djconfig

def setUp(self):
    djconfig.reload_maybe()
```

1.2.6 Admin

Register a config form into django admin.

The following example shows how to register a single form that contains all settings:

```
# admin.py

import djconfig
from .forms import AppConfigForm

class AppConfigAdmin(djconfig.admin.ConfigAdmin):
    change_list_form = AppConfigForm

class AppConfig(djconfig.admin.Config):
    app_label = 'djconfig'
    verbose_name_plural = 'app config'
    slug = 'appconfig'

djconfig.admin.register(AppConfig, AppConfigAdmin)
```

The following example shows how to register a form in multiple apps:

```
# myapp/admin.py

import djconfig
from .forms import MyAppConfigForm

class MyAppConfigAdmin(djconfig.admin.ConfigAdmin):
    change_list_form = MyAppConfigForm
```

(continues on next page)

(continued from previous page)

```

class MyAppConfig(djconfig.admin.Config):
    app_label = 'myapp'
    verbose_name_plural = 'myapp config'
    slug = 'myappconfig'

djconfig.admin.register(MyAppConfig, MyAppConfigAdmin)

# myotherapp/admin.py

import djconfig
from .forms import MyOtherAppConfigForm

class MyOtherAppConfigAdmin(djconfig.admin.ConfigAdmin):
    change_list_form = MyOtherAppConfigForm

class MyOtherAppConfig(djconfig.admin.Config):
    app_label = 'myotherapp'
    verbose_name_plural = 'myotherapp config'
    slug = 'myotherappconfig'

djconfig.admin.register(MyOtherAppConfig, MyOtherAppConfigForm)

```

1.3 Cookbook

1.3.1 Save an image

```

from django import forms
from django.core.files.storage import default_storage
from djconfig.forms import ConfigForm

class MyImageForm(ConfigForm):
    """
    Save an image

    Usage ::

        # on POST, files must be passed
        form = MyImageForm(data=request.POST, files=request.FILES)
        if form.is_valid():
            form.save()
            return redirect('/')

    """

    myapp_image = forms.ImageField(initial=None, required=False)

    def save_image(self):

```

(continues on next page)

(continued from previous page)

```
image = self.cleaned_data.get('myapp_image')
if image:
    # `name` may change if the storage renames the file,
    # so we update it `image.name = ...`
    image.name = default_storage.save(image.name, image)

def save(self):
    self.save_image()
    # the image name will be saved into `conf.myapp_image`
    super(MyImageForm, self).save()
```

1.4 Fields

1.4.1 Supported form fields

The following form fields were tested:

- BooleanField
- CharField
- EmailField
- FloatField
- IntegerField
- URLField
- ChoiceField
- ModelChoiceField
- ModelMultipleChoiceField
- FileField
- ImageField

DateField is not supported at this time (sorry).

1.4.2 Limitations

ChoiceField

The config will always return a *string* representation of the saved value. It's up to you to coerce it to the right type (int, float or boolean), which can be done within the `clean_my_field` method.

Example:

```
# forms.py

from djconfig.forms import ConfigForm

class AppConfigForm(ConfigForm):
```

(continues on next page)

(continued from previous page)

```

    myapp_choice = forms.ChoiceField(initial=None, choices=[(1, 'label_a'), (2,
↪ 'label_b')])

def clean_myapp_choice(self):
    # By doing this, config.myapp_choice
    # will return a int instead of a string
    return int(self.cleaned_data['myapp_choice'])

```

ModelChoiceField

The config will always return the model instance which is frozen in time to when the config was loaded. If you just need the *pk*, consider returning it within the `clean_my_field` method.

The config will return the initial value (usually `None`), if the previously saved choice is ever deleted from the data base.

`to_field_name` parameter is *not* currently supported.

Example:

```

# forms.py

from djconfig.forms import ConfigForm

class AppConfigForm(ConfigForm):

    myapp_model_choice = forms.ModelChoiceField(initial=None, queryset=MyModel.
↪ objects.all())

def clean_myapp_model_choice(self):
    # By doing this, config.myapp_model_choice
    # will return the model instance pk
    # instead of the model instance object
    return self.cleaned_data['myapp_model_choice'].pk

```


2.1 API

2.1.1 djconfig module

`djconfig.config` `djconfig.conf.Config` object (singleton)

Contain registry of config forms and cache of key-value matching the forms field-value.

All methods are private to avoid clashing with the dynamic attributes.

This should be usually accessed through `config`

`djconfig.register` `djconfig.conf.Config._register` attribute

Register a config form into the registry

Parameters

- **form_class** (*object*) – The form class to register. Must be an instance of `djconfig.forms.ConfigForm`
- **check_middleware** (*bool*) – Check `djconfig.middleware.DjConfigMiddleware` is registered into `settings.MIDDLEWARE_CLASSES`. Default True

`djconfig.reload_maybe` `djconfig.conf.Config._reload_maybe` attribute

Reload the config if the config model has been updated. This is called once on every request by the middleware. Should not be called directly.

2.1.2 Config Object

class `djconfig.conf.Config`

Contain registry of config forms and cache of key-value matching the forms field-value.

All methods are private to avoid clashing with the dynamic attributes.

This should be usually accessed through `config`

2.1.3 ConfigForm Object

class `django.config.forms.ConfigForm(*args, **kwargs)`

Base class for every registered config form. It behaves like a regular form.

Inherits from `django.forms.Form`. The `initial` attr will be updated with the config values if any.

All form fields implementing this, should have a unique name to avoid clashing with other registered forms, prefixing them with the app name is a good practice.

Parameters

- **args** – Positional parameters passed to parent class
- **kwargs** – Keyword parameters passed to parent class

save()

Save the config with the cleaned data, update the last modified date so the config is reloaded on other process/nodes. Reload the config so it can be called right away.

2.1.4 Template Context Processors

`django.config.context_processors.config(request)`

Simple context processor that puts the config into every RequestContext. Just make sure you have a setting like this:

```
TEMPLATE_CONTEXT_PROCESSORS = (  
    # ...  
    'django.config.context_processors.config',  
)
```

2.1.5 Middlewares

class `django.config.middleware.DjConfigMiddleware(get_response=None)`

Populate the cache using the database. Reload the cache *only* if it is not up to date with the config model

2.1.6 Test Helpers

`django.config.utils.override_djconfig(**new_cache_values)`

Temporarily override config values.

This is similar to `django.test.override_settings()`, use it in testing.

Parameters **new_cache_values** – Keyword arguments, the key should match one in the config, a new one is created otherwise, the value is overridden within the decorated function

2.1.7 Admin

This module allows to register a config into django's admin.

Usage:

```

class FooConfigAdmin(djconfig.admin.ConfigAdmin):
    change_list_form = FooConfigForm

class FooConfig(djconfig.admin.Config):
    app_label = 'djconfig'
    verbose_name_plural = 'foo config'
    slug = 'fooconfig'

djconfig.admin.register(FooConfig, FooConfigAdmin)

```

`django-djconfig.admin.register` (*conf*, *conf_admin*, ***options*)

Register a new admin section.

Parameters

- **conf** – A subclass of `django-djconfig.admin.Config`
- **conf_admin** – A subclass of `django-djconfig.admin.ConfigAdmin`
- **options** – Extra options passed to `django.contrib.admin.site.register`

class `django-djconfig.admin.ConfigAdmin` (*model*, *admin_site*)

A `ConfigAdmin` is subclass of `django.contrib.admin.ModelAdmin`.

`change_list_form` class var must be set to a valid `django-djconfig.forms.ConfigForm` subclass

class `django-djconfig.admin.Config`

A `Config` is akin to `django`'s model `Meta` class.

`app_label` must be a valid installed app, `'djconfig'` may be used for every registered form, if they don't belong in a particular app. `verbose_name_plural` is the title of the admin's section link, it can be anything. The (`app_label`, `verbose_name_plural`, `name`) must be unique together across registered forms. `name` is used as the link slug, and it might be used in other places, valid chars are `[a-zA-Z_]`

3.1 Changelog

3.1.1 0.9.0

- Add django admin panel support

3.1.2 0.8.0

- Drop support for DJango 1.8, 1.9 and 1.10
- Add support for Django 2.1 (no changes were required)
- Adds Python 3.7 support (no changes were required)

3.1.3 0.7.0

- Add: support for *ModelMultipleChoiceField*

3.1.4 0.6.0

- Add: support for *ImageField* and *FileField* PR #27
- Adds Django 1.11 support (no changes were required)
- Adds Django 2.0 support (no changes were required)
- Adds Python 3.6 support (no changes were required)

3.1.5 0.5.3

- Fix: compat for new style middleware (PR #25)

3.1.6 0.5.2

- Adds compat for new style (Django 1.10) middleware (PR #24)

3.1.7 0.5.1

- Adds Django 1.10 support

3.1.8 0.5.0

- Drops Django 1.7 support
- Adds Django 1.9 support
- Adds Python 3.5 support
- Remove config lazy loading
- Adds *conf.reload_maybe()* to load the config
- Adds *app.py* config
- Docs

3.1.9 0.4.0

- No longer use django cache
- Renamed *DjConfigLocMemMiddleware* to *DjConfigMiddleware*
- *DjConfigMiddleware* is required

3.1.10 0.3.2

- Fix to never expire keys

3.1.11 0.3.1

- Include missing migrations in setup.py

3.1.12 0.3.0

- Drops support for django 1.5 and 1.6 (for no special reason)
- Support for django 1.8
- Adds migrations
- Raise *AttributeError* if the config key/attr is not found

- Fix race condition that caused returning non existent values (None) if the config was not fully loaded
- Huge code refactor

3.1.13 0.2.0

- Configuration is lazy loaded, now. This means the database will get queried the first time an option is accessed (ie: `'confi.my_first_key'`)
- Only `config` and `register` are available for importing from the root module `djconfig`.

3.2 License

The MIT License (MIT)

Copyright (c) 2015 Esteban Castro Borsani <ecastroborsani@gmail.com>

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.

d

- `django`, [9](#)
- `django.admin`, [10](#)
- `django.conf`, [9](#)
- `django.context_processors`, [10](#)
- `django.forms`, [10](#)
- `django.middleware`, [10](#)
- `django.utils`, [10](#)

C

Config (class in `django.config.admin`), 11
Config (class in `django.config.conf`), 9
config (in module `django.config`), 9
config() (in module `django.config.context_processors`), 10
ConfigAdmin (class in `django.config.admin`), 11
ConfigForm (class in `django.config.forms`), 10

D

django (module), 9
django.admin (module), 10
django.conf (module), 9
django.context_processors (module), 10
django.forms (module), 10
django.middleware (module), 10
django.utils (module), 10
DjangoConfigMiddleware (class in `django.config.middleware`), 10

O

override_django() (in module `django.utils`), 10

R

register (in module `django.config`), 9
register() (in module `django.config.admin`), 11
reload_maybe (in module `django.config`), 9

S

save() (`django.config.forms.ConfigForm` method), 10