django-constance Documentation

Release dev

Jazzband

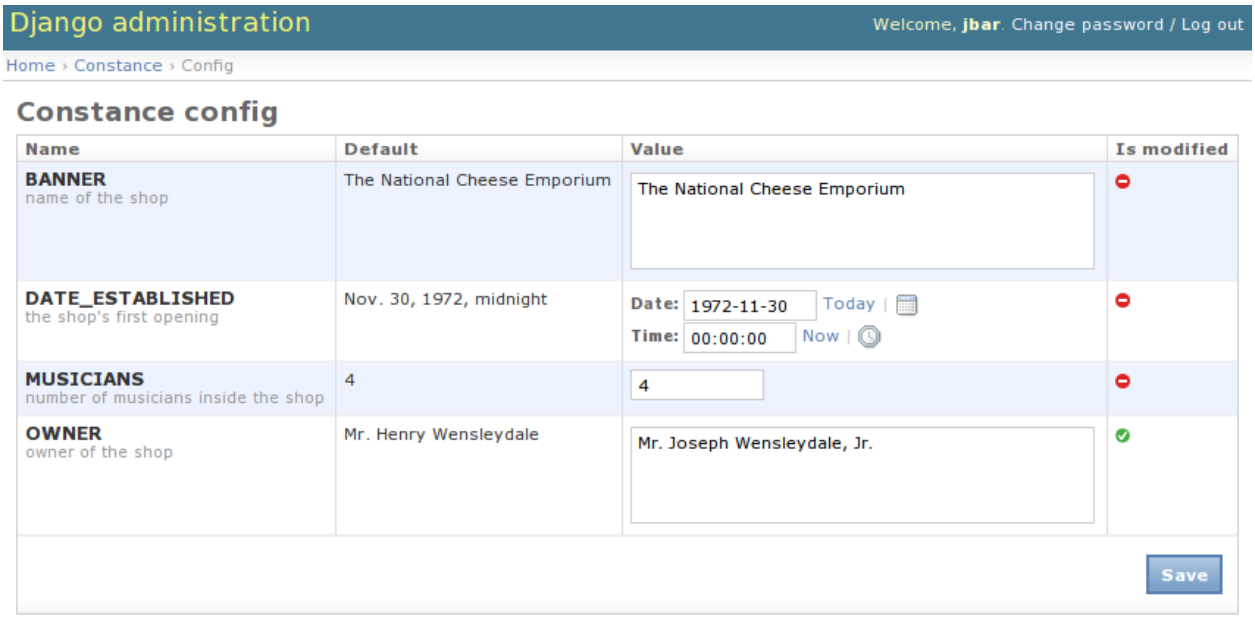
Mar 15, 2018

Contents

CHAPTER 1

Features

- Easily migrate your static settings to dynamic settings.
- Admin interface to edit the dynamic settings.



CHAPTER 2

Installation

Install from PyPI the backend specific variant of django-constance:

For the (default) Redis backend:

```
pip install "django-constance[redis]"
```

For the database backend:

```
pip install "django-constance[database]"
```

Alternatively – if you're sure that the dependencies are already installed – you can also run:

```
pip install django-constance
```

CHAPTER 3

Configuration

Modify your `settings.py`. Add `'constance'` to your `INSTALLED_APPS`, and move each key you want to turn dynamic into the `CONSTANCE_CONFIG` section, like this:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.staticfiles',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    ...
    'constance',
)

CONSTANCE_CONFIG = {
    'THE_ANSWER': (42, 'Answer to the Ultimate Question of Life, '
                       'The Universe, and Everything'),
}
```

Here, `42` is the default value for the key `THE_ANSWER` if it is not found in the backend. The other member of the tuple is a help text the admin will show.

See the *Backends* section how to setup the backend and finish the configuration.

`django-constance`'s hashes generated in different instances of the same application may differ, preventing data from being saved.

Use this option in order to skip hash verification.

```
CONSTANCE_IGNORE_ADMIN_VERSION_CHECK = True
```

CHAPTER 4

Signals

Each time a value is changed it will trigger a `config_updated` signal.

You can use it as:

```
from constance.signals import config_updated

@receiver(config_updated)
def constance_updated(sender, key, old_value, new_value, **kwargs):
    print(sender, key, old_value, new_value)
```

The sender is the `config` object, and the `updated_key` and `new_value` are the ones just changed.

This callback will get the `config` object as the first parameter so you can have an isolated function where you can access the `config` object without dealing with additional imports.

CHAPTER 5

Custom fields

You can set the field type with the third value in the `CONSTANCE_CONFIG` tuple.

The value can be one of the supported types or a string matching a key in your :setting:`CONSTANCE_ADDITIONAL_FIELDS`

The supported types are:

- `bool`
- `int`
- `float`
- `Decimal`
- `long` (on python 2)
- `str`
- `unicode` (on python 2)
- `datetime`
- `date`
- `time`

For example, to force a value to be handled as a string:

```
'THE_ANSWER': (42, 'Answer to the Ultimate Question of Life, '
                   'The Universe, and Everything', str),
```

Custom field types are supported using the dictionary :setting:`CONSTANCE_ADDITIONAL_FIELDS`.

This is a mapping between a field label and a sequence (list or tuple). The first item in the sequence is the string path of a field class, and the (optional) second item is a dictionary used to configure the field.

The `widget` and `widget_kwargs` keys in the field config dictionary can be used to configure the widget used in admin, the other values will be passed as kwargs to the field's `__init__()`

Note: Use later evaluated strings instead of direct classes for the field and widget classes:

```
CONSTANCE_ADDITIONAL_FIELDS = {
    'yes_no_null_select': ['django.forms.fields.ChoiceField', {
        'widget': 'django.forms.Select',
        'choices': ((None, "-----"), ("yes", "Yes"), ("no", "No"))
    }],
}

CONSTANCE_CONFIG = {
    'MY_SELECT_KEY': ('yes', 'select yes or no', 'yes_no_null_select'),
}
```

If you want to work with files you can use this configuration:

```
CONSTANCE_ADDITIONAL_FIELDS = {
    'image_field': ['django.forms.ImageField', {}]
}

CONSTANCE_CONFIG = {
    'LOGO_IMAGE': ('default.png', 'Company logo', 'image_field'),
}
```

When used in a template you probably need to use:

```
{% load static %}

{% get_media_prefix as MEDIA_URL %}
<img src="{{ MEDIA_URL }}{{ constance.LOGO_IMAGE }}">
```

Images are uploaded to MEDIA_ROOT.

CHAPTER 6

Ordered Fields in Django Admin

In order to Order the fields , you can use OrderedDict collection. Here is an example:

```
from collections import OrderedDict

CONSTANCE_CONFIG = OrderedDict([
    ('SITE_NAME', ('My Title', 'Website title')),
    ('SITE_DESCRIPTION', ('', 'Website description')),
    ('THEME', ('light-blue', 'Website theme')),
])
```

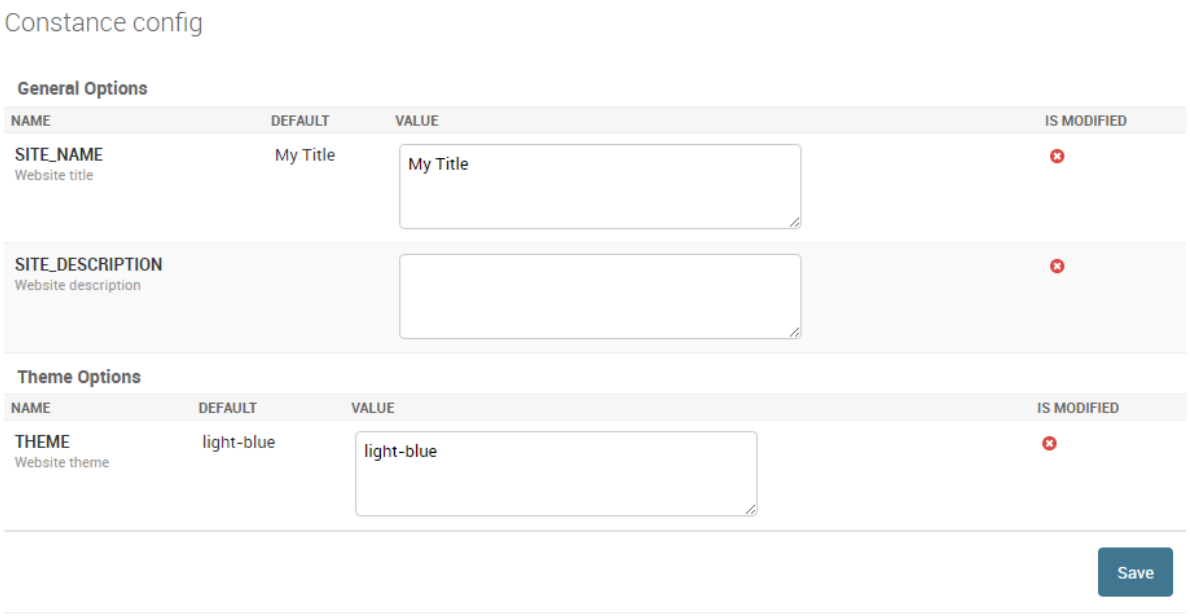
CHAPTER 7

Fieldsets

To group settings together you can define fieldsets. Here's an example:

```
CONSTANCE_CONFIG = {
    'SITE_NAME': ('My Title', 'Website title'),
    'SITE_DESCRIPTION': ('', 'Website description'),
    'THEME': ('light-blue', 'Website theme'),
}

CONSTANCE_CONFIG_FIELDSETS = {
    'General Options': ('SITE_NAME', 'SITE_DESCRIPTION'),
    'Theme Options': ('THEME',),
}
```


Constance config
General Options
NAME
DEFAULT
VALUE
IS MODIFIED
SITE_NAME
Website title
My Title
My Title
SITE_DESCRIPTION
Website description
Theme Options
NAME
DEFAULT
VALUE
IS MODIFIED
THEME
Website theme
light-blue
light-blue
Save

CHAPTER 8

Usage

Constance can be used from your Python code and from your Django templates.

8.1 Python

Accessing the config variables is as easy as importing the config object and accessing the variables with attribute lookups:

```
from constance import config

# ...

if config.THE_ANSWER == 42:
    answer_the_question()
```

8.2 Django templates

To access the config object from your template you can either pass the object to the template context:

```
from django.shortcuts import render
from constance import config

def myview(request):
    return render(request, 'my_template.html', {'config': config})
```

Or you can use the included config context processor. For Django pre-1.8, this looks like this:

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

For Django 1.8 and above, insert `'constance.context_processors.config'` at the top of your `TEMPLATES['OPTIONS']['context_processors']` list. See the Django documentation for details.

This will add the config instance to the context of any template rendered with a `RequestContext`.

Then, in your template you can refer to the config values just as any other variable, e.g.:

```
<h1>Welcome on {{ config.SITE_NAME }}</h1>
{% if config.BETA_LAUNCHED %}
    Woohoo! Head over <a href="/sekrit/">here</a> to use the beta.
{% else %}
    Sadly we haven't launched yet, click <a href="/newsletter/">here</a>
    to signup for our newletter.
{% endif %}
```

8.3 Command Line

Constance settings can be get/set on the command line with the manage command *constance*

Available options are:

list - output all values in a tab-separated format:

```
$ ./manage.py constance list
THE_ANSWER 42
SITE_NAME  My Title
```

get KEY - output a single values:

```
$ ./manage.py constance get THE_ANSWER
42
```

set KEY VALUE - set a single value:

```
$ ./manage.py constance set SITE_NAME "Another Title"
```

If the value contains spaces it should be wrapped in quotes.

Note: Set values are validated as per in admin, an error will be raised if validation fails:

E.g., given this config as per the example app:

```
CONSTANCE_CONFIG = {
    ...
    'DATE_ESTABLISHED': (date(1972, 11, 30), "the shop's first opening"),
}
```

Then setting an invalid date will fail as follow:

```
$ ./manage.py constance set DATE_ESTABLISHED '1999-12-00'
CommandError: Enter a valid date.
```

Note: If the admin fields is a *MultiValueField*, (e.g. datetime, which uses *SplitDateTimeField* by default)

---

then the separate field values need to be provided as separate arguments.

E.g., given this config:

```
CONSTANCE_CONFIG = {
    'DATETIME_VALUE': (datetime(2010, 8, 23, 11, 29, 24), 'time of the first commit'),
}
```

Then this works (and the quotes are optional):

```
./manage.py constance set DATETIME_VALUE '2011-09-24' '12:30:25'
```

This doesn't work:

```
./manage.py constance set DATETIME_VALUE '2011-09-24 12:30:25'
CommandError: Enter a list of values.
```

CHAPTER 9

Editing

Fire up your `admin` and you should see a new app called `Constance` with `THE_ANSWER` in the `Config` pseudo model.

By default changing the settings via the admin is only allowed for super users. But in case you want to use the admin's ability to implement custom authorization checks, feel free to set the `CONSTANCE_SUPERUSER_ONLY` setting to `False` and give the users or user groups access to the `constance.change_config` permission.

Fig. 9.1: The virtual application `Constance` among your regular applications.

CHAPTER 10

Custom settings form

If you aim at creating a custom settings form this is possible in the following way: You can inherit from `ConstanceAdmin` and set the `form` property on your custom admin to use your custom form. This allows you to define your own formsets and layouts, similar to defining a custom form on a standard Django ModelAdmin. This way you can fully style your settings form and group settings the way you like.

```
from constance.admin import ConstanceAdmin, ConstanceForm, Config
class CustomConfigForm(ConstanceForm):
      def __init__(self, *args, **kwargs):
        super(CustomConfigForm, self).__init__(*args, **kwargs)
        #... do stuff to make your settings form nice ...

class ConfigAdmin(ConstanceAdmin):
    change_list_form = CustomConfigForm
    change_list_template = 'admin/config/settings.html'

admin.site.unregister([Config])
admin.site.register([Config], ConfigAdmin)
```

You can also override the `get_changelist_form` method which is called in `changelist_view` to get the actual form used to change the settings. This allows you to pick a different form according to the user that makes the request. For example:

```
class SuperuserForm(ConstanceForm):
    # Do some stuff here

class MyConstanceAdmin(ConstanceAdmin):
    def get_changelist_form(self, request):
        if request.user.is_superuser:
          return SuperuserForm:
        else:
          return super(MyConstanceAdmin, self).get_changelist_form(request)
```

Note that the default method returns `self.change_list_form`.

CHAPTER 11

More documentation

11.1 Backends

Constance ships with a bunch of backends that are used to store the configuration values. By default it uses the Redis backend. To override the default please set the `CONSTANCE_BACKEND` setting to the appropriate dotted path.

11.1.1 Redis

The configuration values are stored in a redis store and retrieved using the redis-py library. Please install it like this:

```
pip install django-constance[redis]
```

Configuration is simple and defaults to the following value, you don't have to add it to your project settings:

```
CONSTANCE_BACKEND = 'constance.backends.redisd.RedisBackend'
```

Settings

There are a couple of options:

CONSTANCE_REDIS_CONNECTION

A dictionary of parameters to pass to the to Redis client, e.g.:

```
CONSTANCE_REDIS_CONNECTION = {
    'host': 'localhost',
    'port': 6379,
    'db': 0,
}
```

Alternatively you can use a URL to do the same:

```
CONSTANCE_REDIS_CONNECTION = 'redis://username:password@localhost:6379/0'
```

CONSTANCE_REDIS_CONNECTION_CLASS

An (optional) dotted import path to a connection to use, e.g.:

```
CONSTANCE_REDIS_CONNECTION_CLASS = 'myproject.myapp.mockup.Connection'
```

If you are using django-redis, feel free to use the `CONSTANCE_REDIS_CONNECTION_CLASS` setting to define a callable that returns a redis connection, e.g.:

```
CONSTANCE_REDIS_CONNECTION_CLASS = 'django_redis.get_redis_connection'
```

CONSTANCE_REDIS_PREFIX

The (optional) prefix to be used for the key when storing in the Redis database. Defaults to `'constance:'`. E.g.:

```
CONSTANCE_REDIS_PREFIX = 'constance:myproject:'
```

11.1.2 Database

The database backend is optional and stores the configuration values in a standard Django model. It requires the package django-picklefield for storing those values. Please install it like so:

```
pip install django-constance[database]
```

You must set the `CONSTANCE_BACKEND` Django setting to:

```
CONSTANCE_BACKEND = 'constance.backends.database.DatabaseBackend'
```

Then add the database backend app to your `INSTALLED_APPS` setting to make sure the data model is correctly created:

```
INSTALLED_APPS = (
    # other apps
    'constance.backends.database',
)
```

Please make sure to apply the database migrations:

```
python manage.py migrate database
```

Note: If you're upgrading Constance to 1.0 and use Django 1.7 or higher please make sure to let the migration system know that you've already created the tables for the database backend.

You can do that using the `--fake` option of the migrate command:

```
python manage.py migrate database --fake
```

Just like the Redis backend you can set an optional prefix that is used during database interactions (it defaults to an empty string, `''`). To use something else do this:

```
CONSTANCE_DATABASE_PREFIX = 'constance:myproject:'
```

Caching

The database backend has the ability to automatically cache the config values and clear them when saving. Assuming you have a `CACHES` setting set you only need to set the the `CONSTANCE_DATABASE_CACHE_BACKEND` setting to the name of the configured cache backend to enable this feature, e.g. "default":

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211',
    }
}
CONSTANCE_DATABASE_CACHE_BACKEND = 'default'
```

Warning: The cache feature won't work with a cache backend that is incompatible with cross-process caching like the local memory cache backend included in Django because correct cache invalidation can't be guaranteed.

Note: By default Constance will autofill the cache on startup and after saving any of the config values. If you want to disable the cache simply set the `CONSTANCE_DATABASE_CACHE_AUTOFILL_TIMEOUT` setting to `None`.

11.2 Testing

Testing how your app behaves with different config values is achieved with the *override_config* class. This intentionally mirrors the use of Django's `override_setting`.

class override_config(***kwargs*)

Replaces key-value pairs in the config. Use as decorator or context manager.

11.2.1 Usage

It can be used as a decorator at the `TestCase` level, the method level and also as a context manager.

```
from constance import config
from constance.test import override_config

from django.test import TestCase


@override_config(YOUR_NAME="Arthur of Camelot")
class ExampleTestCase(TestCase):

    def test_what_is_your_name(self):
```

```
        self.assertEqual(config.YOUR_NAME, "Arthur of Camelot")

    @override_config(YOUR_QUEST="To find the Holy Grail")
    def test_what_is_your_quest(self):
        self.assertEqual(config.YOUR_QUEST, "To find the Holy Grail")

    def test_what_is_your_favourite_color(self):
        with override_config(YOUR_FAVOURITE_COLOR="Blue?"):
            self.assertEqual(config.YOUR_FAVOURITE_COLOR, "Blue?")
```

11.3 Changelog

11.3.1 v2.1.0 (2018/02/07)

- Move inline JavaScript to constance.js.
- Remove translation from the app name.
- Added file uploads.
- Update information on template context processors.
- Allow running set while database is not created.
- Moved inline css/javascripts out to their own files.
- Add French translations.
- Add testing for all supported Python and Django versions.
- Preserve sorting from fieldset config.
- Added datetime.timedelta support.
- Added Estonian translations.
- Account for server timezone for Date object.

11.3.2 v2.0.0 (2017/02/17)

- **BACKWARD INCOMPATIBLE** Added the old value to the config_updated signal.
- Added a *get_changelist_form* hook in the ModelAdmin.
- Fix create_perm in apps.py to use database alias given by the post_migrate signal.
- Added tests for django 1.11.
- Fix Reset to default to work with boolean/checkboxes.
- Fix handling of MultiValueField's (eg SplitDateTimeField) on the command line.

11.3.3 v1.3.4 (2016/12/23)

- Fix config ordering issue
- Added localize to check modified flag
- Allow to rename Constance in Admin

- Preserve line breaks in default value
- Added functionality from django-constance-cli
- Added “Reset to default” feature

11.3.4 v1.3.3 (2016/09/17)

- Revert broken release

11.3.5 v1.3.2 (2016/09/17)

- Fixes a bug where the signal was sent for fields without changes

11.3.6 v1.3.1 (2016/09/15)

- Improved the signal path to avoid import errors
- Improved the admin layout when using fieldsets

11.3.7 v1.3 (2016/09/14)

- **BACKWARD INCOMPATIBLE** Dropped support for Django < 1.8).
- Added ordering constance fields using OrderedDict
- Added a signal when updating constance fields

11.3.8 v1.2.1 (2016/09/1)

- Added some fixes to small bugs
- Fix cache when key changes
- Upgrade django_redis connection string
- Autofill cache key if key is missing
- Added support for fieldsets

11.3.9 v1.2 (2016/05/14)

- Custom Fields were added as a new feature
- Added documentation on how to use Custom settings form
- Introduced `CONSTANCE_IGNORE_ADMIN_VERSION_CHECK`
- Improved documentation for `CONSTANCE_ADDITIONAL_FIELDS`

11.3.10 v1.1.2 (2016/02/08)

- Moved to Jazzband organization (https://github.com/jazzband/django-constance)
- Added Custom Fields
- Added Django 1.9 support to tests
- Fixes icons for Django 1.9 admin

11.3.11 v1.1.1 (2015/10/01)

- Fixed a regression in the 1.1 release that prevented the rendering of the admin view with constance values when using the context processor at the same time.

11.3.12 v1.1 (2015/09/24)

- **BACKWARD INCOMPATIBLE** Dropped support for Python 2.6 The supported versions are 2.7, 3.3 (on Django < 1.9) and 3.4.
- **BACKWARD INCOMPATIBLE** Dropped support for Django 1.4, 1.5 and 1.6 The supported versions are 1.7, 1.8 and the upcoming 1.9 release
- Added compatibility to Django 1.8 and 1.9.
- Added Spanish and Chinese (`zh_CN`) translations.
- Added *`override_config`* decorator/context manager for easy *testing*.
- Added the ability to use linebreaks in config value help texts.
- Various testing fixes.

11.3.13 v1.0.1 (2015/01/07)

- Fixed issue with import time side effect on Django >= 1.7.

11.3.14 v1.0 (2014/12/04)

- Added docs and set up Read The Docs project:

  https://django-constance.readthedocs.io/

- Set up Transifex project for easier translations:

  https://www.transifex.com/projects/p/django-constance

- Added autofill feature for the database backend cache which is enabled by default.
- Added Django>=1.7 migrations and moved South migrations to own folder. Please upgrade to South>=1.0 to use the new South migration location.

  For Django 1.7 users that means running the following to fake the migration:

  ```
  django-admin.py migrate database --fake
  ```

- Added consistency check when saving config values in the admin to prevent accidentally overwriting other users' changes.

- Fixed issue with South migration that would break on MySQL.
- Fix compatibility with Django 1.6 and 1.7 and current master (to be 1.8).
- Fixed clearing database cache en masse by applying prefix correctly.
- Fixed a few translation related issues.
- Switched to tox as test script.
- Fixed a few minor cosmetic frontend issues (e.g. padding in admin table header).
- Deprecated a few old settings:

deprecated	replacement
CONSTANCE_CONNECTION_CLASS	CONSTANCE_REDIS_CONNECTION_CLASS
CONSTANCE_CONNECTION	CONSTANCE_REDIS_CONNECTION
CONSTANCE_PREFIX	CONSTANCE_REDIS_PREFIX

- The undocumented feature to use an environment variable called CONSTANCE_SETTINGS_MODULE to define which module to load settings from has been removed.

11.3.15 v0.6 (2013/04/12)

- Added Python 3 support. Supported versions: 2.6, 2.7, 3.2 and 3.3. For Python 3.x the use of Django > 1.5.x is required.
- Fixed a serious issue with ordering in the admin when using the database backend. Thanks, Bouke Haarsma.
- Switch to django-discover-runner as test runner to be able to run on Python 3.
- Fixed an issue with refering to static files in the admin interface when using Django < 1.4.

11.3.16 v0.5 (2013/03/02)

- Fixed compatibility with Django 1.5's swappable model backends.
- Converted the key field of the database backend to use a CharField with uniqueness instead of just TextField.

  For South users we provide a migration for that change. First you have to "fake" the initial migration we've also added to this release:

```
django-admin.py migrate database --fake 0001
```

  After that you can run the rest of the migrations:

```
django-admin.py migrate database
```

- Fixed compatibility with Django>1.4's way of refering to static files in the admin.
- Added ability to add custom authorization checks via the new CONSTANCE_SUPERUSER_ONLY setting.
- Added Polish translation. Thanks, Janusz Harkot.
- Allow CONSTANCE_REDIS_CONNECTION being an URL instead of a dict.
- Added CONSTANCE_DATABASE_PREFIX setting allow setting a key prefix.
- Switched test runner to use django-nose.

CHAPTER 12

Indices and tables

- genindex
- modindex
- search

Index

O