django-constance Documentation

Release dev

Jazzband

Aug 21, 2023

Contents

1 Features **1**

2 Quick Installation **3**

3 Configuration **5**

4 Signals **7**

5 Custom fields **9**

6 Ordered Fields in Django Admin **11**

7 Fieldsets **13**

8 Fieldsets collapsing **15**

9 Field internationalization **17**

10 Usage **19**
- 10.1 Python . . . 19
- 10.2 Django templates . . . 19
- 10.3 Command Line . . . 20

11 Editing **23**

12 Custom settings form **25**

13 More documentation **27**
- 13.1 Backends . . . 27
- 13.2 Testing . . . 30
- 13.3 Changelog . . . 32

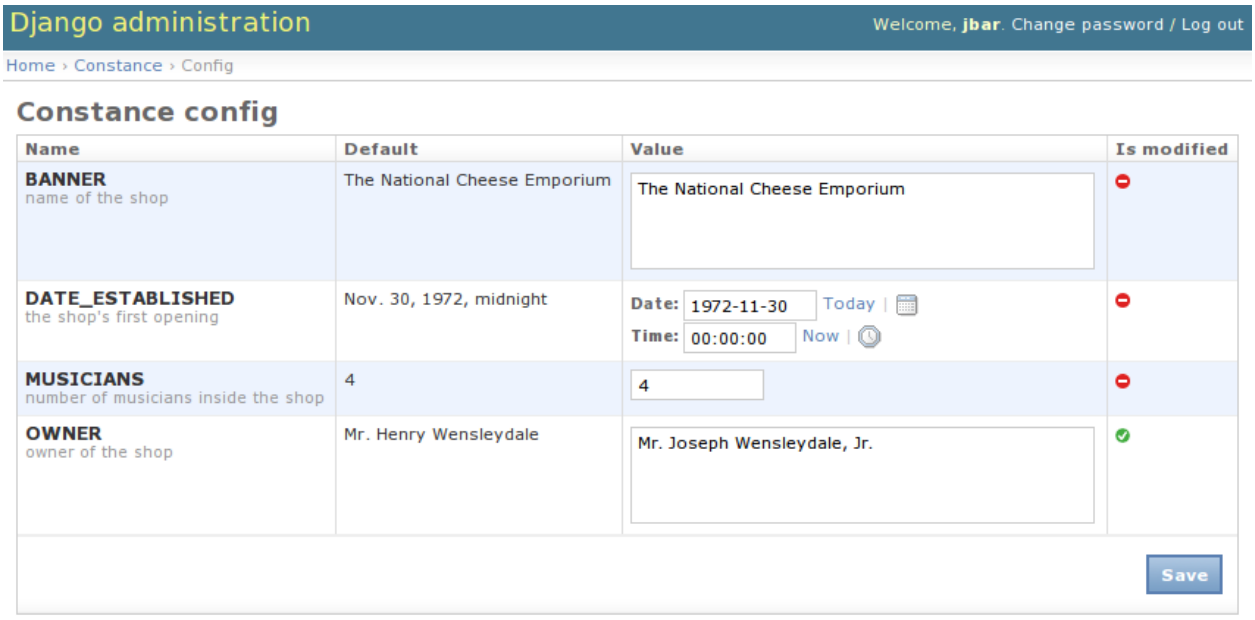
14 Indices and tables **39**

Index **41**

CHAPTER 1

Features

- Easily migrate your static settings to dynamic settings.
- Edit the dynamic settings in the Django admin interface.



CHAPTER 2

Quick Installation

For complete installation instructions, including how to install the database backend, see *Backends*.

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'),
}
```

Note: Add constance *before* your project apps.

Note: If you use admin extensions like Grapelli, `'constance'` should be added in `INSTALLED_APPS` *before* those extensions.

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 CONSTANCE_IGNORE_ADMIN_VERSION_CHECK 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.

```
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 `key` and `new_value` are the changed settings.

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`
- `str`
- `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 images or 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 }}{{ config.LOGO_IMAGE }}">
```

Images and files are uploaded to `MEDIA_ROOT` by default. You can specify a subdirectory of `MEDIA_ROOT` to use instead by adding the `CONSTANCE_FILE_ROOT` setting. E.g.:

```
MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
CONSTANCE_FILE_ROOT = 'constance'
```

This will result in files being placed in `media/constance` within your `BASE_DIR`. You can use deeper nesting in this setting (e.g. `constance/images`) but other relative path components (e.g. `../`) will be rejected.

CHAPTER 6

Ordered Fields in Django Admin

To sort the fields, you can use an OrderedDict:

```
from collections import OrderedDict

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

CHAPTER 7

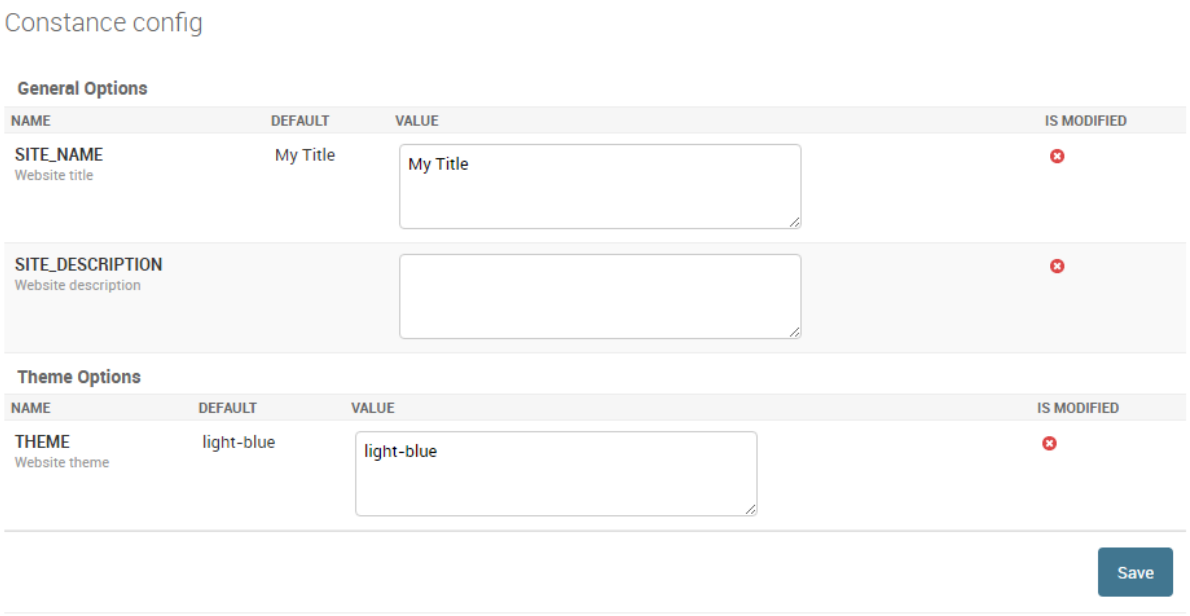
Fieldsets

You can define fieldsets to group settings together:

```
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',),
}
```

Note: CONSTANCE_CONFIG_FIELDSETS must contain all fields from CONSTANCE_CONFIG.


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

Fieldsets collapsing

To make some fieldsets collapsing you can use new format in CONSTANCE_CONFIG_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': {
        'fields': ('SITE_NAME', 'SITE_DESCRIPTION'),
        'collapse': True
    },
    'Theme Options': ('THEME',),
}
```

CHAPTER 9

Field internationalization

Field description and fieldset headers can be integrated into Django's internationalization using the `gettext_lazy` function. Note that the `CONSTANCE_CONFIG_FIELDSETS` must be converted to a tuple instead of dict as it is not possible to have lazy proxy objects as dictionary keys in the settings file. Example:

```
from django.utils.translation import gettext_lazy as _

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

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

CHAPTER 10

Usage

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

10.1 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()
```

10.2 Django templates

To access the config object from your template you can 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})
```

You can also use the included context processor.

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 %}
```

10.3 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"),
}
```

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 field is a *MultiValueField*, then the separate field values need to be provided as separate arguments.

E.g., a datetime using *SplitDateTimeField*:

```
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.
```

remove_stale_keys - delete all stale records in database:

```
$ ./manage.py constance remove_stale_keys
```

Record is considered stale if it exists in database but absent in config

CHAPTER 11

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 superusers. To change this, feel free to set the `CONSTANCE_SUPERUSER_ONLY` setting to `False` and give users or user groups access to the `constance.change_config` permission.

Fig. 1: The virtual application `Constance` among your regular applications.

CHAPTER 12

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, Config
from constance.forms import ConstanceForm
class CustomConfigForm(ConstanceForm):
      def __init__(self, *args, **kwargs):
        super().__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().get_changelist_form(request)
```

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

CHAPTER 13

More documentation

13.1 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.

13.1.1 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'
```

Default redis backend retrieves values every time. There is another redis backend with local cache. *CachingRedisBackend* stores the value from a redis to memory at first access and checks a value ttl at next. Configuration installation is simple:

```
CONSTANCE_BACKEND = 'constance.backends.redisd.CachingRedisBackend'
# optionally set a value ttl
CONSTANCE_REDIS_CACHE_TIMEOUT = 60
```

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:'
```

CONSTANCE_REDIS_PICKLE_VERSION

The (optional) protocol version of pickle you want to use to serialize your python objects when storing in the Redis database. Defaults to `pickle.DEFAULT_PROTOCOL`. E.g.:

```
CONSTANCE_REDIS_PICKLE_VERSION = pickle.DEFAULT_PROTOCOL
```

You might want to pin this value to a specific protocol number, since `pickle.DEFAULT_PROTOCOL` means different things between versions of Python.

CONSTANCE_REDIS_CACHE_TIMEOUT

The (optional) ttl of values in seconds used by *CachingRedisBackend* for storing in a local cache. Defaults to *60* seconds.

13.1.2 Database

Database backend stores configuration values in a standard Django model. It requires the package django-picklefield for storing those values.

You must set the `CONSTANCE_BACKEND` Django setting to:

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

Please make sure to apply the database migrations:

```
python manage.py migrate
```

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.

If you try this, Constance will throw an error and refuse to let your application start. You can work around this by subclassing `constance.backends.database.DatabaseBackend` and overriding *__init__* to remove the check. You'll want to consult the source code for that function to see exactly how.

We're deliberately being vague about this, because it's dangerous; the behavior is undefined, and could even cause your app to crash. Nevertheless, there are some limited circumstances in which this could be useful, but please think carefully before going down this path.

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`.

13.1.3 Memory

The configuration values are stored in a memory and do not persist between process restarts. In order to use this backend you must set the `CONSTANCE_BACKEND` Django setting to:

```
CONSTANCE_BACKEND = 'constance.backends.memory.MemoryBackend'
```

The main purpose of this one is to be used mostly for testing/developing means, so make sure you intentionally use it on production environments.

13.2 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.

13.2.1 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?")
```

13.2.2 Pytest usage

Django-constance provides pytest plugin that adds marker `@pytest.mark.override_config()`. It handles config override for module/class/function, and automatically revert any changes made to the constance config values when test is completed.

pytest.mark.**override_config**(***kwargs*)
Specify different config values for the marked tests in kwargs.

Module scope override

```
pytestmark = pytest.mark.override_config(API_URL="/awesome/url/")

def test_api_url_is_awesome():
    ...
```

Class/function scope

```
from constance import config

@pytest.mark.override_config(API_URL="/awesome/url/")
class SomeClassTest:
    def test_is_awesome_url(self):
        assert config.API_URL == "/awesome/url/"

    @pytest.mark.override_config(API_URL="/another/awesome/url/")
    def test_another_awesome_url(self):
        assert config.API_URL == "/another/awesome/url/"
```

If you want to use override as a context manager or decorator, consider using

```
from constance.test.pytest import override_config

def test_override_context_manager():
    with override_config(BOOL_VALUE=False):
        ...
# or
@override_config(BOOL_VALUE=False)
def test_override_context_manager():
    ...
```

Pytest fixture as function or method parameter (NOTE: no import needed as fixture is available globally)

```
def test_api_url_is_awesome(override_config):
    with override_config(API_URL="/awesome/url/"):
        ...
```

Any scope, auto-used fixture alternative can also be implemented like this

```
@pytest.fixture(scope='module', autouse=True)  # e.g. module scope
def api_url(override_config):
    with override_config(API_URL="/awesome/url/"):
        yield
```

13.2.3 Memory backend

If you don't want to rely on any external services such as Redis or database when running your unittests you can select `MemoryBackend` for a test Django settings file

```
CONSTANCE_BACKEND = 'constance.backends.memory.MemoryBackend'
```

It will provide simple thread-safe backend which will reset to default values after each test run.

13.3 Changelog

13.3.1 v3.1.0 (2023/08/21)

- Add support for using a subdirectory of *MEDIA_ROOT* for file fields
- Remove pypy from tox tests

13.3.2 v3.0.0 (2023/07/27)

- Refactor database backend Backward incompatible changes: remove `'constance.backends.database'` from `INSTALLED_APPS`
- Dropped support for python < 3.7 and django < 3.2
- Example app now supports django 4.1
- Add support for django 4.2
- Forward the request when saving the admin changelist form

13.3.3 v2.9.1 (2022/08/11)

- Add support for gettext in fieldset headers
- Add support for Django 4.1
- Fix text format for MultiValueField usage

13.3.4 v2.9.0 (2022/03/11)

- Added arabic translation
- Add concrete_model class attribute to fake admin model
- Added tests for django 3.2
- Fix do not detect datetime fields as date type
- Added support for python 3.10
- Fixes for Ukrainian locale
- Added documentation for constance_dbs config
- Add caching redis backend
- Serialize according to widget
- Add default_auto_field to database backend

13.3.5 v2.8.0 (2020/11/19)

- Prevent reset to default for file field
- Fields_list can be a dictionary, when a fieldset is defined as collapsible
- Create and add fa language translations files

- Respect other classes added by admin templates
- Removed deprecated url()
- Use gettext_lazy instead of ugettext_lazy
- Updated python and django version support

13.3.6 v2.7.0 (2020/06/22)

- Deleted south migrations
- Improve grammar of documentation index file
- Simplify documentation installation section
- Fix IntegrityError after 2.5.0 release (Allow concurrent calls to *DatabaseBackend.set()* method)
- Make groups of fieldsets collapsable
- Allow override_config for pytest
- Put back wheel generation in travis
- Fix wrong "is modified" in admin for multi line strings
- Switch md5 to sha256
- Fix Attempts to change config values fail silently and appear to succeed when user does not have change permissions
- Make constance app verbose name translatable
- Update example project for Django>2
- Add anchors in admin for constance settings
- Added a sticky footer in django constance admin
- Add memory backend
- Added Ukrainian locale
- Added lazy checks for pytest

13.3.7 v2.6.0 (2020/01/29)

- Drop support py<3.5 django<2.2
- Set pickle protocol version for the Redis backend
- Add a command to delete stale records

13.3.8 v2.5.0 (2019/12/23)

- Made results table responsive for Django 2 admin
- Add a Django system check that CONFIG_FIELDSETS accounts for all of CONFIG
- Rewrite set() method of database backend to reduce number of queries
- Fixed "can't compare offset-naive and offset-aware datetimes" when USE_TZ = True
- Fixed compatibility issue with Django 3.0 due to django.utils.six

- Add Turkish language

13.3.9 v2.4.0 (2019/03/16)

- Show not existing fields in field_list
- Drop Django<1.11 and 2.0, fix tests vs Django 2.2b
- Fixed "Reset to default" button with constants whose name contains a space
- Use default_storage to save file
- Allow null & blank for PickleField
- Removed Python 3.4 since is not longer supported

13.3.10 v2.3.1 (2018/09/20)

- Fixes javascript typo.

13.3.11 v2.3.0 (2018/09/13)

- Added zh_Hans translation.
- Fixed TestAdmin.test_linebreaks() due to linebreaksbr() behavior change on Django 2.1
- Improved chinese translation
- Fix bug of can't change permission chang_config's name
- Improve consistency of reset value handling for *date*
- Drop support for Python 3.3
- Added official Django 2.0 support.
- Added support for Django 2.1

13.3.12 v2.2.0 (2018/03/23)

- Fix ConstanceForm validation.
- *CONSTANCE_DBS* setting for directing constance permissions/content_type settings to certain DBs only.
- Added config labels.
- Updated italian translations.
- Fix *CONSTANCE_CONFIG_FIELDSETS* mismatch issue.

13.3.13 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.

13.3.14 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.

13.3.15 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

13.3.16 v1.3.3 (2016/09/17)

- Revert broken release

13.3.17 v1.3.2 (2016/09/17)

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

13.3.18 v1.3.1 (2016/09/15)

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

13.3.19 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

13.3.20 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

13.3.21 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`

13.3.22 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

13.3.23 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.

13.3.24 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.

13.3.25 v1.0.1 (2015/01/07)

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

13.3.26 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.

13.3.27 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.

13.3.28 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 14

Indices and tables

- genindex
- modindex
- search

Index

O

override_config (*built-in class*), 30

P

pytest.mark.override_config() (*built-in function*), 30