django CMS Helper Documentation

Release 0.5.0

Iacopo Spalletti

March 13, 2016

Contents

django CMS Helper is a set of commandline utilities to help developers of applications for the django CMS ecosystem.

It's a modified version of django CMS's own *develop.py* script, allowing you to work with your application even without having a full project - which is not always possible or convenient - set up.

It does this by spawning its own virtual project - a basic generic project built in to itself - that's ready to integrate with your application with just a little extra configuration.

The utilities provided:

- help setting up and running tests
- give you access to a Django shell
- run the Django check command
- compile and update locale message
- help manage Django and South migrations
- perform static analysis using pyflakes
- build an authors list automatically
- setting up a Django environment

django CMS Helper was created by Iacopo Spalletti.

CHAPTER 1

Installation

Installing from pip:

```
pip install djangocms-helper
```

or:

```
pip install djangocms-helper[cms]
```

to install it along with django CMS

Installing from source:

```
pip install git+https://github.com/nephila/djangocms-helper#egg=djangocms-helper
```

CHAPTER 2

Requirements

- django CMS 3.0 (django CMS 3.0.4 is required for pyflake command), optional; required only to work with `--cms` option
- docopt
- tox
- dj-database-url

Warning: Since version 0.7 django CMS is no more a hard dependency; install it manually to enable `--cms` option

2.1 How to use django CMS Helper

We'll assume that you have an application for django CMS that you're working on.

Once you have django CMS installed, it'll be available using `djangocms-helper` command.

`cd` into the root directory of your application (that is, the outer directory containing its `setup.py`). You need to be here to run the `djangocms-helper` command.

2.1.1 Running django CMS Helper command

Try it:

```
djangocms-helper <myapp> test --cms  # change <myapp> to your application's actual name
```

It'll spawn its virtual project and run your tests in it. You should see some output along these lines (there may well be some other output before it gets to this stage):

```
Creating test database for alias 'default'...
F
======================================================================
FAIL: test_bad_maths (djangocms_maths.tests.SmokeTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "./djangocms_maths/tests.py", line 6, in test_bad_maths
    self.assertEqual(1 + 1, 3)
AssertionError: 2 != 3
```

```
----------------------------------------------------------------------
Ran 1 test in 0.000s

FAILED (failures=1)
```

All commands take a form similar to the one you've just run, sharing the basic command structure:

```
djangocms-helper <application> <command> [options ...]
```

where **<application>** is the Django application name and **<command>** is one of the available commands. Options vary for each command.

But I haven't written any tests yet!

It helps if you actually have some tests of course - if you don't, simply create a `tests.py` file in your application (not in this directory, but in the package directory, alongside its models and views and so on):

```
from django.test import TestCase

class SmokeTest(TestCase):

    # a deliberately-failing test
    def test_bad_maths(self):
        self.assertEqual(1 + 1, 3)
```

2.1.2 The –cms option

You'll need the `--cms` option most of the time. It sets up the virtual project appropriately for django CMS, providing the required configuration (see *–cms option* for details).

2.1.3 Other commands

Try a couple of the other commands; they're mostly self-explanatory:

```
djangocms-helper <myapp> shell --cms  # start a Django shell for the virtual project

djangocms-helper <myapp> check --cms  # runs the Django check command

djangocms-helper <myapp> cms_check  # runs the django CMS check command
```

Note that the last of these doesn't require the `--cms` option, because of course that is implied anyway by `cms_check`.

2.1.4 Integrated runner

In some contexts running commands by using the complete syntax can be clunky or unfeasible.

django CMS Helper contains function that allows to run the commands with a much shorter syntax:

```
python cms_helper.py
```

to run tests

Or:

```
python cms_helper.py server
```

to invoke a server.

See Integrated runner for details.

2.1.5 Sphinx integration

When documenting a project using Sphinx autodoc, you mostly need a proper project setup, because the imports in your application's modules will trigger Django setup code anyway.

Using the *Naked setup* it's easy to let helper setup an environment for you:

- setup the *Naked setup*
- add the following code to sphinx `conf.py`:

```
sys.path.insert(0, os.path.abspath('..'))
import cms_helper
cms_helper.setup()
```

2.2 django CMS Helper reference

2.2.1 Commands

Commands take the general form:

```
djangocms-helper <application> <command> [options ...]
```

where **<application>** is the Django application name and **<command>** is a Django supported command, *or* one of the djangocms-helper commands detailed below. Options vary for each command.

Common options

- `--extra-settings=path`: loads the extra settings from the provided file instead of the default `cms_helper.py`
- `cms`: loads django CMS specific options (see *–cms option* for details)

Django commands

django CMS Helper supports any Django command available according to the project setup; the general syntax is:

```
djangocms-helper <application> <command> [options] [--extra-settings=</path/to/settings.py>] [--cms]
```

Example: `djangocms-helper some_application shell --cms`

Arguments

- `<command>` is any available Django command
- `[options]` is any option/argument accepted by the above command

test

```
djangocms-helper <application> test [--failfast] [--migrate] [<test-label>...] [--xvfb] [--runner=<te
```

Example: `djangocms-helper some_application test --cms`

Runs the application's test suite in django CMS Helper's virtual environment.

Arguments

- `<test-label>`: a space-separated list of tests to run; test labels depends on the runner test suite building protocol, please, refer to the runner documentation to know the test label format;

Options

- `--runner`: custom test runner to use in dotted path notation;
- `--runner-options=<option1>,<option2>`: comma separated list of command line options for the test runner: e.g. `--runner-options=--with-coverage,--cover-package=my_package`
- `--failfast`: whether to stop at first test failure;
- `--migrate`: use migrations (default);
- **`--persistent`: use persistent storage for media and static; if given without arguments the** storage is created in `data` directory in the root of the application; if argument is provided, it's used as parent path (either absolute or relative) for `media` and `static`;
- `--no-migrate`: skip migrations;
- `--boilerplate`: adds `aldryn-boilerplates` configuration to settings;
- `--xvfb`: whether to configure `xvfb` (for frontend tests);
- `--nose-runner`: use django nose test suite
- `--simple-runner` use Django DjangoTestSuiteRunner
- `--native` use the native Django command: the use of this option is **incompatible** with the options above.

Test structure

Currently two different tests layouts are supported:

- tests outside the application module:

```
setup.py
tests
    __init__.py
    test_module1.py
    ....
```

- tests inside the application:

```
setup.py
application
    tests
        __init__.py
```

```
            test_module1.py
            ...
```

Depending on the used test runner you may need to setup your tests accordingly.

Currently supported test runners are:

- Django's DiscoverRunner (default on Django 1.6+)
- Nose's NoseTestSuiteRunner (option `--nose-runner`)

You can also specify your own custom runner with the `--runner` option.

cms_check

```
djangocms-helper <application> cms_check [--extra-settings=</path/to/settings.py>] [--migrate]
```

Runs the django CMS `cms check` command.

Example: `djangocms-helper some_application cms_check`

update and compile locales

```
djangocms-helper <application> makemessages [--extra-settings=</path/to/settings.py>] [--cms] [--loca
djangocms-helper <application> compilemessages [--extra-settings=</path/to/settings.py>] [--cms]
```

Examples:

```
djangocms-helper some_application makemessages --cms
djangocms-helper some_application compilemessages --cms
```

These two commands compiles and update the locale messages.

Options

- **`--locale=locale`: `makemessages` allows a single option to choose the locale to update.** If not provided **en** is used.

makemigrations

```
djangocms-helper <application> makemigrations [--extra-settings=</path/to/settings.py>] [--cms] [--me
```

Updates the application migrations (south migrations or Django migrations according to the current installed Django version). For South, it automatically handles *initial* and *auto* options.

Options

- `--merge`: Enable fixing of migration conflicts (for Django 1.7+ only)
- `--empty`: It generates an empty migration for customisations
- `--dry-run`: Does not create migrations file (for Django 1.7+ only)

Arguments

- `<extra-applications>`: Spaces separated list of applications to migrate

squashmigrations

```
djangocms-helper <application> squashmigrations <migration-name>
```

Runs the `squashmigrations` command. It operates on the current application.

Arguments

- `<migration-name>`: Squash migrations until this migration

pyflakes

```
djangocms-helper <application> pyflakes [--extra-settings=</path/to/settings.py>] [--cms]
```

Performs static analysis using pyflakes, with the same configuration as django CMS.

authors

```
djangocms-helper <application> authors [--extra-settings=</path/to/settings.py>] [--cms]
```

Generates an authors list from the git log, in a form suitable for the **AUTHORS** file.

server

```
djangocms-helper <application> server [--port=<port>] [--bind=<bind>] [--extra-settings=</path/to/set
```

Starts a runserver instance.

2.3 Settings with django CMS Helper

2.3.1 Extra settings

django CMS Helper provide a basic set of settings, you'll probably need to provide your own.

Extra settings can be provided by creating a `cms_helper.py` file in the application root directory and providing the settings as a dictionary named `HELPER_SETTINGS`:

```
HELPER_SETTINGS={
    'INSTALLED_APPS': [
        'any_django_app',
    ],
    'ANY_SETTING: False,
    ...
}
```

An alternative, and possibly clearer form is:

```
HELPER_SETTINGS=dict(
    INSTALLED_APPS=[
        'any_django_app',
    ],
    ANY_SETTING=False,
    ...
)
```

By default any setting option provided in `cms_helper.py` will override the default ones.

Special settings

The following settings will not override the defaults ones, but they are appended to the defaults to make easier to customise the configuration:

- `INSTALLED_APPS`
- `TEMPLATE_CONTEXT_PROCESSORS`
- `TEMPLATE_LOADERS`
- `TEMPLATE_DIRS`
- `MIDDLEWARE_CLASSES`

Other extra setting:

- `TOP_INSTALLED_APPS`: items in this setting will be inserted on top of `INSTALLED_APPS` (e.g.: to control the templates and static files override from standard applications configured by djangocms-helper).
- `TOP_MIDDLEWARE_CLASSES`: items in this setting will be inserted on top of `MIDDLEWARE_CLASSES`.

Django 1.8 support

All `TEMPLATES_` settings from Django 1.6/1.7 are automatically translated to Django 1.8 `TEMPLATE` setting. To support both, just use the **old** names, and `djangocms-helper` will take care of converting.

2.3.2 default settings

These are the applications, context processors and middlewares loaded by default

Applications:

```
'django.contrib.contenttypes',
'django.contrib.auth',
'django.contrib.sessions',
'django.contrib.sites',
'django.contrib.staticfiles',
'django.contrib.admin',
'djangocms_helper.test_data',  # this provides basic templates and urlconf
'django.contrib.messages',
```

Template context processors:

```
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
'django.core.context_processors.i18n',
'django.core.context_processors.csrf',
```

```
'django.core.context_processors.debug',
'django.core.context_processors.tz',
'django.core.context_processors.request',
'django.core.context_processors.media',
'django.core.context_processors.static',
```

Note: On Django 1.8 these are translated to the new path `django.template.context_processors.*`

Middlewares:

```
'django.middleware.http.ConditionalGetMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.middleware.locale.LocaleMiddleware',
'django.middleware.common.CommonMiddleware',
```

2.3.3 –cms option

When using `--cms` option, `INSTALLED_APPS`, `TEMPLATE_CONTEXT_PROCESSORS` and `MIDDLEWARE_CLASSES` related to django CMS are added to the default settings so you won't need to provide it yourself.

Applications:

```
'djangocms_admin_style',
'mptt',
'cms',
'menus',
'sekizai',
```

When django CMS 3.1+ is used, `treebeard` is configured instead of `mptt`.

Template context processors:

```
'cms.context_processors.cms_settings',
'sekizai.context_processors.sekizai',
```

Middlewares:

```
'cms.middleware.language.LanguageCookieMiddleware',
'cms.middleware.user.CurrentUserMiddleware',
'cms.middleware.page.CurrentPageMiddleware',
'cms.middleware.toolbar.ToolbarMiddleware',
```

`djangocms-helper` discovers automtically the South / Django migrations layout and configure the settings accordingly. As of the current version `filer`, `djangocms_text_ckeditor`, `cmplugin_filer` are supported.

2.4 Integrated runner

django CMS Helper provide a runner to invoke the commands without requiring the `djangocms-helper` file; this can be useful to invoke tests with coverage or to have a simpler syntax to remember.

Typically you'd setup the runner function in the *extra settings file*:

```
HELPER_SETTINGS={
    'INSTALLED_APPS': [
        'any_django_app',
    ],
    'ANY_SETTING: False,
    ...
}


def run():
    from djangocms_helper import runner
    runner.cms('my_app')

if __name__ == "__main__":
    run()
```

with the above code in place you can run any django CMS Helper command as:

```
python cms_helper.py <command>
```

and adding the `test_suite` argument to `setup.py`:

```
setup(
    ...
    test_suite='cms_helper.run',
    ...
)
```

you can invoke the tests with:

```
python setup.py test
```

2.4.1 Django environment

If you don't need django CMS, you can use a runner function with no CMS attached:

```
def run():
    from djangocms_helper import runner
    runner.run('my_app')

if __name__ == "__main__":
    run()
```

Warning: The runner **must** be invoked from the **settings** file. The runner takes care of setting up the file in which is invoked as the `extra_settings` file.

2.4.2 Naked setup

Sometimes you just want to properly setup a Django environment without running any commands (e.g: when building Sphinx docs using autodoc). Naked setup allows to do so:

```
def setup():
    import sys
    from djangocms_helper import runner
    runner.setup('my_app', sys.modules[__name__], use_cms=True)
```

Warning: The runner **must** be invoked from the **settings** file. The runner takes care of setting up the file in which is invoked as the `extra_settings` file.

2.5 Base test class

class `djangocms_helper.base_test.`**`BaseTestCase`**(*methodName='runTest'*)

Utils class that provides some helper methods to setup and interact with Django testing framework.

`captured_output`(**args*, ***kwds*)

Context manager that patches stdout / stderr with StringIO and return the instances. Use it to test output

Returns stdout, stderr wrappers

`create_django_image_object`()

Create a django image file object suitable for FileField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path

Returns django file object

It requires Pillow installed in the environment to work

`create_filer_image_object`()

Create a filer image object suitable for FilerImageField It also sets the following attributes:

- `self.image_name`: the image base name
- `self.filename`: the complete image path
- `self.filer_image`: the filer image object

Returns filer image object

It requires Pillow and django-filer installed in the environment to work

`create_image`(*mode=u'RGB'*, *size=(800*, *600)*)

Create a random image suitable for saving as DjangoFile :param mode: color mode :param size: tuple of width, height :return: image object

It requires Pillow installed in the environment to work

static `create_pages`(*source*, *languages*)

Build pages according to the pages data provided by *`get_pages_data()`* and returns the list of the draft version of each

`create_user`(*username*, *email*, *password*, *is_staff=False*, *is_superuser=False*, *base_cms_permissions=False*, *permissions=None*)

Creates a user with the given properties

Parameters

- **`username`** – Username
- **`email`** – Email
- **`password`** – password
- **`is_staff`** – Staff status

- **is_superuser** – Superuser status
- **base_cms_permissions** – Base django CMS permissions
- **permissions** – Other permissions

Returns User instance

get_page_request(*page*, *user*, *path=None*, *edit=False*, *lang=u'en'*, *use_middlewares=False*, *secure=False*)

Create a GET request for the given page suitable for use the django CMS toolbar

This method requires django CMS installed to work. It will raise an ImportError otherwise; not a big deal as this method makes sense only in a django CMS environment

Parameters

- **page** – current page object
- **user** – current user
- **path** – path (if different from the current page path)
- **edit** – whether enabling editing mode
- **lang** – request language
- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request

Returns request

get_pages()

Create pages using self._pages_data and self.languages :return: list of created pages

get_pages_data()

Construct a list of pages in the different languages available for the project. Default implementation is to return the `_pages_data` attribute

Returns list of pages data

get_plugin_context(*page*, *lang*, *plugin*, *edit=False*)

Returns a context suitable for CMSPlugin.render_plugin / render_placeholder

Parameters

- **page** – Page object
- **lang** – Current language
- **plugin** – Plugin instance
- **edit** – Enable edit mode for rendering

Returns PluginContext instance

get_request(*page*, *lang*, *user=None*, *path=None*, *use_middlewares=False*, *secure=False*)

Create a GET request for the given page and language

Parameters

- **page** – current page object
- **lang** – request language
- **user** – current user
- **path** – path (if different from the current page path)

- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request

Returns request

login_user_context(*user*, *password=None*)
Context manager to make logged in requests :param user: user username :param password: user password (if omitted, username is used)

post_request(*page*, *lang*, *data*, *user=None*, *path=None*, *use_middlewares=False*, *secure=False*)
Create a POST request for the given page and language with CSRF disabled

Parameters

- **page** – current page object
- **lang** – request language
- **data** – POST payload
- **user** – current user
- **path** – path (if different from the current page path)
- **use_middlewares** – pass the request through configured middlewares.
- **secure** – create HTTPS request

Returns request

reload_model(*obj*)
Reload a models instance from database :param obj: model instance to reload :return: the reloaded model instance

render_plugin(*page*, *lang*, *plugin*, *edit=False*)
Renders a single plugin using CMSPlugin.render_plugin

Parameters

- **page** – Page object
- **lang** – Current language
- **plugin** – Plugin instance
- **edit** – Enable edit mode for rendering

Returns Rendered plugin

temp_dir(**args*, ***kwds*)
Context manager to operate on a temporary directory

2.6 Development & community

django CMS Helper is an open-source project.

You don't need to be an expert developer to make a valuable contribution - all you need is a little knowledge, and a willingness to follow the contribution guidelines.

2.6.1 Nephila

django CMS Helper was created by Iacopo Spalletti at Nephila and is released under a GNU GENERAL PUBLIC LICENSE.

Nephila is an active supporter of django CMS and its community. django CMS Helper is intended to help make it easier for developers in the django CMS ecosystem to work effectively and create high quality applications.

Nephila maintains overall control of the django CMS Helper repository.

2.6.2 Standards & policies

django CMS Helper is a django CMS application, and shares much of django CMS's standards and policies.

These include:

- guidelines and policies for contributing to the project, including standards for code and documentation
- standards for managing the project's development
- a code of conduct for community activity

Please familiarise yourself with this documentation if you'd like to contribute to django CMS Helper.

2.7 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

2.7.1 Types of Contributions

Report Bugs

Report bugs at https://github.com/nephila/djangocms-helper/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "feature" is open to whoever wants to implement it.

Write Documentation

djangocms-helper could always use more documentation, whether as part of the official djangocms-helper docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/nephila/djangocms-helper/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

2.7.2 Get Started!

Ready to contribute? Here's how to set up *djangocms-helper* for local development.

1. Fork the *djangocms-helper* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/djangocms-helper.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv djangocms-helper
$ cd djangocms-helper/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ make lint
    $ python setup.py test
$ tox
```

To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

2.7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, and 3.3, and for PyPy. Check https://travis-ci.org/nephila/djangocms-helper/pull_requests and make sure that the tests pass for all supported Python versions.

2.8 History

2.8.1 0.9.4 (XXXX-XX-XX)

- Add Naked setup mode
- Add TEMPLATE_DIRS to special settings
- Add TEMPLATE_LOADERS to special settings
- Allow to specify a locale in makemessages

2.8.2 0.9.3 (2015-10-07)

- Add –no-migrate option to skip migrations
- Add secure argument to generate HTTPS requests
- Better request mocking
- Fix test on django CMS 3.2 (develop)
- Add support for Python 3.5
- Add –persistent option for persistent storage

2.8.3 0.9.2 (2015-09-14)

- Add support for apphooks and parent pages in BaseTestCase.create_pages
- If pages contains apphook, urlconf is reloaded automatically
- Improve documentation
- Add support for top-positioned MIDDLEWARE_CLASSES
- Code cleanup

2.8.4 0.9.1 (2015-08-30)

- Better support for aldryn-boilerplates

2.8.5 0.9.0 (2015-08-20)

- Complete support for Django 1.8 / django CMS develop
- Support for aldryn-boilerplates settings
- Migrations are now enabled by default during tests
- Minor BaseTestCase refactoring
- Remove support for Django 1.5
- Fix treebeard support
- Minor fixes
- Adds login_user_context method to BaseTestCase

2.8.6 0.8.1 (2015-05-31)

- Add basic support for Django 1.8 / django CMS develop
- Code cleanups
- Smarter migration layout detection

2.8.7 0.8.0 (2015-03-22)

- Add –native option to use native test command instead of djangocms-helper one
- Use django-discover-runner on Django 1.5 if present
- Better handling of runner options
- Add support for empty/dry-run arguments to makemigrations
- Add USE_CMS flag to settings when using django CMS configuration

2.8.8 0.7.0 (2015-01-22)

- Fix an error which prevents the runner to discover the settings
- django CMS is no more a dependency, install it manually to enable django CMS support

2.8.9 0.6.0 (2015-01-10)

- Add a runner to make cms_helper file itself a runner for djangocms-helper
- Fix issues with mptt / treebeard and Django 1.7
- Fix some makemigrations / –migrate issues
- Make djangocms-helper less django CMS dependent

2.8.10 0.5.0 (2015-01-01)

- Fixing bugs when using extra settings
- Add messages framework to default environment
- Add CSRF middleware / context_processor to default settings
- Add base helper class for test cases
- Complete Django 1.7 support
- Smarter detection of migration operations in Django 1.6-
- Add option to create migrations for external applications

2.8.11 0.4.0 (2014-09-18)

- Add support for command line test runner options;
- Add check command on Django 1.7+;
- Add cms check command (which triggers cms inclusion);
- Add squashmigration command Django 1.7+;
- Add support for makemigrations merge on Django 1.7+;
- Add helpers for custom user models;

2.8.12 0.3.1 (2014-08-25)

- Add staticfiles application;
- Add djangocms_admin_style if cms is enabled;

2.8.13 0.3.0 (2014-08-14)

- Add support for django nose test runner;
- Add default CMS template;

2.8.14 0.2.0 (2014-08-12)

- Add option to customize sample project settings;
- Add option to exclude djanigo CMS from test project configurations;
- Add support for Django 1.7;

2.8.15 0.1.0 (2014-08-09)

- First public release.

d

B

C

D

G

L

P

R

T