

---

# **django-nested-admin Documentation**

*Release 3.0.0*

**The Atlantic**

**Mar 09, 2017**



---

# Contents

---

<b>1</b>	<b>Compatibility</b>	<b>3</b>
1.1	Contents . . . . .	3
1.2	License . . . . .	7
1.3	Indices and tables . . . . .	7



django-nested-admin is a project that makes it possible to nest admin inlines (that is, to define inlines on InlineModelAdmin classes). It also allows these inlines to be sorted by drag-and-drop, and has optional Grappelli support.



django-nested-admin is compatible with Django 1.8+ and Python versions 2.7 and 3.4+. It is also compatible with Grappelli and Django Suit. It does not require Grappelli or Django Suit for the drag-and-drop sorting features.

## Contents

### Quick start guide

Django (version 1.8+) needs to be installed to use django-nested-admin.

### Installation

```
pip install django-nested-admin
```

Go to [GitHub](#) if you need to download or install from source.

### Setup

Open `settings.py` and add `nested_admin` to your `INSTALLED_APPS`

```
INSTALLED_APPS = (  
    # ...  
    'nested_admin',  
)
```

Add URL-patterns:

```
urlpatterns = patterns('',  
    # ...  
    url(r'^_nested_admin/', include('nested_admin.urls')),  
)
```

Collect the static files:

```
$ python manage.py collectstatic
```

## Example Usage

In order to use `django-nested-admin`, use the following classes in place of their `django` admin equivalents:

<code>django.contrib.admin</code>	<code>nested_admin</code>
<code>ModelAdmin</code>	<code>NestedModelAdmin</code>
<code>InlineModelAdmin</code>	<code>NestedInlineModelAdmin</code>
<code>StackedInline</code>	<code>NestedStackedInline</code>
<code>TabularInline</code>	<code>NestedTabularInline</code>

There is also `nested_admin.NestedGenericStackedInline` and `nested_admin.NestedGenericTabularInline` which are the nesting-capable versions of `GenericStackedInline` and `GenericTabularInline` in `django.contrib.contenttypes.admin`.

```
# An example admin.py for a Table of Contents app

from django.contrib import admin
import nested_admin

from .models import TableOfContents, TocArticle, TocSection

class TocArticleInline(nested_admin.NestedStackedInline):
    model = TocArticle
    sortable_field_name = "position"

class TocSectionInline(nested_admin.NestedStackedInline):
    model = TocSection
    sortable_field_name = "position"
    inlines = [TocArticleInline]

class TableOfContentsAdmin(nested_admin.NestedModelAdmin):
    inlines = [TocSectionInline]

admin.site.register(TableOfContents, TableOfContentsAdmin)
```

## Why use `django-nested-admin`?

	<code>django-nested-admin</code>	<code>django-super-inlines</code>	<code>django-nested-inlines</code>	<code>dj-nested-inlines</code>	<code>django-nested-inline</code>
Nested inlines	✓	✓	✓	✓	✓
Grappelli support	✓	✓			
Generic inlines support	✓	✓			
Drag-and-drop sorting	✓				
Selenium Tests	✓				
Django Suit support	✓				



Above, you will find a feature comparison of all other implementations of nested inlines in the django admin of which I am aware. [django-nested-inline](#), [django-nested-inlines](#), and [dj-nested-inlines](#) are all variations upon a [patch](#) posted to Django [ticket #9025](#). [django-nested-inlines](#) has had no commits since 2013 and appears to be abandoned, with [dj-nested-inlines](#) being a better maintained fork. It, along with [django-nested-inline](#), are maintained insofar as the maintainers merge in pull requests and update PyPI, but they do not appear to be working on improving these implementations themselves. [django-super-inlines](#) is a fresher take on the problem, and looks to be more promising.

The largest functional difference between [django-nested-admin](#) and these other projects is its support for drag-and-drop sorting functionality within and between inlines, similar to Grappelli's sortable inline feature (though it does not require Grappelli). It also has fairly extensive test coverage. And lastly [django-nested-admin](#) is and will remain an integral part of the CMS that powers [TheAtlantic.com](#) (and [CityLab.com](#)) where it is used by editors to curate the homepages, email newsletters, and landing pages for special projects and site sections (the way we used [django-nested-admin](#) to power the curation of [TheAtlantic.com](#)'s homepage was the subject of a [talk we gave at the 2015 Djangocon](#)).

## Customization

### Sortable

The steps for enabling drag-and-drop sorting functionality is identical to Grappelli's (even with the vanilla Django admin), so the [Grappelli documentation](#) for this feature is the best reference. This is equally true of sortable in Django Suit—do not follow Django Suit's instructions for sortable, as they will not work. The [other features of Grappelli](#) have not been ported to work with vanilla Django, but in principle they should all still work with nested inlines using [django-nested-admin](#) if Grappelli is installed. If you run into any difficulty with this, please [create an issue](#) on this project's Github.

### Events

Adding or removing an inline fires the `formset:added` and `formset:removed` events, respectively, that were [added to Django in 1.9 \(#15760\)](#). This is particularly useful for when you need to execute javascript upon an inline being added (for instance, to initialize a custom widget). The API is the same as Django 1.9, so reference the [official Django docs](#) for information about how to use these events. In addition, [django-nested-admin](#) also emits events for when the delete checkbox (or button, in Grappelli) is toggled with the `formset:deleted` and `formset:undeleted` events. Less likely to be used, but still available, are the following events:

**`djnesting:mutate`** Fired whenever the state of a formset changes

**`djnesting:initialized`** Fired after an inline has been initialized, either on load or after being added

**`djnesting::attrchanged`** Fired whenever a drag-and-drop operation or removal necessitates changing the `name`, `id`, and `for` attributes of the elements within a form or formset. Ideally custom Django widgets with javascript hooks should be written so that they do not need to be reinitialized if a form field's `id` or `name` changes, but in cases where they do, use this event to refresh or reinitialize the bindings.

## Contributing

### Editing javascript and css

This project uses [gulp](#), [babel](#), [browserify](#), and [scss](#) for building its javascript and css. To install the dependencies for the build process, run `npm install` from the root of the repository. You can then run `gulp` to rebuild the static files, or `gulp watch` when actively editing these files to detect changes and rebuild automatically.

### Running tests

django-nested-admin has fairly extensive test coverage. The best way to run the tests is with `tox`, which runs the tests against all supported Django installs. To run the tests within a virtualenv run `python runtests.py` from the repository directory. The tests require a selenium webdriver to be installed. By default the tests run with `phantomjs`, but it is also possible to run the tests with the `chrome` webdriver by passing `--selenium=chrome` to `runtests.py` or, if running with `tox`, running `tox -- --selenium=chrome`. See `runtests.py --help` for a complete list of the options available.

Pull requests are automatically run through Travis CI upon submission to verify that the changes do not introduce regressions.

### Writing tests

The tests for this project are patterned off of the system that the Django project itself uses for tests (including the changes that are being made in 1.10), so [Django's guide on writing unit tests](#) is a good place to start.

Selenium tests are difficult to write, particularly for complicated user interactions such as drag-and-drop. To assist in writing tests for the functionality of nested inlines, a base `TestCase` with methods for executing the possible user interactions with inlines (e.g. adding inlines, removing inlines, setting field values, drag-and-drop re-ordering) is provided with `base.BaseNestedAdminTestCase`.

Depending on what you are trying to test, it might make sense to add the test to one of the existing “test apps” in django-nested-admin. These include:

**admin\_widgets** Integration tests for built-in Django widgets that use javascript, e.g. `prepopulated_fields` or `ManyToManyFields` with `filter_horizontal`.

**gfk** Tests for `nested_admin.NestedGenericStackedInline` and `nested_admin.NestedGenericTabularInline`

**one\_deep** Tests that compare screenshots of standard, non-nested inlines using the standard inline classes and their nested equivalents (but without any actual nesting), to ensure that the custom templates, css, and javascript used by django-nested-admin looks the same as their non-nested Django or Grappelli equivalents.

**two\_deep** Tests for `nested_admin.NestedStackedInline` and `nested_admin.NestedTabularInline` where there is only one “nested” inline (using two levels of inlines, hence “two deep”).

**three\_deep** Tests the same things as `two_deep`, except with inlines nested “three deep.”

If your test requires the creation of new test models, then it may make sense to write a new test app. Create a new subdirectory under `nested_admin/tests` with an `__init__.py`, `admin.py`, `models.py`, and `tests.py`. The test runner will automatically add this to the list of `INSTALLED_APPS` and execute the tests defined in `tests.py`. The tests in `two_deep` are the most complete, and can serve as a reference for how to use the helper methods for simulating user interactions.

## Changelog

### 3.0.2 (April 17, 2016)

- Feature: django-suit support
- Fixed: javascript syntax error
- Fixed: bug where tabular inlines without sortables could not be updated

### 3.0.0 (April 13, 2016)

- Added documentation
- Fixed visual discrepancies between the appearance of nested and their usual appearance in Django and Grappelli. Added screenshot comparison tests to prevent future regressions.
- Support nesting of generic inlines (fixes #30)
- Support for `show_change_link` (fixes #22)
- Support for Django 1.10dev
- Dropped support for version of Django prior to 1.8, which greatly simplified the Python code.
- Use gulp for building static assets, rewritten with scss and ES6

## API reference

### `nested_admin` Package

`nested_admin.tests` Module

## License

The django code is licensed under the [Simplified BSD License](#). View the `LICENSE` file under the root directory for complete license and copyright information.

## Indices and tables

- [genindex](#)
- [modindex](#)
- [search](#)



## N

nested\_admin.tests (module), 7