Django Admin Sortable Documentation

Release 1.7.0

Brandon Taylor

November 19, 2014

Contents

Django Admin Sortable is a super-easy way to add drag-and-drop ordering to almost any model you manage through Django admin. Inlines for a sortable model may also be made sortable, enabling individual items or groups of items to be sortable.

CHAPTER 1

Supported Django Versions

1.1 Django 1.4.x

Use django-admin-sortable 1.4.9 or below.

Note: v1.5.2 introduced backwards incompatible changes for Django 1.4.x

1.2 Django >= 1.5.x

Use the latest version of django-admin-sortable.

Warning: v1.6.6 introduced a backwards-incompatible change for `sorting_filters`. Please update your `sorting_filters` attribute(s) to the new, tuple-based format.

CHAPTER 2

What's New in 1.7.0?

- Python 2.6 backwards compatibility. Thanks @EnTeQuAk

CHAPTER 3

Contents:

3.1 Quickstart

To get started using `django-admin-sortable` simply install it using `pip`:

```
$ pip install django-admin-sortable
```

Add `adminsortable` to your project's `INSTALLED_APPS` setting.

Ensure `django.core.context_processors.static` is in your `TEMPLATE_CONTEXT_PROCESSORS` setting.

Define your model, inheriting from `adminsortable.Sortable`:

```
# models.py
from adminsortable.models import Sortable

class MySortableClass(Sortable):
    class Meta(Sortable.Meta):
        pass

    title = models.CharField(max_length=50)

    def __unicode__(self):
        return self.title
```

Wire up your sortable model to Django admin:

```
# admin.py
from adminsortable.admin import SortableAdmin
from .models import MySortableClass

class MySortableAdminClass(SortableAdmin):
    """Any admin options you need go here"""

admin.site.register(MySortableClass, MySortableAdminClass)
```

Your model's ChangeList view should now have an extra tool link when there are 2 or more objects present that will take you to a view where you can drag-and-drop the objects into your desired order.

3.2 Configuring Django Admin Sortable

Configuring django-admin-sortable is quite simple:

1. Add `adminsortable` to your `INSTALLED_APPS`.
2. Ensure `django.core.context_processors.static` is in your `TEMPLATE_CONTEXT_PROCESSORS`.

3.2.1 Static Media

django-admin-sortable includes a few CSS and JavaScript files. The preferred method of getting these files into your project is to use the staticfiles app.

Alternatively, you can copy or symlink the `adminsortable` folder inside the `static` directory to the location you serve static files from.

3.3 Using Django Admin Sortable

3.3.1 Models

To add sorting to a model, your model needs to inherit from `Sortable` and have an inner `Meta` class that inherits from `Sortable.Meta`:

```
# models.py
from adminsortable.models import Sortable

class MySortableClass(Sortable):
    class Meta(Sortable.Meta):
        pass

    title = models.CharField(max_length=50)

    def __unicode__(self):
        return self.title
```

It is also possible to order objects relative to another object that is a ForeignKey.

Note: A small caveat here is that `Category` must also either inherit from `Sortable` or include an `order` property which is a `PositiveSmallInteger` field. This is due to the way Django admin instantiates classes.

```
# models.py
from adminsortable.fields import SortableForeignKey

class Category(Sortable):
    class Meta(Sortable.Meta):
        pass

    title = models.CharField(max_length=50)
    ...

class Project(Sortable):
    class Meta(Sortable.Meta):
        pass
```

```
    category = SortableForeignKey(Category)
    title = models.CharField(max_length=50)

    def __unicode__(self):
        return self.title
```

`Sortable` has one field: `order` and adds a default ordering value set to `order`, ascending.

Adding Sortable to an existing model

If you're adding Sorting to an existing model, it is recommended that you use django-south to create a schema migration to add the "order" field to your model. You will also need to create a data migration in order to add the appropriate values for the `order` column.

Example assuming a model named "Category":

```
def forwards(self, orm):
    for index, category in enumerate(orm.Category.objects.all()):
        category.order = index + 1
        category.save()
```

See this link for more information on Data Migrations.

3.3.2 Django Admin

To enable sorting in the admin, you need to inherit from `SortableAdmin`:

```
from django.contrib import admin
from myapp.models import MySortableClass
from adminsortable.admin import SortableAdmin

class MySortableAdminClass(SortableAdmin):
    """Any admin options you need go here"""

admin.site.register(MySortableClass, MySortableAdminClass)
```

To enable sorting on TabularInline models, you need to inherit from SortableTabularInline:

```
from adminsortable.admin import SortableTabularInline

class MySortableTabularInline(SortableTabularInline):
    """Your inline options go here"""
```

To enable sorting on StackedInline models, you need to inherit from SortableStackedInline:

```
from adminsortable.admin import SortableStackedInline

class MySortableStackedInline(SortableStackedInline):
    """Your inline options go here"""
```

There are also generic equivalents that you can inherit from:

```
from adminsortable.admin import (SortableGenericTabularInline,
    SortableGenericStackedInline)
    """Your generic inline options go here"""
```

Overriding `queryset()`

django-admin-sortable supports custom queryset overrides on admin models and inline models in Django admin!

If you're providing an override of a `SortableAdmin` or `Sortable` inline model, you don't need to do anything extra. django-admin-sortable will automatically honor your queryset.

Have a look at the `WidgetAdmin` class in the sample project for an example of an admin class with a custom `queryset()` override.

Overriding `queryset()` for an inline model

This is a special case, which requires a few lines of extra code to properly determine the sortability of your model. Example:

```
# add this import to your admin.py
from adminsortable.utils import get_is_sortable


class ComponentInline(SortableStackedInline):
    model = Component

    def queryset(self, request):
        qs = super(ComponentInline, self).queryset(request).filter(
            title__icontains='foo')

        # You'll need to add these lines to determine if your model
        # is sortable once we hit the change_form() for the parent model.

        if get_is_sortable(qs):
            self.model.is_sortable = True
        else:
            self.model.is_sortable = False
        return qs
```

If you override the queryset of an inline, the number of objects present may change, and adminsortable won't be able to automatically determine if the inline model is sortable from here, which is why we have to set the `is_sortable` property of the model in this method.

Sorting subsets of objects

It is also possible to sort a subset of objects in your model by adding a `sorting_filters` tuple. This works exactly the same as `.filter()` on a QuerySet, and is applied *after* `get_queryset()` on the admin class, allowing you to override the queryset as you would normally in admin but apply additional filters for sorting. The text "Change Order of" will appear before each filter in the Change List template, and the filter groups are displayed from left to right in the order listed. If no `sorting_filters` are specified, the text "Change Order" will be displayed for the link.

An example of sorting subsets would be a "Board of Directors". In this use case, you have a list of "People" objects. Some of these people are on the Board of Directors and some not, and you need to sort them independently:

```
class Person(Sortable):
    class Meta(Sortable.Meta):
        verbose_name_plural = 'People'

    first_name = models.CharField(max_length=50)
    last_name = models.CharField(max_length=50)
    is_board_member = models.BooleanField('Board Member', default=False)
```

```
    sorting_filters = (
        ('Board Members', {'is_board_member': True}),
        ('Non-Board Members', {'is_board_member': False}),
    )

    def __unicode__(self):
        return '{} {}'.format(self.first_name, self.last_name)
```

Warning: django-admin-sortable 1.6.6 introduces a backwards-incompatible change for `sorting_filters`. Previously this attribute was defined as a dictionary, so you'll need to change your values over to the new tuple-based format.

Extending custom templates

By default, adminsortable's change form and change list views inherit from Django admin's standard templates. Sometimes you need to have a custom change form or change list, but also need adminsortable's CSS and JavaScript for inline models that are sortable for example.

`SortableAdmin` has two attributes you can override for this use case:

```
change_form_template_extends
change_list_template_extends
```

These attributes have default values of:

```
change_form_template_extends = 'admin/change_form.html'
change_list_template_extends = 'admin/change_list.html'
```

If you need to extend the inline change form templates, you'll need to select the right one, depending on your version of Django. For Django 1.5.x or below, you'll need to extend one of the following:

```
templates/adminsortable/edit_inline/stacked-1.5.x.html
templates/adminsortable/edit_inline/tabular-inline-1.5.x.html
```

For Django >= 1.6.x, extend:

```
templates/adminsortable/edit_inline/stacked.html
templates/adminsortable/edit_inline/tabular.html
```

Note: A Special Note About Stacked Inlines... The height of a stacked inline model can dynamically increase, which can make them difficult to sort. If you anticipate the height of a stacked inline is going to be very tall, I would suggest using TabularStackedInline instead.

3.4 Django-CMS Integration

Django-CMS plugins use their own change form, and thus won't automatically include the necessary JavaScript for django-admin-sortable to work. Fortunately, this is easy to resolve, as the `CMSPlugin` class allows a change form template to be specified:

```
# example plugin
from cms.plugin_base import CMSPluginBase

class CMSCarouselPlugin(CMSPluginBase):
```

```
    admin_preview = False
    change_form_template = 'cms/sortable-stacked-inline-change-form.html'
    inlines = [SlideInline]
    model = Carousel
    name = _('Carousel')
    render_template = 'carousels/carousel.html'

    def render(self, context, instance, placeholder):
        context.update({
            'carousel': instance,
            'placeholder': placeholder
        })
        return context

plugin_pool.register_plugin(CMSCarouselPlugin)
```

The contents of sortable-stacked-inline-change-form.html at a minimum need to extend the extrahead block with:

```
{% extends "admin/cms/page/plugin_change_form.html" %}
{% load static from staticfiles %}

{% block extrahead %}
    {{ block.super }}
    <script type="text/javascript" src="{% static 'adminsortable/js/jquery-ui-django-admin.min.js' %
    <script type="text/javascript" src="{% static 'adminsortable/js/jquery.django-csrf.js' %}"></scri
    <script type="text/javascript" src="{% static 'adminsortable/js/admin.sortable.stacked.inlines.js

    <link rel="stylesheet" type="text/css" href="{% static 'adminsortable/css/admin.sortable.inline.c
{% endblock extrahead %}
```

Sorting within Django-CMS is really only feasible for inline models of a plugin as Django-CMS already includes sorting for plugin instances. For tabular inlines, just substitute:

```
<script type="text/javascript" src="{% static 'adminsortable/js/admin.sortable.stacked.inlines.js' %
```

with:

```
<script type="text/javascript" src="{% static 'adminsortable/js/admin.sortable.tabular.inlines.js' %
```

3.5 Known Issue(s)

Because of the way inline models are added to their parent model in the change form, it is not currently possible to have sortable inline models whose parent does not inhert from `Sortable`.

3.6 Testing

Have a look at the included `/sample_project` directory to see a working project. The login credentials for admin are: admin/admin

When a model is sortable, a tool-area link will be added that says "Change Order". Click this link, and you will be taken to the custom view where you can drag-and-drop the records into order.

Inlines may be drag-and-dropped into any order directly from the change form.

Unit and functional tests may be found in the `app/tests.py` file and run via:

```
$ python manage.py test app
```

3.7 Rationale

3.7.1 Why another drag-and-drop ordering plugin?

Other projects have added drag-and-drop ordering to the ChangeList view, however this introduces a couple of problems...

- The ChangeList view supports pagination, which makes drag-and-drop ordering across pages impossible.
- The ChangeList view by default, does not order records based on a foreign key, nor distinguish between rows that are associated with a foreign key. This makes ordering the records grouped by a foreign key impossible.
- The ChangeList supports in-line editing, and adding drag-and-drop ordering on top of that just seemed a little much in my opinion.

3.8 Status

django-admin-sortable is stable and currently used in production.

3.9 Future Plans

- Support for foreign keys that are self referential
- Move unit tests out of sample project (I could really use some help with this one)
- Travis CI integration

3.10 License

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*