Django Extra Views Documentation

Release 0.5.4

Andrew Ingram

October 07, 2015

Contents

1 Installation **3**

2 Contents **5**

2.1 Views 5

3 Indices and tables **9**

Django Extra Views provides a number of additional class-based generic views to complement those provide by Django itself.

CHAPTER 1

Installation

Installing from pypi (using pip).

```
pip install django-extra-views
```

Installing from github.

```
pip install -e git://github.com/AndrewIngram/django-extra-views.git#egg=django-extra-views
```

CHAPTER 2

Contents

2.1 Views

For all of these views we've tried to mimic the API of Django's existing class-based views as closely as possible, so they should feel natural to anyone who's already familiar with Django's views.

2.1.1 FormSetView

This is the formset equivalent of Django's FormView. Use it when you want to display a single (non-model) formset on a page.

A simple formset:

```
from extra_views import FormSetView
from foo.forms import MyForm


class MyFormSetView(FormSetView):
    template_name = 'myformset.html'
    form_class = MyForm
    success_url = 'success/'

    def get_initial(self):
        # return whatever you'd normally use as the initial data for your formset.
      return data

    def formset_valid(self, formset):
        # do stuff
        return super(MyFormSetView, self).formset_valid(formset)
```

This view will render the template *myformset.html* with a context variable *formset* representing the formset of MyForm. Once POSTed and successfully validated, *formset_valid* will be called which is where your handling logic goes, then it redirects to *success_url*.

FormSetView exposes all the parameters you'd normally be able to pass to formset_factory. Example (using the default settings):

```
class MyFormSetView(FormSetView):
    template_name = 'myformset.html'
    form_class = MyForm
    success_url = 'success/'
    extra = 2
```

```
    max_num = None
    can_order = False
    can_delete = False

    ...
```

2.1.2 ModelFormSetView

ModelFormSetView makes use of Django's modelformset_factory, using the declarative syntax used in FormSetView as well as Django's own class-based views. So as you'd expect, the simplest usage is as follows:

```
from extra_views import ModelFormSetView
from foo.models import MyModel


class MyModelFormSetView(ModelFormSetView):
    template_name = 'mymodelformset.html'
    model = MyModel
```

Like *FormSetView*, the *formset* variable is made available in the template context. By default this will populate the formset with all the instances of *MyModel* in the database. You can control this by overriding *get_queryset* on the class, which could filter on a URL kwarg (*self.kwargs*), for example:

```
class MyModelFormSetView(ModelFormSetView):
    template_name = 'mymodelformset.html'
    model = MyModel

    def get_queryset(self):
        slug = self.kwargs['slug']
        return super(MyModelFormSetView, self).get_queryset().filter(slug=slug)
```

2.1.3 InlineFormSetView

When you want to edit models related to a parent model (using a ForeignKey), you'll want to use InlineFormSetView. An example use case would be editing user reviews related to a product:

```
from extra_views import InlineFormSetView


class EditProductReviewsView(InlineFormSetView):
    model = Product
    inline_model = Review

    ...
```

Aside from the use of *model* and *inline_model*, InlineFormSetView works more-or-less in the same way as ModelFormSetView.

2.1.4 GenericInlineFormSetView

You can also use generic relationships for your inline formsets, this makes use of Django's *generic_inlineformset_factory*. The usage is the same, but with the addition of *ct_field* and *ct_fk_field*:

```
from extra_views.generic import GenericInlineFormSetView


class EditProductReviewsView(GenericInlineFormSetView):
    model = Product
    inline_model = Review
    ct_field = "content_type"
    ct_fk_field = "object_id"
    max_num = 1

    ...
```

2.1.5 CreateWithInlinesView and UpdateWithInlinesView

These are the most powerful views in the library, they are effectively replacements for Django's own CreateView and UpdateView. The key difference is that they let you include any number of inline formsets (as well as the parent model's form), this provides functionality much like the Django Admin change forms. The API should be fairly familiar as well. The list of the inlines will be passed to the template as context variable *inlines*.

Here is a simple example that demonstrates the use of each view with both normal inline relationships and generic inlines:

```
from extra_views import InlineFormSet, CreateWithInlinesView, UpdateWithInlinesView,
from extra_views.generic import GenericInlineFormSet


class ItemsInline(InlineFormSet):
    model = Item


class TagsInline(GenericInlineFormSet):
    model = Tag


class OrderCreateView(CreateWithInlinesView):
    model = Order
    inlines = [ItemsInline, TagsInline]

    def get_success_url(self):
        return self.object.get_absolute_url()


class OrderUpdateView(UpdateWithInlinesView):
    model = Order
    form_class = OrderForm
    inlines = [ItemsInline, TagsInline]

    def get_success_url(self):
        return self.object.get_absolute_url()
```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search