django-sitemapper Documentation

Release 1.0.0

Mike Hurt

October 09, 2014

Contents

Sitemapper is a Django app to manage sitemap.xml overrides on a per-object basis.

Migrations are provided for both Django 1.7+ and earlier versions using South.

CHAPTER 1

Getting started

The first thing you'll need to do is check out the *installation guide and requirements*.

If you're familiar with installing Django apps then the installation is totally standard, with no additional dependencies.

CHAPTER 2

Contribute

- Issue Tracker: https://bitbucket.org/mhurt/django-sitemapper/issues
- Source Code: https://bitbucket.org/mhurt/django-sitemapper/

CHAPTER 3

License

The project is licensed under the MIT license.

3.1 Installation

This part covers installing django-sitemapper and configuring your Django project to use it.

3.1.1 Requirements

Sitemapper currently requires Django 1.4.2 or greater and Python 2.7 or greater.

South migrations are provided for Django versions prior to 1.7, so if you'd like to use these please make sure you have South installed before continuing.

3.1.2 Get the code

Installing Sitemapper is simple with pip (or, if you must, with easy_install), just run this in your terminal:

```
$ pip install django-sitemapper
or
$ easy_install django-sitemapper
```

Sitemapper is actively developed on Bitbucket, where you can grab the latest code.

Either clone the repository:

```
$ hg clone https://bitbucket.org/mhurt/django-sitemapper
```

or download and unpack the tar-ball or zip-ball of your choice.

Once you have a copy of the source, you can install it into your site packages easily:

```
$ cd django-sitemapper
$ python setup.py install
```

3.1.3 Configuring your Django project

Add `"sitemapper"` to your `INSTALLED_APPS` setting like this:

```
INSTALLED_APPS = {
  ...
  'sitemapper'
}
```

- For Django 1.7 users, run `python manage.py migrate` to create the

models.

- If you're using South, please see *Using Sitemapper with South*.
- Otherwise simply run `python manage.py syncdb`.

3.1.4 Using Sitemapper with South

If you're using Django 1.7 you won't need to use South as migrations are built in.

If you're using an earlier version of Django with South 1.0 the provided south_migrations will be automatically detected.

For earlier versions of South you'll need to tell explicitly define which migrations to use by adding to, or creating, the `SOUTH_MIGRATION_MODULES` in your settings file:

```
# settings.py
...
SOUTH_MIGRATION_MODULES = {
    'sitemapper': 'sitemapper.south_migrations',
}
```

Don't worry, though, as running running a `migrate` will complain loudly if you've forgotten this step.

3.2 Setting up a sitemap

This section details all of the steps from a minimum working example, through to a fully configured Sitemapper sitemap. In general this process is very simple, and mirrors Django's own sitemaps, with only minor differences which are explained below.

3.2.1 Getting Started

Create a `sitemaps.py` file within your app directory and add the following (replacing MyModel with whatever you called yours):

```
# project/myapp/sitemaps.py
from sitemapper.sitemaps import Sitemap
from .models import MyModel


class MyModelSitemap(Sitemap):

    # You'll need a queryset...
    queryset = MyModel.objects.all()
```

From this point onwards you can use `MyModelSitemap` as you would any other sitemaps instance. In your root `urls.py` set up the sitemap as you would normally:

```
# project/urls.py
...
...
from django.contrib.sitemaps.views import sitemap
from myapp.sitemaps import MyModelSitemap

sitemaps = {
    'mymodel': MyModelSitemap,
    }

urlpatterns = [
    # Your other patterns here
    ...
    ...

    url(r'^sitemap\.xml', sitemap, {'sitemaps': sitemaps},
        name='django.sitemaps.views.sitemap')

    ]
```

As you can see, the URL configuration is exactly the same as for Django's built-in sitemaps. If you need more information on those see Django's sitemaps documentation for lots more details.

Note: For this simplistic configuration the output of `/sitemaps.xml` will contain the `<loc>` URL for each object in your queryset.

It will only show `<changefreq>` and `<priority>` if these have been assigned via a SitemapEntry.

The `<lastmod>` field will not be displayed since we haven't, yet, defined how to find it.

3.2.2 Setting Defaults

So, lets make our sitemap a little more informative. We'll keep using our basic urlconf for the moment:

```
# project/myapp/sitemaps.py
from sitemapper.sitemaps import Sitemap
from .models import MyModel


class MyModelSitemap(Sitemap):

    # You'll need a queryset...
    queryset = MyModel.objects.all()

    # Assign some sensible defaults...
    default_changefreq = 'weekly'
    default_priority = 0.5
```

Here we've defined two new attributes: `default_changefreq` and `default_priority`. Like Django's own Sitemap class these, as you might guess, allow you define the default values to use. But, unlike Django's Sitemap class these will be overridden *if* a model instance has a `changefreq` or `priority` value assigned via a SitemapEntry.

Caution: Make sure you don't accidentally override the `changefreq` or `priority` attributes, as doing so will prevent that data being picked up from the SitemapEntry.

3.2.3 Getting the Timestamp

The final stage of fully configuring our sitemap is to get the last-modfied date or time:

```
# project/myapp/sitemaps.py
from sitemapper.sitemaps import Sitemap
from .models import MyModel


class MyModelSitemap(Sitemap):

    # You'll need a queryset...
    queryset = MyModel.objects.all()

    # Assign some sensible defaults...
    default_changefreq = 'weekly'
    default_priority = 0.5

    # Get the date-/time-stamp
    def lastmod(self, item):
      return item.lastmodified
```

3.3 API

3.3.1 sitemapper.sitemaps

class sitemapper.sitemaps.**SitemapBase**

Bases: django.contrib.sitemaps.Sitemap

The root class for sitemapper.Sitemaps.

Inherits from django.contrib.sitemaps.Sitemap and overrides the Sitemap.items() method.

In this class calling items() does two things:

1. It creates a private attribute _entries containing a dictionary of sitemapper.SitemapEntry objects having the same ContentType as the supplied queryset's model. This dictionary is keyed to the object_id of each SitemapEntry instance.
2. It then returns the queryset.

Attributes:

queryset (queryset): A queryset containing the objects to appear in the sitemap.

_entries (dict): A private variable populated by the items() method and containing SitemapEntries matching the ContentType and object_id of the queryset above.

items()

Assign the result of _get_entries_for_model(model) to the private attribute _entries, and return the queryset attribute.

class sitemapper.sitemaps.**Sitemap**

Bases: sitemapper.sitemaps.SitemapBase

changefreq(*item*)

Return one of three values:

1. the SitemapEntry.changefreq related to item, if set;

2.or, the default_changefreq, if set;

3.or, None

Args: item (model instance): An member instance of self.queryset.

priority(*item*)
Return one of three values:

1.the SitemapEntry.priority related to `item`, if set;

2.or, the default_priority, if set;

3.or, None

Args: item (model instance): An member instance of self.queryset.

3.3.2 sitemapper.models

class `sitemapper.models.`**SitemapEntry**(**args*, ***kwargs*)
SitemapEntry(id, content_type_id, object_id, changefreq, priority)

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*

S

C

I

P

S