django-uuidstore Documentation

Release 0.0.1

Mike Hurt

October 16, 2014

Contents

1 Installation **3**
1.1 Configuration 3
1.2 South Migrations 3

2 Getting Started **5**
2.1 Registering your model 5
2.2 Monkey patching 5
2.3 Denormalisation 6
2.4 Denormalisation to a UUID field 7

3 Indices and tables **9**

Contents:

CHAPTER 1

Installation

Use your favorite Python installer to install it from PyPI:

```
$ pip install django-uuidstore
```

If you are using `pip` version 1.4 or later you'll need to explicitly allow pre-release installation:

```
$ pip install --pre django-uuidstore
```

Or get the source from the application site:

```
$ hg clone https://bitbucket.org/mhurt/django-uuidstore
$ cd django-uuidstore
$ python setup.py install
```

1.1 Configuration

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

```
INSTALLED_APPS = {
  ...
  'uuidstore'
}
```

- For Django 1.7 users, run `python manage.py migrate uuidstore` to create the models.
- If you're using South, please see *South Migrations*.
- Otherwise simply run `python manage.py syncdb`.

1.2 South Migrations

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 = {
    'uuidstore': 'uuidstore.south_migrations',
}
```

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

CHAPTER 2

Getting Started

2.1 Registering your model

Somewhere at the end of your models.py add the following:

```
# models.py
from django.db import models


class MyModel(models.Model):
    title = models.Charfield(max_length=60)


import uuidstore.registry
uuidstore.registry.register(MyModel)
```

With the code above `MyModel` is registered with *uuidstore*. When you save an instance of your model for the first time, and new ObjectUUID will be created containing a UUID and a generic-relation to your model instance.

Let's see how that works by running `python manage.py shell`:

```
>>> from myapp.models import MyModel
>>> from uuidstore.models import ObjectUUID
>>>
>>> # Create an instance of your model
>>> instance = MyModel.objects.create(title='Foo')
>>>
>>> # Retrieve the related ObjectUUID
>>> stored = ObjectUUID.objects.get_for_instance(instance)
>>> stored.uuid
u'472eca67-3726-4fbf-8e94-5a48849b3e0c'
```

In this case we can see that the ObjectUUID has been created. The UUID itself is only accessible by explicitly querying ObjectUUID. For most uses you'll probably want the UUID accessible via the model instance itself. The following examples show how this is achieved.

2.2 Monkey patching

This example shows how to both register your model *and* have the UUID accessible as an attribute of your model.

```
# models.py
from django.db import models


class MyModel(models.Model):
    title = models.Charfield(max_length=60)

import uuidstore.registry
uuidstore.registry.register(MyModel, uuid_descriptor='uuid')
```

The only thing we've changed with this code is to add a keyword argument `uuid_descriptor` to the registration call. This works as the previous example, but this time the next instantiation of a MyModel object will look up its UUID and attach it as a property of MyModel with the name you supplied.

Let's see how that works by running `python manage.py shell`:

```
>>> from myapp.models import MyModel
>>>
>>> # Create an instance of your model
>>> instance = MyModel.objects.create(title='Bar')
>>>
>>> # Unlike the first example, the UUID is now attached to your instance:
>>> instance.uuid
u'5c8ddcab-05e6-49c1-8f0d-4530dec8edb9'
```

This works well for cases where you are unable to modify the base model itself, but it's not very efficient as we're invoking a ContentType lookup for each instance.

2.3 Denormalisation

Here we'll cut out some of the inefficiency of monkey patching by denormalising the UUID to a CharField.

```
# models.py
from django.db import models


class MyModel(models.Model):
    title = models.Charfield(max_length=60)
    uuid = models.CharField(max_length=36, blank=True, editable=False)


import uuidstore.registry
uuidstore.registry.register(MyModel, uuid_descriptor='uuid')
```

Let's see how that works by running `python manage.py shell`:

```
>>> from myapp.models import MyModel
>>>
>>> # Create an instance of your model
>>> instance = MyModel.objects.create(title='Fizz')
>>>
>>> instance.uuid
u'e8a245fc-31ef-4160-906d-b6ba47449a26'
```

2.4 Denormalisation to a UUID field

```
# models.py
from django.db import models
from django_extensions.db.fields import UUIDField


class MyModel(models.Model):
    title = models.Charfield(max_length=60)
    uuid = UUIDField(
        auto=False,
        blank=True,
        null=True,
        unique=True,
        editable=False
        )


import uuidstore.registry
uuidstore.registry.register(MyModel, uuid_descriptor='uuid')
```

In this case we've replaced the CharField from the previous example with the UUIDField from django-extensions.

Let's see how that works by running `python manage.py shell`:

```
>>> from myapp.models import MyModel
>>>
>>> # Create an instance of your model
>>> instance = MyModel.objects.create(title='Buzz')
>>>
>>> instance.uuid
u'5da4a32e-0bb8-47ab-82fd-35110cf340a2'
```

Hint: You can use any suitable UUID field for this, provided that *uuidstore* can set its value *after* it has been saved. i.e. any auto behaviour is disabled, and you've set `blank=True, null=True`.

CHAPTER 3

Indices and tables

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