django-feedmapper Documentation

Release 1.0

National Geographic

August 18, 2015

Contents

django-feedmapper is a library for synchronizing data from feeds with Django models. The process of synchronizing the data requires the use of three pieces: a parser, a mapping, and a schedule.

CHAPTER 1

Installation

Install from PyPI:

```
pip install django-feedmapper
```

Add `feedmapper` to your `settings.py` file:

```
INSTALLED_APPS = (
    ...
    'feedmapper',
    ...
)
```

If you are using South, run the migrations:

```
./manage.py migrate feedmapper
```

Otherwise, run `syncdb`:

```
./manage.py syncdb
```

CHAPTER 2

Parsers

A parser defines methods for validating and parsing data from incoming feeds. There are two built-in parsers, `XMLParser` and `AtomParser`. You can write your own parser by subclassing the base `Parser` class.

CHAPTER 3

Mapping

A mapping is written in JSON and describes how and when data from an incoming feed should be mapped to Django models. You can perform the following types of mappings:

- One field in a model to one field from a feed
- One field in a model to multiple fields from a feed
- One field in a model to a transformer method on the model

You can also set the following properties on a mapping through the Django admin:

- Data source
- Synchronization schedule
- Purge existing data

3.1 An example: users

Let's get into an example. Suppose we have the following incoming XML data and we want to map each `<user>` to Django's `User` model:

```xml
<?xml version="1.0" ?>
<auth>
    <users>
        <user>
            <username>vader</username>
            <first_name>Anakin</first_name>
            <last_name>Skywalker</last_name>
            <email>vader@sith.org</email>
            <date_joined>2050-01-31T20:00-4:00</date_joined>
        </user>
        <user>
            <username>kenobi</username>
            <first_name>Obi-Wan</first_name>
            <last_name>Kenobi</last_name>
            <email>kenobi@jedi.org</email>
            <date_joined>2000-01-31T20:00-4:00</date_joined>
        </user>
    </users>
</auth>
```

We need to specify a JSON map:

```
{
  "models": {
    "myapp.Thing": {
      "nodePath": "users.user",
      "identifier": "username",
      "fields": {
        "username": "username",
        "email": "email",
        "name": ["first_name", "last_name"],
        "date_joined": {
          "transformer": "convert_date",
          "fields": ["date_joined"]
        },
      }
    }
  }
}
```

Let's break this down a bit. First, we can specify one or more models to map:

```
  "models": {
    "myapp.Thing": {
```

We need to tell the parser the path to all of the `<user>` elements:

```
      "nodePath": "users.user",
```

If the mapping has purging turned off, we need to supply a unique idenfier for Django ORM `get` calls. In this case our resulting ORM call would be `User.objects.get(username=username)`:

```
      "identifier": "username",
```

Now the fun part. Mapping the fields:

```
      "fields": {
        "username": "username",
        "email": "email",
        "name": ["first_name", "last_name"],
        "date_joined": {
          "transformer": "convert_date",
          "fields": ["date_joined"]
        },
      }
```

We've got example of all three types of field mappings here.

`username` and `email` are one-to-one mappings:

```
        "username": "username",
        "email": "email",
```

`name` is mapped to multiple fields. The parser will concatenate these fields, putting a space between them:

```
        "name": ["first_name", "last_name"],
```

`date_joined` uses a transformer, which is simply a method defined on your model to do some manipulation to the incoming data before inserting it in a field. Here we tell the parser that the `date_joined` field should map to the `date_joined` field in the XML but use the `convert_date` method to transform the incoming data:

```
          "date_joined": {
            "transformer": "convert_date",
            "fields": ["date_joined"]
          },
```

CHAPTER 4

Scheduling

There are two ways to schedule the synchonization of mappings.

4.1 Using django-celery

The first scheduling method, and the preferred, is to use django-celery. To take advantage of this scheduling method, take the following steps:

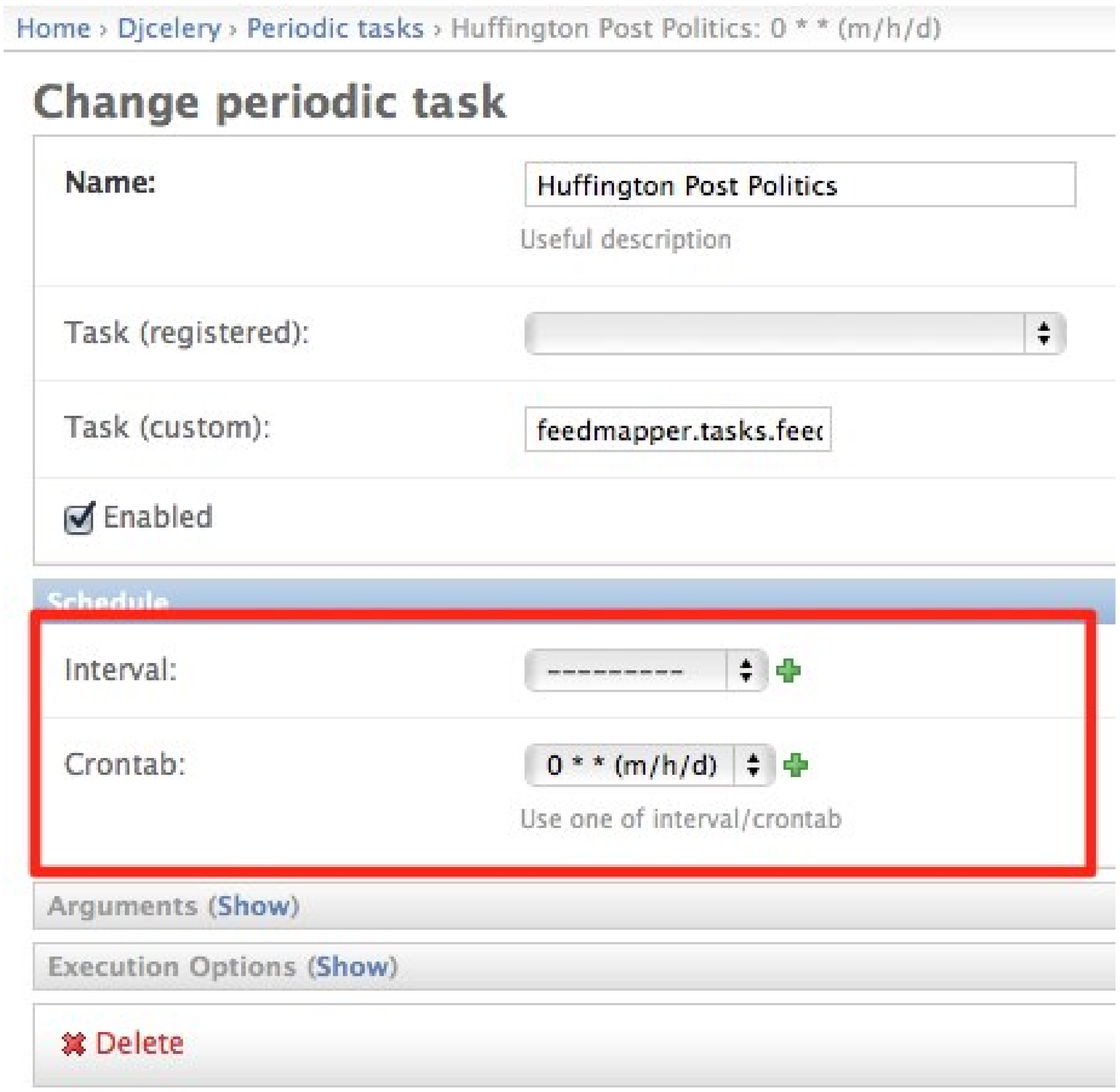
1. Install django-celery. If you've never done this before, it can be a little complicated. You'll want to read through the official docs. An example of some basic settings is in `example/settings_celery.py`:

```
from .settings import *

import djcelery
djcelery.setup_loader()

INSTALLED_APPS += ('djcelery',)
CELERYBEAT_SCHEDULER = 'djcelery.schedulers.DatabaseScheduler'
BROKER_HOST = "localhost"
BROKER_PORT = 5672
BROKER_USER = "guest"
BROKER_PASSWORD = "guest"
BROKER_VHOST = "/"
```

2. Make sure you enable the Django database scheduler of django-celery by adding the following to your `settings.py` file:

```
CELERYBEAT_SCHEDULER = 'djcelery.schedulers.DatabaseScheduler'
```

Now every time you save a mapping, it will either create or update a matching django-celery PeriodicTask in the database. By default the periodic task will run once an hour. If you want to change this, visit the PeriodicTask in the Django admin (`/admin/djcelery/periodictask/` by default) and modify the interval or crontab settings:


Home › Djcelery › Periodic tasks › Huffington Post Politics: 0 * * (m/h/d)

Change periodic task

Name:	Huffington Post Politics Useful description
Task (registered):	
Task (custom):	feedmapper.tasks.feed
✓ Enabled	

Schedule

Interval:	---------
Crontab:	0 * * (m/h/d) Use one of interval/crontab

4.2 Using feedmapper_sync

Of course, not everyone has resources or need to use a message queue solution. The second scheduling method is by setting up a cron job and using the `feedmapper_sync` management command. Make sure you have the `DJANGO_SETTINGS_MODULE` environment variable set and add the following to your crontab:

```
* * * * * /full/path/to/bin/django-admin.py feedmapper_sync
```

If you only want to sync a subset of the mappings you can supply one or more mapping IDs to the management command:

```
* * * * * /full/path/to/bin/django-admin.py feedmapper_sync 3 8 22
```

CHAPTER 5

Contributing

To contribute to django-feedmapper create a fork on github. Clone your fork, make some changes, and submit a pull request.

CHAPTER 6

Issues

Use the github issue tracker for django-feedmapper to submit bugs, issues, and feature requests.

CHAPTER 7

Contents

7.1 Reference

7.1.1 Parsers

7.1.2 Mappings

CHAPTER 8

Indices and tables

- genindex
- modindex
- search