GeoDjango
Geographic Web Applications for Perfectionists with Deadlines

Justin Bronn (jbronn@gmail.com)
Travis L. Pinney (travis.pinney@gmail.com)

FOSS4G 2007
September 25, 2007

Overview
• Background
• Third-Party Components and ctypes interfaces
• Building a Spatial Web Application with U.S. Census data.
Background

• Legal Stuff:
  – Django® is a registered trademark of Lawrence Journal-World (Lawrence, KS)
  – The opinions herein are solely the authors and do not necessarily represent or reflect the opinions of the Django Project and/or the Lawrence Journal-World.

Background – Quick Bios

• Justin Bronn
  – B.S. Computer Science, Trinity University, San Antonio, Texas
  – 4+ Years at Southwest Research Institute, San Antonio, Texas
  – Third-year Law Student, University of Houston

• Travis L. Pinney
  – B.S. Computer Science, Trinity University
  – 10+ years programming experience, including HP/Compaq
  – 6+ years developing in Python
Background
How’d we get into this stuff?
- Justin created a website, Houston Crime Maps using Django.
- In the process, learned open source GIS software.
- Wouldn’t it be nice if my hacks could be part of a coherent package…
  - Met Django developers at 2007 PyCon
  - “gis” branch created in February, obtained commit access in March.

Background – What is Django?
- Django
  - High-level rapid web application development framework. “For perfectionists with deadlines.”
  - Philosophies
  - “MTV”
    - Models
    - Templates
    - Views
Background – Why Django?

- Rapid web application development.
- RESTful URLs (e.g.,
  foo.com/calendar/2007/ instead of
  foo.com/calendar.asp?year=2007&session=331337)
- Lightweight

Background – What is GeoDjango?

- “The GIS branch of Django intends to be a
  world-class geographic web framework.
  Our goal is to make it as easy as possible
  to build GIS web applications and harness
  the power of spatially enabled data.”

1 http://code.djangoproject.com/wiki/GeoDjango
Background – Requirements

- Python
- PostgreSQL/PostGIS
- psycopg2
- PROJ.4
- GEOS
- GDAL/OGR (optional, but recommended)

Third-Party Components – Overview

- GEOS
- GDAL/OGR
- GeoIP
- Spatial Database Support
Third-Party Components – Why?

• Problem: Good GIS code, Troublesome SWIG Interfaces.
  – Compiling w/SWIG is daunting for new users, let alone experienced developers.
  – Cross-platform compatibility is problematic.

• GEOS
  – SWIG Python interface no longer supported, maintainer Sean Gillies left to work on PCL (Python Cartographic Library) and, more recently, ShapeLy.

• GDAL/OGR
  – Old bindings are no-longer supported; project transitioning to “next-generation” SWIG interfaces.
  – For example, GDAL 1.4.1 had issues w/Python 2.5.
Third-Party Components – ctypes Solution

- ctypes is a Python interface for accessing external C library capabilities in Python
  - "A fundamental shift in how to integrate languages."
- GeoDjango has ctypes interfaces for the following C APIs:
  - GEOS
  - GDAL/OGR
  - MaxMind GeoIP

Advantages
- No longer dependent on temperamental SWIG Python interfaces.
- Portability: interfaces are highly-cross platform.
  - For example, the GEOS interfaces work with Windows, Linux, Mac, and Solaris.
- Have access to very powerful routines in a "Pythonic" way:
  ```python
  mpoly = GEOSGeometry(wkt)
  for poly in mpoly:
    for ring in poly:
      ...
  ```

Third-Party Components – GEOS

```python
>>> from django.contrib.gis.geos import *
>>> pnt = Point(5, 23)
>>> ring = LinearRing((0, 0), (0, 50), (50, 50), (50, 0), (0, 0))
>>> poly = Polygon(ring)
>>> print poly.contains(pnt)
True
>>> print poly
POLYGON ((0.000000000000000, 0.000000000000000, 0.000000000000000, 50.000000000000000, 50.000000000000000, 50.000000000000000, 50.000000000000000, 0.000000000000000, 0.000000000000000))
```
Third-Party Components – GDAL

```python
>>> from django.contrib.gis.gdal import *
>>> s1 = SpatialReference(4326)
>>> s2 = SpatialReference('NAD83')
>>> s3 = SpatialReference('+proj=lcc +lat_1=27.5 +lat_2=35
    +lat_0=18 +lon_0=-100 +x_0=1500000 +y_0=5000000 +ellps=GRS80
    +units=m +no_defs')
>>> geom = OGRGeometry('POINT(5 23)')
```
Census Geoapp – Building a web enabled GIS application in 10 minutes (+ 10 minutes to explain what is being done)

- Application will be able to explore the State Boundaries, the County Boundaries, and the Urban Areas of the entire U.S.
- Shapefile Data obtained from the U.S. census bureau Cartographic Boundary Files website.

Census Geoapp – Overview

- Setting-up.
- Inspecting the Data
- Importing the Data with LayerMapping
- Spatial Queries
- Using databrowse and admin to build a dynamic website.
Census Geoapp – Setting Up

- Create new Django project
- Create `geoapp` application.
- Begin editing models.
- Edit settings.
- Syncing the database

```
C:\> python django-admin.py startproject foss4g
C:\> cd foss4g
C:\foss4g> python manage.py startapp geoapp
```
Census Geoapp – Inspecting the Data

- Start Django shell
- Use `ogrinfo` utility to explore shapefiles.
- Use `DataSource` to explore.
- Use only what you need in your models!

```
In [1]: from django.contrib.gis.utils import ogrinfo
In [2]: ogrinfo('C:/gis_data/st99_d00.shp,
   num_features=1)
```
In [1]: from django.contrib.gis.utils import ogrinfo
In [2]: ogrinfo('C:/gis_data/st99_d00.shp', num_features=1)
data source : C:/projects/gis_data/st99_d00.shp
==== layer 0
  shape type: Polygon
  # features: 273
  srs: None
  extent: (-179.14734000000001, 17.884813000000001) - (179.77847000000001, 71.352560643999809)
Displaying the first 1 features ====
==== Feature 0
  AREA: Real (271.25438)
  PERIMETER: Real (227.17142)
  ST99_D00_: Real (2.0)
  ST99_D00_I: Real (1.0)
  STATE: String ("02")
  NAME: String ("Alaska")
  LSAD: String ("01")
  REGION: String ("4")
  DIVISION: String ("9")
  LSAD_TRANS: String (None)

In [3]: from django.contrib.gis.gdal import *
In [4]: ds = DataSource('C:/gis_data/st99_d00.shp')
In [5]: layer = ds[0]
In [6]: print layer.fields
['AREA', 'PERIMETER', 'ST99_D00_', 'ST99_D00_I', 'STATE', 'NAME', 'LSAD', 'REGION', 'DIVISION', 'LSAD_TRANS']
In [7]: print max(map(len, layer.get_fields('NAME'))) 20
Census Geoapp – Geographic Models

- The following Geographic Models are available:
  - `PointField`
  - `LineStringField`
  - `PolygonField`
  - `MultiPointField`
  - `MultiLineStringField`
  - `MultiPolygonField`
  - `GeometryCollectionField`

```python
from django.contrib.gis.db import models

# Create your models here
class State(models.Model, models.GeoMixin):
    fips = models.CharField(max_length=2)
    name = models.CharField(max_length=20)
    region = models.CharField(max_length=1)
    division = models.CharField(max_length=1)
    poly = models.PolygonField(srid=4269)
    objects = models.GeoManager()

class Admin: pass

def __unicode__(self):
    return unicode(self.name)
```
Census Geoapp – Editing settings.py

DATABASE_ENGINE = 'postgresql_psycopg2'
DATABASE_NAME = 'foss4g'
DATABASE_USER = 'foss4g'
DATABASE_PASSWORD = 'foss'

INSTALLED_APPS = (
    'foss4g.geoapp',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.gis',
    'django.contrib.sessions',
    'django.contrib.sites',
)

C:\foss4g> python manage.py sqlall geoapp
BEGIN;
CREATE TABLE "geoapp_state" (
    "id" serial NOT NULL PRIMARY KEY,
    "fips" varchar(2) NOT NULL,
    "name" varchar(20) NOT NULL,
    "region" varchar(1) NOT NULL,
    "division" varchar(1) NOT NULL
);
SELECT AddGeometryColumn('geoapp_state', 'poly', 4269, 'POLYGON', 2);
ALTER TABLE "geoapp_state" ALTER "poly" SET NOT NULL;
CREATE INDEX "geoapp_state_poly_id" ON "geoapp_state" USING GIST ( "poly"
    GIST_GEOMETRY_OPS );
COMMIT;
C:\foss4g> python manage.py syncdb
Creating table geoapp_state
Installing custom SQL for geoapp.State model
Loading 'initial_data' fixtures...
No fixtures found.
C:\foss4g>

Census Geoapp – LayerMapping
- LayerMapping was created to automate repetitive code in importing and transforming geographical data from OGR-supported data sources (SHP files)
- Requires GDAL/OGR ctypes interface.
In [1]: from django.contrib.gis.utils import LayerMapping
In [2]: from foss4g.geoapp.models import State
In [3]: from foss4g.mapping import *
In [4]: lm = LayerMapping(State, shp_file, mapping,
   source_srs=4269, encoding='cp437')
In [5]: lm.save(verbos=True)

Census Geoapp – Spatial Queries

- Most of the PostGIS operations are available through GeoDjango’s database API.
Census Geoapp – Spatial Queries

Available Spatial Lookup Types

- overlaps, bboverlaps
- overlaps_left, overlaps_right
- overlaps_below, overlaps_above
- strictly_below, strictly_above
- left, right
- same_as/exact
- contained, bbcontains
- equals, disjoint, touches, crosses, within, intersects, relate
- dwithin, coveredby, covers (PostGIS 1.3.1+ only)
```python
In [1]: from foss4g.geoapp.models import County
In [2]: bexar = County.objects.get(name='Bexar')
In [3]: print bexar
Bexar
In [4]: bexar = County.objects.transform('poly', srid=3084).get(name='Bexar')
In [5]: print bexar.poly.geom_type
Polygon
In [6]: print bexar.poly.wkt
POLYGON ((1630726.0407146986000000 6311696.3955783 ... 1630726.0555427652000000 6311696.3955783 ...))
In [7]: print bexar.poly.srid
3084
In [8]: buf = bexar.poly.buffer(1.0)
In [9]: qs = County.objects.filter(poly__intersects=buf)
In [10]: print qs
[<County: Kendall>, <County: Comal>, <County: Bandera>,
   <County: Guadalupe>, <County: Bexar>, <County: Medina>,
   <County: Wilson>, <County: Atascosa>]
```
Census Geoapp – Configuring URLs

- URLs are configured with regular expressions.
- Parameters are strictly defined, no unexpected data.
- Encourages making your application RESTful

Databrowse dynamically creates a rich, browsable Web site by introspecting your models.

The admin interfaces provides a built-in utility for manipulating your spatial data.
Census Geoapp – Configuring Databrowse and Admin

```python
from django.conf.urls.defaults import *
from django.contrib import databrowse
from foss4g.geoapp.models import County, State, UrbanArea

databrowse.site.register(County)
databrowse.site.register(State)
databrowse.site.register(UrbanArea)

urlpatterns = patterns('',
    (r'^admin/', include('django.contrib.admin.urls')),
    (r'^(.*)$', databrowse.site.root),
)
```

- For more information, see
  http://code.djangoproject.com/wiki/GeoDjango
  http://www.djangoproject.com/

- We need help, bugfixes, implementing features, etc. – please contribute.
Any Questions?