HELLO WORLD

Web Maps of Renewable Energy

By Dmitry Batenkov

Geographical Information Systems (GIS) technology has long been a major component in modern applications. There are tools ranging from back-end geo-enabled database servers (such as PostGIS), through desktop viewers, analysis tools, to rich clients and, of course, software libraries and frameworks. One trend that has become popular in recent years is web mapping. In this tutorial we will learn how to create a simple interactive map application using some of the GIS technologies and tools.

Step 0: Software

For this tutorial we will need:

1. GeoServer: a software server that can expose geo-graphical data on the web using open standards and protocols. It should be installed and configured according to the instructions on its web site.

2. OpenLayers: a JavaScript library for creating interactive web maps. There is no need to download it to a local machine; we will use the hosted version from www.openlayers.org.

3. Optional: GDAL command-line tools. For Windows, one can use the FWTools bundle.

Step 1: Download

U.S. National Renewable Energy Laboratory (NREL) publishes several GIS data sets which are important for planning of renewable energy facilities. One of these is "Solar resource potential," which provides the average amount of solar energy available to a collector, such as a solar dish, averaged over 10km*10km squares, for the entire territory of the contiguous 48 states. The data is provided in a popular "shapefile" format. Go to http://www.nrel.gov/gis/cfm/input.cfm and fill in all the required information, then choose "GIS Data Technology Specific –> United States –> Solar –>High Resolution –> Lower 48 DNI High Resolution." Save the file and unzip it into a new directory (let's call it "`solar_data`" for definiteness). It contains a bunch of files that are collectively called "the shapefile."

Step 2: Understand

The data in a shapefile is organized into "layers," where each layer is just a data table, each row having an associated geometry (such as a point, a line, or a polygon) and several associated properties (the columns of the table). There are several options to inspect our solar data, such as loading it into a desktop GIS application (for example, QGIS, ArcView), or into a database with geospatial capabilities (for example, PostGIS). We will use the `ogrinfo` utility from the GDAL command-line tools. Open up a command prompt to the parent directory of `solar_data` and run the command `ogrinfo solar_data`:

```
#> ogrinfo solar_data
INFO: Open of 'solar_data'
      using driver 'ESRI
          Shapefile'successful.
1: us9805_dni (Polygon)
```

So our shapefile has a single layer called `us9805_dni`. To see its attributes, issue the following command:

```
> ogrinfo solar_data us9805_dni -so
INFO: Open of 'solar_data'
      using driver 'ESRI Shapefile '
successful.
Layer name: us9805_dni
Geometry: Polygon
Feature Count: 90508
Extent: (-125.100000, 24.200000) -
(-66.500000,
        49.700000)
...
LON: Real (7.2)
LAT: Real (5.2)
DNI01: Real (9.4)
...
DNI12: Real (9.4)
DNIANN: Real (9.4)
```

So each of the 90508 data rows ("features") is a polygon with attributes "..., DNI01,..., DNI12, DNIANN," These are the monthly (and annual) averages of the solar energy. Each polygon here is a 10km-by-10km square (this is evident from the documentation file `metadata.xml`).

Step 3: Expose

According to the principles of Service-Oriented Architecture, the Open Geospatial Consortium (OGC) defines several web service specifications through which clients can access and modify geospatial data. One of them is Web Map Service (WMS), which defines the protocol for retrieving raster maps (tiled or untiled). So we need a WMS server that can expose our data. For this task we will use GeoServer. Installation is quick and easy. All of the configuration can be done through a web interface accessible at http://localhost:8080/geoserver. Log in with default credentials (username "admin," password "geoserver").

Our first task is to define a "store" with our shapefile as its back-end. Click on "Stores" in the left sidebar, then "Add new store." Choose "Shapefile." Enter "solardata" as store name, and the full URL path to the `solar_data` directory. After clicking "Save," we need to select the `us9805_dni` layer (by clicking on the "Publish" link). Remember the name assigned to the layer—we will need it afterward (in the examples below it is `solar-48-highres`). Scroll down to "Coordinate Reference Systems." Enter "EPSG:4326" into the "Declared SRS" field, then scroll down to "Bounding boxes," click on "Compute from data" and then on "Compute from native bounds" (in this order). If all is well, you should be able to click on the "Save" button at the bottom of the form. From now on, GeoServer will serve the solar data through the WMS protocol.

Step 4: Style

Click on "Layer Preview" in the left sidebar, then scroll to our new layer and click on "OpenLayers" link. A pop-up will show up, displaying the map. But... it's all black! Zoom in, and you will see the 10km-by-10km squares. The problem is that we haven't really specified how our data should be rendered. We should now

define "styles." Click on "Styles" in the left sidebar, then "Add new style." The styles are specified in the so-called Styled Layer Descriptors (SLD) syntax. We will assign a fill color to each polygon according to the value in the "DNIANN" field.

I have prepared a simple style for our data which can be downloaded from the "Code" section of the XRDS website. So just upload it by clicking "Browse," then "Submit." Now we only need to assign the style to the layer. Go to layer properties, switch to "Publishing" tab and choose the new style from the "Default style" dropdown. Hit "Save." Go to the preview again, now the map is much nicer!
Note that Geoserver can automatically generate the legend graphic according to the style, which is retrievable via http://localhost/8080geoserver/wms?service=WMS&request=GetLegendGraphic&layer=nrel:solar-48-highres&format=image/png (here "nrel" is the workspace name I chose for the store).

Step 5: Client-Side

The last ingredient is to write an OpenLayers client which will send requests to the GeoServer WMS and display the map. In fact, the "Layer Preview" feature has generated such a client, but let us do this from scratch. Create a new HTML file with the following content:

```
<html><head><title>DNI Example</title>
<script src="http://www.openlayers.org/api/
      OpenLayers.js">
</script></head>
<body>
<div style="width:100%; height:100%"
id="map"><
      /div>
<script defer="defer" type="text/
javascript">
  var bounds =
    new OpenLayers.Bounds
(-125.1,24.2,-66.5,49.7);
  map = new OpenLayers.Map ('map');
  var solar=new OpenLayers.Layer.WMS (
    "solar-48-highres",
    "http://localhost:8080/geoserver/wms",
     {layers:'nrel:solar-48-
highres',tiled: 'true'}) ;
    map.addLayer (solar);
    map.zoomToExtent (bounds);
</script></body></html>
```

The "root" object is `OpenLayers.Map`. In our simplest example, it has a single WMS layer with our solar data. A layer can be either a "base layer" or an overlay. The map shows only one base layer at a given time, while the overlays can be turned on and off. Try and add another layer from the ones shipped with GeoServer. (Do not forget to add an `OpenLayers.Controls.LayerSwitcher` control as well). If you want a good background map, you should consider OpenStreetMap (OSM):

```
var osm = new OpenLayers.Layer. OSM ();
map.addLayer(osm);
```

In order to overlay the solar layer over the OSM, the solar layer should be retrieved from Geoserver in the Spherical Mercator projection (EPSG code 900913), which is different from the default WGS84 (EPSG code 4326). This is accomplished by specifying an additional option `srs:'EPSG:900913'` in the layer constructor, and also transforming the map bounds from (latitude, longitude) coordinates to metric ones. To make the solar layer an overlay, add the option `transparent:'true'` to the constructor. Please see the complete source code on the XRDS site.

There is much more to Open-Layers than showing a static map with several overlays. For example, we can retrieve from the WMS server the underlying data for a specific region. Add the following code just before the call to `zoomToExtent()`:

```
info = new OpenLayers.Control.
WMSGetFeatureInfo ({
  url:'http://localhost:8080/ geoserver/wms',
  queryVisible: true, maxFeatures:1,
  infoFormat:'application/vnd.ogc.gml',
  eventListeners:{
    getfeatureinfo:function (event) {
    map. addPopup (new OpenLayers.
Popup. FramedCloud (
    "data", map.
getLonLatFromPixel(event.xy),
    null, event.features[0].data.
DNIANN,
    null, true));}}});
map.addControl (info);
info.activate ();
```

When a user clicks on the map, the info control sends a WMS GetFeatureInfo request to the server asking for a single feature (remember, our features are 10-by-10km squares) near the point which was clicked, parses the output (in GML format), extracts the DNIANN property and displays it in a pop-up. Let's add a marker at this location. We would add it to an `OpenLayers.Layer.Vector`—another type of layer in Open Layers. A vector layer contains geometric features such as points, lines and polygons, which can also be edited dynamically. In order to display these features on a map, a style should be attached to the layer (similarly to the SLD styles of Geoserver). Let us define a vector layer with a simple style:

```
var markers = new OpenLayers.Layer.Vector(
  "Markers", {
  styleMap: new OpenLayers. StyleMap
({'default':{
    strokeWidth: 3, fillColor: "# FF5500",
    pointRadius: 6,
    label: '${dniann}',
    fontWeight:"bold", labelAlign:"rt"
}})});
map.addLayer (markers);
```

We assume that each geometric feature has an attribute "dniann." Now let us populate the new layer. Replace the call to `map.addPopup` inside the `getfeatureinfo` callback with the following code:

```
var ll = map.getLonLatFromPixel (event.xy);
var m = new OpenLayers.Feature.Vector(
  new OpenLayers.Geometry.Point(ll.
lon,ll.lat));
  m.attributes = {
    dniann:event.features [0]. data.DNIANN};
markers.addFeatures ([m]);
```

Here is the final map!

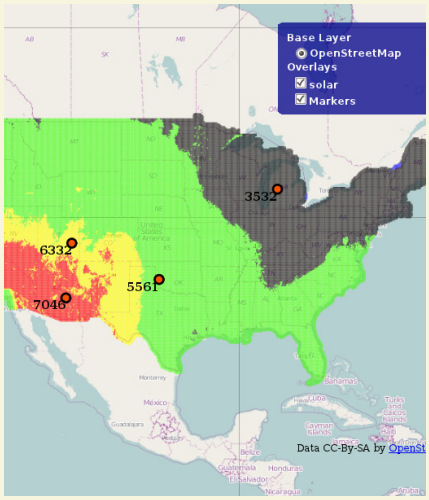

Figure 1: Solar map with styling.