Extracted from:

GIS for Web Developers

Adding *Where* to Your Web Applications

Chapter 8

OGC Clients

Here in our third (and final) OGC chapter, we look at applications and web frameworks that consume OGC services. We start with Mapbuilder, an Ajax framework that ships with GeoServer. Next, we look at another Ajax framework named OpenLayers. This toolkit was inspired by the architecture of Google Maps. We finish the chapter with uDig, a desktop viewer that allows you to mix OGC data sets with local shapefiles and PostGIS data sets.

8.1 Mapbuilder

Mapbuilder is the Ajax toolkit that powers the map previews in GeoServer. We're going to take a look at it in greater detail because you get it for free—why wouldn't you use it? Additionally, it gives us an excuse to look at another OGC standard, the Web Map Context file.

Recall that the preview maps are autogenerated each time GeoServer starts. (See Section 6.3, *Installing GeoServer*, on page 139.) That makes them a great learning tool. No matter how badly things get screwed up, you are always just a restart away from starting over with a clean slate. Of course, this can also be a hindrance. If you're not careful, all of your changes to the files can get blown away in a single reboot. Later in this section we'll copy a map out of harm's way so that our changes will be permanent.

Let's go hunting for those default maps. Take a look at geoserver/webapps/geoserver/preview. You should see three files per preview map. These files take the form [namespace]_[layername]. g4wd_st99_d00.html is the map. g4wd _st99_d00.xml is the OGC Context file. Finally, g4wd_st99_d00Config.xml is the Mapbuilder configuration file. Let's take a closer look at each one.

The HTML Map

Open g4wd_st99_d00.html in a text editor:

```
<html>
<head>
  <title>g4wd:st99_d00 Preview</title>
  <link rel="stylesheet" href="../../style.css" type="text/css">
  <link rel="stylesheet" href="../mb/lib/skin/default/html.css"
        type="text/css">
  <script type="text/javascript">
    var mbConfigUrl='g4wd_st99_d00Config.xml';
  </script>
  <script type="text/javascript" src="../mb/lib/Mapbuilder.js"></script>
</head>
```

In the head section, a couple of CSS files are linked in. The core Mapbuilder.js file is included as well. But most important, a pointer back to the Mapbuilder config file is created.

```
<body onload="mbDoLoad()">
<table border="0">
  <tr>
    <td valign="top" id="locatorMap"
        style="background-color: white;" />
    <td rowspan="2" valign="top">
      <table border="0">
        <tr>
          <td align="left" id="mainButtonBar"/>
          <td align="right" id="cursorTrack" />
        </tr>
        <tr>
          <td colspan="2" id="mainMapPane"
              style="background-color: white;" />
        </tr>
        <tr align="right">
          <td colspan="2">
            <table>
              <tr>
                <td align="left" id="mapScaleText"/>
                <td align="right">
                  Powered by
                  <a href="http://mapbuilder.sourceforge.net">
                    Community Map Builder
                  </a>
                </td>
                <td>
                  <img src="../mb/lib/skin/default/images/Icon.gif" alt="" />
                </td>
              </tr>
            </table>
          </td>
        </tr>
      </table>
    </td>
  </tr>
  <tr><td id="legend" /></tr>
  <tr><td colspan="3" id="featureList" /></tr>
  <tr><td colspan="3" id="transactionResponse" /></tr>
  <tr><td colspan="3"><div id="eventLog" /></td></tr>
</table>
</body>
</html>
```

Ignoring the cardinal sin of using HTML tables for page layout (hey, this is free code—you get what you pay for), what should leap out at you is the copious use of id attributes. These ids are placeholders for the various map widgets. The most important one of the bunch is mainMapPane—that is where the data layer appears. Everything else is reasonably well named. Widgets such as locatorMap, cursorTrack, and mapScaleText should leave little to the imagination in terms of what they do.

If you strip away everything else on the page, here is a bare-bones Mapbuilder map:

```
<html>
<head>
  <title>g4wd:st99_d00 Preview</title>
  <link rel="stylesheet" href="../../style.css" type="text/css">
  <link rel="stylesheet" href="../mb/lib/skin/default/html.css"
        type="text/css">
  <script type="text/javascript">
    var mbConfigUrl='g4wd_st99_d00Config.xml';
  </script>
  <script type="text/javascript" src="../mb/lib/Mapbuilder.js"></script>
</head>

<body onload="mbDoLoad()">
  <div id="mainMapPane" style="background-color: white;" />
</body>
</html>
```

Before we can try this bare-bones HTML, we need to "skinny" down the Mapbuilder config file as well. Right now it is expecting many more ids to be available on the page. It'll fail silently until we get those two files back in sync again. (OK, technically it will throw errors into the JavaScript console. But who looks there, right?)

The Config File

The Mapbuilder config file contains the instructions used to fill in the id placeholders with working widgets. Open g4wd_st99_d00Config.xml in a text editor. There's a lot going on, isn't there? The following is a greatly thinned-out config file. It won't actually run, but it will help us see the basic elements without getting bogged down in all the details.

```
<MapbuilderConfig>
  <models>
    <Context id="mainMap">
      <defaultModelUrl>g4wd_st99_d00.xml</defaultModelUrl>
      <widgets>
        <MapPane id="mainMapWidget">...</MapPane>
      </widgets>
    </Context>

    <Context id="locator">
      <defaultModelUrl>g4wd_st99_d00.xml</defaultModelUrl>
      <widgets>
        <MapPane id="locatorWidget">...</MapPane>
    </Context>
  </models>

  <widgets>
    <ZoomIn id="zoomIn">
      <buttonBar>mainButtonBar</buttonBar>
      <targetModel>mainMap</targetModel>
      ...
    </ZoomIn>
    <ZoomOut id="zoomOut">
      <buttonBar>mainButtonBar</buttonBar>
      <targetModel>mainMap</targetModel>
      ...
    </ZoomOut>
    <DragPan id="dragPan">
      <buttonBar>mainButtonBar</buttonBar>
      <targetModel>mainMap</targetModel>
      ...
    </DragPan>
    <Reset id="reset">
      <buttonBar>mainButtonBar</buttonBar>
      <targetModel>mainMap</targetModel>
      ...
    </Reset>
  </widgets>

</MapbuilderConfig>
```

Notice that the model element has two Contexts. The preview map has two maps—the main one in the center and the little map up in the left corner. Each Context has a pointer back to a OGC Context file. This, as you'll see in just a moment, is where you define the data layers to be displayed. Notice the clean separation of MVC concerns? Here, we're simply defining a map widget, which doesn't much care what data it displays. Defining the map layers and the styling is someone else's job.

Each Context has a list of widgets. I'm displaying only the important one here—the map widget. Notice that there are widgets defined outside of a context as well. These are the zoom buttons. They are tied back to a specific Context through the targetModel element.

Removing all of the extraneous stuff, here is a bare-bones Mapbuilder config file to go with our stripped-down HTML file:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<MapbuilderConfig version="0.2.1"
    id="referenceTemplate"
    xmlns="http://mapbuilder.sourceforge.net/mapbuilder"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://mapbuilder.sourceforge.net/mapbuilder
       ../../mapbuilder/lib/schemas/config.xsd">
  <models>
    <Context id="mainMap">
      <defaultModelUrl>g4wd_st99_d00.xml</defaultModelUrl>
      <widgets>
        <MapPane id="mainMapPane">
          <mapContainerId>mainMapContainer</mapContainerId>
        </MapPane>
      </widgets>
    </Context>
  </models>
  <skinDir>../mb/lib/skin/default</skinDir>
</MapbuilderConfig>
```

Save this file, and click the Refresh button in your browser. (See Figure 8.1, on the next page.) Notice that we don't have to update GeoServer when we make changes to these files. The server infrastructure is in place; we're just playing around in the web tier. All of the normal web development life cycles apply.

The OGC Web Map Context File

Let's take a look at the last file of the three. Open g4wd_st99_d00.xml in a text editor:

The Context file is short, sweet, and to the point. It defines the viewable nonspatial attributes of the map such as the size and the title. It also identifies the data layer(s) that should be included on the map. (You'll learn more about multiple layers in just a moment.)

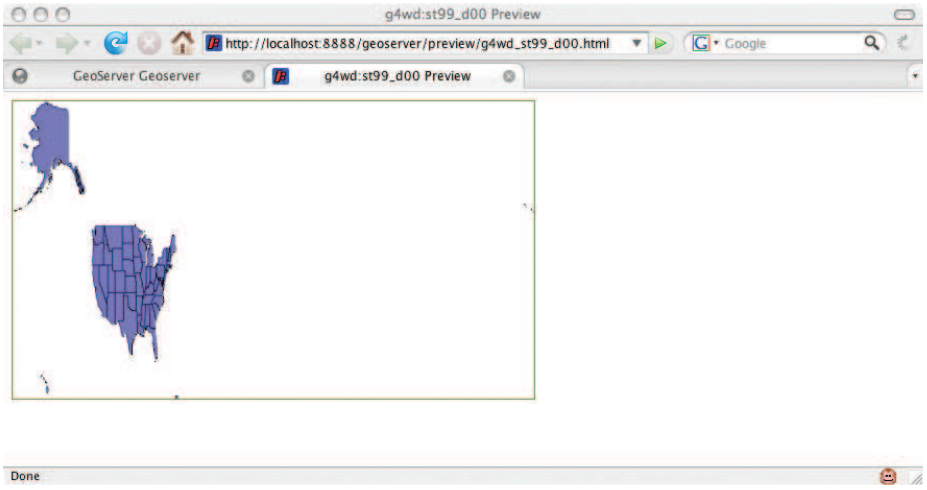

Figure 8.1: A simple Mapbuilder map

```
<ViewContext>
  <General>
    <Window width="500" height="285"/>
    <BoundingBox SRS="EPSG:4326"
                 minx="-179.14734"
                 miny="17.884813"
                 maxx="179.77847000000006"
                 maxy="71.35256064399981"/>
    <Title>g4wd:st99_d00 Map</Title>
    <KeywordList>
      <Keyword>g4wd:st99_d00</Keyword>
    </KeywordList>
    <Abstract></Abstract>
  </General>
  <LayerList>
    <Layer queryable="1" hidden="0">
      <Server service="OGC:WMS" version="1.1.1"
              title="g4wd:st99_d00 Preview">
        <OnlineResource xlink:type="simple" xlink:href="../wms"/>
      </Server>
      <Name>g4wd:st99_d00</Name>
      <Title>g4wd:st99_d00</Title>
      <SRS>EPSG:4326</SRS>
      <FormatList><Format current="1">image/png</Format></FormatList>
    </Layer>
  </LayerList>
</ViewContext>
```

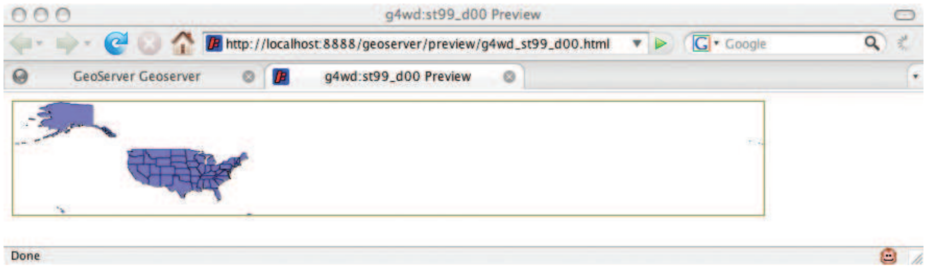

Figure 8.2: The U.S. map with better dimensions

Like the SLD file, the Context file is an OGC standard[1] that can be shared across server implementations. Write this file once, and it is reusable from one server to the next.

This file finally allows us to do something about our poor, misshapen U.S. map. The culprits are right there in plain sight: the Window and BoundingBox elements. The Window is the same size for all of the preview maps. The aspect ratio is roughly 2:1 (width:height); 500 pixels wide by 285 pixels high is a reasonable default if we assume a minimum screen resolution of 800 by 600 for our web visitors.

The problem is the dimensions of the BoundingBox. They don't come close to matching the ratio of the Window, giving us the dreaded "Silly Putty" effect once again. Let's naively pretend that EPSG 4326 is a planar projection to keep the concepts simple. We'll figure out in raw degrees what our map dimension should be and use them unchanged as pixel coordinates.

First, let's tackle the longitude. Notice that the min and max are both basically 180 degrees. That means the width of the map runs the full 360 degrees. (Recall that there are a couple of Alaskan islands that cross the International Date Line, making for a pretty wide map.) If we let 1 pixel equal 1 degree, then our Window should have a width of 360.

Looking at the latitude, the height should be roughly 71–17, or 54 pixels tall. That's not very tall, so let's double both values to give us a map 720 pixels wide by 108 pixels tall.

1. http://www.opengeospatial.org/standards/wmc

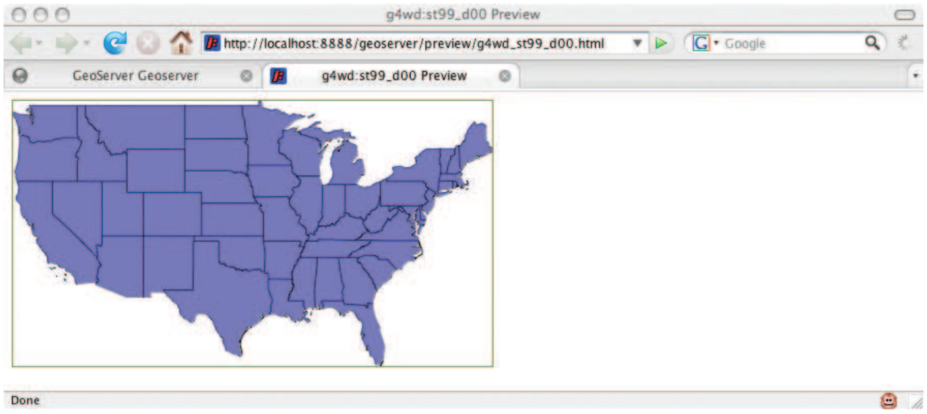

Figure 8.3: Adjusting the BBOX to the map aspect ratio

Save the file, and hit Refresh in your browser. (See Figure 8.2, on the preceding page.) The map dimensions might be kind of funny, but the data layer is visibly less distorted than it was before.

The other thing we can do is adjust the BBOX to something that fits the aspect ratio of the map. Open topp_states.xml in a text editor. Notice the BBOX it is using to frame just the lower 48 states:

```
<Window width="500" height="285"/>
<BoundingBox SRS="EPSG:4326"
             minx="-124.731422"
             miny="24.955967"
             maxx="-66.969849"
             maxy="49.371735"/>
```

Flip our BBOX and Window settings to match these values, and click Refresh in your browser. (See Figure 8.3.)

If we wanted to tweak the aspect ratio of the map using our naive algorithm, the dimensions are 58 by 32. Multiplying each by eight yields 464 by 256—pretty close to the existing 500 by 285.

Building a Permanent Map

OK, we've had our fun. Restart GeoServer to get the default map in place. Visit the preview map for st99_d00 one more time to make sure that it has all of the widgets back in place.

Now let's move it out of harm's way. Create a directory named g4wd in geoserver/webapps/geoserver/. Copy st99_d00*.* from preview to the new directory. We're unashamedly taking the easy way out here—remember all of those relative references to CSS and JavaScript files? By creating our own directory at the same depth as preview, we're ensuring that none of the paths will break.

To make sure that there is no aspect ratio distortion, set the BBOX to be the maximum possible and the Window to a perfect 2:1 ratio to match. Pull it up in a browser so that you can see your changes as you go.

```
<Window width="500" height="250"/>
<BoundingBox SRS="EPSG:4326"
             minx="-180"
             miny="-90"
             maxx="180"
             maxy="90"/>
```

As the name LayerList implies, a Context document supports multiple layers. What happens if you add the Canadian Provinces layer? Copy it from the Canadian Context document. While we're at it, let's change the titles to something a bit more user-friendly. The legend should reflect these changes. (See Figure 8.4, on the following page.)

```
<LayerList>
  <Layer queryable="1" hidden="0">
    <Server service="OGC:WMS" version="1.1.1"
            title="g4wd:st99_d00 Preview">
      <OnlineResource xlink:type="simple" xlink:href="../wms"/>
    </Server>
    <Name>g4wd:st99_d00</Name>
    <Title>US</Title>
    <SRS>EPSG:4326</SRS>
    <FormatList><Format current="1">image/png</Format></FormatList>
  </Layer>
  <Layer queryable="1" hidden="0">
    <Server service="OGC:WMS" version="1.1.1"
            title="g4wd:prov_ab_p_geo83_e Preview">
      <OnlineResource xlink:type="simple" xlink:href="../wms"/>
    </Server>
    <Name>g4wd:prov_ab_p_geo83_e</Name>
    <Title>Canada</Title>
    <SRS>EPSG:4326</SRS>
    <FormatList><Format current="1">image/png</Format></FormatList>
  </Layer>
</LayerList>
```

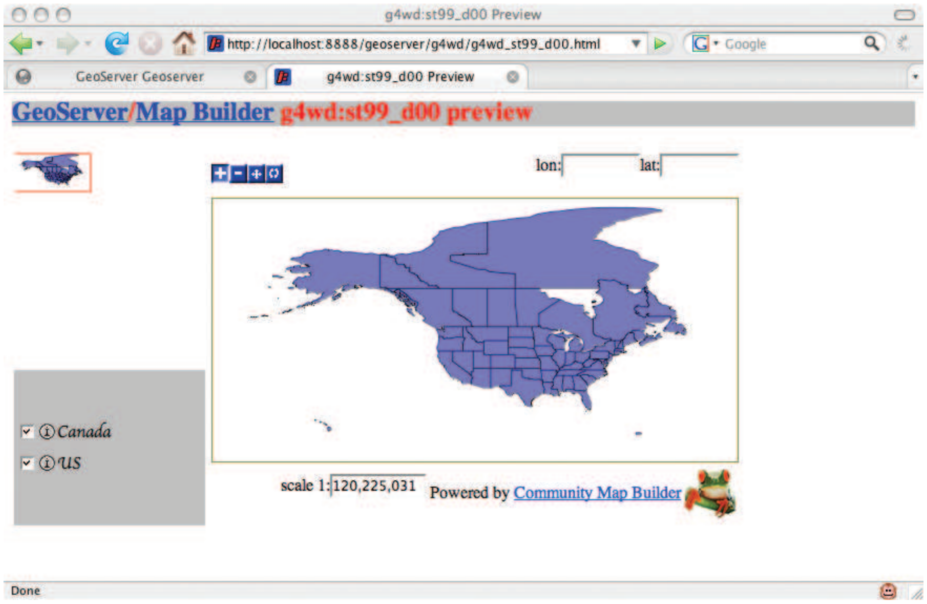
g4wd:st99_d00 Preview
http://localhost:8888/geoserver/g4wd/g4wd_st99_d00.html
GeoServer Geoserver
g4wd:st99_d00 Preview
GeoServer/Map Builder g4wd:st99_d00 preview
lon:
lat:
Canada
US
scale 1:120,225,031
Powered by Community Map Builder
Done

Figure 8.4: Mapbuilder displaying two layers

Notice that you can use the checkboxes in the legend to turn the layers on and off. Pretty cool, eh? We aren't limited to local data layers either. Let's add a live radar weather layer. Iowa State University offers this data up in an OGC feed:

```
<Layer queryable="1" hidden="0">
  <Server service="OGC:WMS" version="1.1.1" title="weather">
   <OnlineResource xlink:type="simple"
    xlink:href="http://mesonet.agron.iastate.edu/cgi-bin/wms/nexrad/n0r.cgi"/>
  </Server>
  <Name>nexrad-n0r-m45m</Name>
  <Title>Weather</Title>
  <SRS>EPSG:4326</SRS>
  <FormatList><Format current="1">image/png</Format></FormatList>
</Layer>
```

If you copy one of the existing layers, you need to adjust four values. The server Title attribute needs to be something unique. The OnlineResource HREF can point either to a local server or to a remote one. (Technically, it needs to point to that server's GetCapabilities document. Surprised? You shouldn't be.) You should change the Name element to the name of the data layer. Finally, change the Title element to what you'd like to appear in the legend.

The way this weather data gets to us is interesting. The National Oceanic and Atmospheric Administration (NOAA) offers a free weather web service,[2] but unfortunately it is SOAP-based. We can get the data, but not in a format that can be easily mapped. The Iowa State University Department of Agronomy offers the same data, but as a WMS service.[3] Are you beginning to see the power of a standards-based solution?

Remember our old friend the Blue Marble raster set? NASA offers it up as an WMS service.[4] Let's add it our map. Put it at the end of the LayerList.

```
<Layer queryable="1" hidden="0">
  <Server service="OGC:WMS" version="1.1.1" title="blue marble">
    <OnlineResource xlink:type="simple"
      xlink:href="http://wms.jpl.nasa.gov/wms.cgi?"/>
  </Server>
  <Name>BMNG</Name>
  <Title>Blue Marble</Title>
  <SRS>EPSG:4326</SRS>
  <FormatList><Format current="1">image/png</Format></FormatList>
</Layer>
```

Hmmm. Did your vector layers disappear? Opacity issues, right? Move the Blue Marble layer to the top of the list, and refresh your browser. Better? Good. (See Figure 8.5, on the next page.)

Now that we have several layers interacting, we might want to go back and play with the SLDs a bit more. Maybe you'd like to turn off the fill color in the U.S. and Canadian data layers. Maybe you want to tweak the borders to bright yellow so that they stand out against the dark Blue Marble background. The possibilities are endless.

Unfortunately, finding WMS servers on the Web is as hit or miss as finding the raw data. The upside is that once you've found a server, integrating it is a breeze (as we just demonstrated). And asking whether a website supports WMS is a pretty unambiguous question. Either it does or it doesn't. For example, you can pull data from TerraServer-USA via WMS.[5]

A couple of good directories of WMS services are available. Refractions Research[6] (the folks behind PostGIS) uses the Google Web API to har-

2. http://www.weather.gov/xml/
3. http://mesonet.agron.iastate.edu/ogc/
4. http://onearth.jpl.nasa.gov/
5. http://terraserver.microsoft.com/WebServices.aspx
6. http://www.refractions.net/white_papers/ogcsurvey/index.php

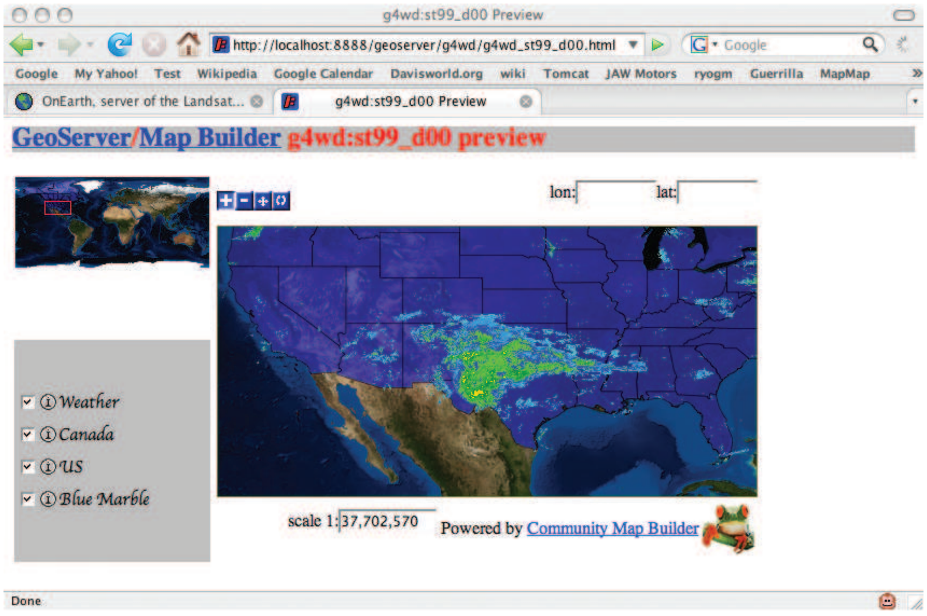
g4wd:st99_d00 Preview
http://localhost:8888/geoserver/g4wd/g4wd_st99_d00.html
Google My Yahoo! Test Wikipedia Google Calendar Davisworld.org wiki Tomcat JAW Motors ryogm Guerrilla MapMap
OnEarth, server of the Landsat...
g4wd:st99_d00 Preview
GeoServer/Map Builder g4wd:st99_d00 preview
lon:
lat:
Weather
Canada
US
Blue Marble
scale 1:37,702,570
Powered by Community Map Builder
Done

Figure 8.5: Pulling in data layers from remote servers

vest servers from the across the Web. ExploreOurPla.net[7] offers a big generated listing of WMS servers as well.

Take this opportunity to poke around these listings and find some other interesting data layers. Knowing that you are just a copy and paste away from a new data set makes the power of OGC interfaces manifest.

8.2 OpenLayers

Why introduce another Ajax mapping framework? Nothing is intrinsically wrong with Mapbuilder. There are, however, a couple of reasons why I find OpenLayers[8] an attractive alternative:

- I can create a map in significantly fewer lines of code using a single file instead of three.

7. http://exploreourpla.net/gis/wms-servers/
8. http://openlayers.org/