

Web Mapping- Why? How?

1. **Why Web Mapping?**
2. **Explanation of stack/structure**
3. **Data**
4. **Map server**
5. **WMS/WFS**
6. **Mapping API**
7. **Cataloging**

Why web mapping?

Web is *the* communication medium of our time. It's becoming more and more a part of every day life. Maps are inherently the best way of visualising geospatial data. It's a knock-out combination.

Web mapping is about more than google maps, mashups, or ArcIMS. It's possible to build your own "stack" (more later) and control every aspect of your mapping application, to build it all with standards-compliant software and file formats, without worrying about licensing, and without costing a penny (other than your time, and time *is* money).

What's a stack?

The "stack" is the set of components needed to make a web-mapping application. The formal name for the stack is a Spatial Data Infrastructure, or SDI. In some packages, various (or all) aspects of the stack/SDI are merged into one application. It can, however, be easier to maintain a stack if you choose a modular approach and choose a dedicated piece of software for each aspect.

The stack components:

In simple terms there are three layers to the stack. The first is your data, vector or raster, stored in a file system or database. The second is the map server, which translates your data into web-compatible formats. These are then delivered to your map on your web page.

Why do I need to know all of this?

Because if you build your own web-based map you need to understand each component of the stack.

(Finally) the point of this afternoon's session:

I will explain the different levels of the stack and talk about some (not all) of the key packages that drive each component. At the end, you will set up your own map server, connect to some data, and display it on a web page.

Data:

Data comes in two main forms- vector and raster, as discussed in the earlier sessions. If you're going to display it online, you need to make sure that it is stored somewhere the web server can access it. This means it must be in folders that the web server can read/write, and if it's stored in a database you need to make sure that the database is

configured for web access. It is even more necessary to ensure that your data is "clean" (topologically sound and so on) and has the correct metadata and coordinate systems. That is all your web server has to go on- it can't come and ask where or what something is. The cataloguing component (discussed later) may help with this side of things.

Map server:

In simple terms, a map server takes a request from a web page and returns some data. It interprets bounding box requests to decide how much data to return, and ensures that it is in the correct projection. The two main options for map servers are minnesota mapserver (mapserver to it's friends) and geoserver.

Mapserver, in it's simplest form, is a cgi application that lives on a web server. It takes requests in the form of parameters tacked on to the end of a URL, consults a text-based configuration file known as a map file, and returns data to the web server. It is very easy to install and use- binaries are available for Windows and Mac OSX, packages are available for many linux distributions, and the source code is publically available.

For windows, it is possible to download a web server and mapserver setup pre-configured and ready to go with all the support files (mapserver uses a set of open source libraries and tools for dealing with spatial references and format read/write/conversion). This can be downloaded at <http://maptools.org/ms4w>. Alternatively it can be installed on an existing web server, although it's now quite hard to find the separate components in windows form!

Once installed, you can test mapserver is running correctly by calling it from a command prompt:

C:\path\to\mapserv.exe (or in this case */var/www/cgi-bin/mapserv*) should return the following output: "This script can only be used to decode form results and should be initiated as a CGI process via a httpd server".

This is good!

You can also check the version by running *mapserv.exe -v*. This will show you all the options mapserver was compiled with, including the input and output file formats, and the server options that it supports.

Finally you can check mapserver from the web by running:

http://localhost/cgi-bin/mapserv.exe?

or in this case:

http://www.maths.lancs.ac.uk/cgi-bin/mapserv?

If mapserver is set up correctly you should get a web page saying "No query information to decode. QUERY_STRING is set, but empty.", in other words, mapserver is working but hasn't been supplied with any information. Note the syntax of the web request- using unix it should say *mapserv*, rather than *mapserv.exe*, but you always need the question mark.

As well as serving data in web-compatible formats, mapserver can also be used to create the map itself, so contains a lot of options for creating reference maps and scale bars that are not relevant when using it primarily to serve data.

The important part of any mapserver installation is the map file. This is a plain-text configuration file that lists the components of the map as objects. Some are optional, some are not. These are nested within the main MAP object, and all finish with an END statement. Comment lines are marked with a # sign. Note that text within the map file is **not** case-sensitive!

The basic components are as follows:

MAP- defines the settings for the whole map such as the size and extent (includes one or more LAYER objects)

LAYER- defines the individual layers or data sources within the map, including their name and type (polygon, point, line) (includes one or more CLASS objects)

CLASS- defines the set of objects relating to the way the features will be rendered, such as whether they will be labelled (includes one or more STYLE objects)

STYLE- defines how symbols will be rendered, such as the colour of dots or the thickness of lines.

These work as follows:

```
MAP
...
  LAYER
    ...
      CLASS
        ...
          STYLE
            ...
          END
        END
      END
    END
  END
```

In order to make your data accessible to the web server, copy it from your H: drive to \\wwwmaths\yourusername. This translates to /web/home/yourusername, and is web-accessible at <http://www.maths.ac.uk/~yourusername>

Important things that you need to know about your data are the coordinate system, units of measurement, and extent (bounding box). It is possible to find these things out in a number of ways, but perhaps the quickest way is to use the mapserver export plugin in Quantum GIS. Once loaded, this can be used to create the entire map file for you. At this stage, please tick the box "Export layer definition only", as we don't need the advanced elements.

Here's an example of a simple map with a single layer. You should find this saved as test.map in your home folder. You will need to edit the SHAPEPATH line to use your username

```
MAP
```

```

NAME "My first map"
UNITS dd
EXTENT -10908931.354601 -2813375.945688 8689298.182275 11986946.775106
PROJECTION
    'init=epsg:4326'
END
IMAGETYPE PNG
SIZE 400 300
SHAPEPATH "/web/home/cookj1/data/vmap0_shapefiles"
IMAGECOLOR 255 255 255
LAYER
    NAME alaska
    DATA alaska
    STATUS on
    TYPE POLYGON
    PROJECTION
        'init=epsg:4326'
    END
    CLASS
        NAME "Alaska"
        STYLE
            COLOR 232 232 232
            OUTLINECOLOR 32 32 32
        END
    END
END
END

```

Mapserver comes with built-in tools to convert a map into a static image:

```
/path/to/shp2img -m my.map -o mymap.png
```

Or you can test it from a web page using the following url:

```
http://www.maths.lancs.ac.uk/cgi-bin/mapserv?map=/web/home/cookj1/test.map&mode=map
```

You will need to change the map parameter to point at your username.

The end result of either of these should be a png image with a very exciting map of Alaska. Any errors in your map file will result in the commands failing, so this is a good way of testing.

However, if you want to create a dynamic web-based map you have to do a little more work. Mapserver itself can generate dynamic maps in html pages, with some additions to the map file, but without a lot of php coding the results are a little old-fashioned. Openlayers gives a more intuitive "slippy" interface so we'll concentrate on that.

You can use the demo setup

```
(http://www.maths.lancs.ac.uk/~rowlings/MapServer/workshop-5.0/)
```

on the webserver to see how the different components work together to produce simple maps.

Geoserver is a java-based map server. It is purely a map server, and can't be used to create maps on it's own, it just serves data to mapping apis. It has more sophisticated layer styling and controls than mapserver, so can produce a more attractive result. It has binaries for windows, mac OSX and unix variants, and source code. Installation on windows is relatively straightforward- download the executable, follow the instructions, start the server, and go to the web configuration pages. It

generally runs as a service, so it starts automatically when you start windows, but can be run manually.

Actually using geoserver is more complex than mapserver if all you want is to create simple web servers. You start by creating DATASTORES, which point at data sources (eg shapefiles). You then create FEATURE TYPES, which point at the DATA STORES and describe their symbology, their projection and so on. Finally, you can preview the FEATURE TYPES using the integrated openlayers client. This is all done via the web interface, in comparison to mapserver's text-based configuration file. For the purposes of today's session, we'll use mapserver as it's a bit easier to manage and understand.

Web Services for Mapping

In some ways these are the coolest parts of this session, including the (hopefully) snazzy maps you create. Web Services allow you to deploy your geospatial data from a central source via a URL. Rather than having many copies of your data in diverse files and worrying about version control and access rights, you can keep one copy of your data in a central location and have everyone access it via a URL. Even desktop gis clients can access web mapping services. The most common ones for use with mapserver are WMS (web mapping service) and WFS (web feature service).

WMS: WMS allows you to share and access vector and raster data as an image. It is useful for sharing background mapping data, or in situations where you need a static image rather than something interactive. Maps are requested with a GETMAP request, which has the following required parameters:

Common Name	Description	WMS Example
Type of service	Tell the server you are making a WMS request	service=wms
Request a map	Tell the server that you want a map (rather than information about the server, for example)	request=getmap
Specify the version of wms to use	Some WMS clients only support certain versions	version=1.1.1
Projection or spatial reference to use	Tell the server what projection you want the information in (some servers can serve data in more than one projection)	srs=EPSG:4326
Image format	What format should the image be sent in?	format=image/jpeg
Layer name or data source	Names used by the server to describe the layers you want in your image	layers=Countries, road, water, cities
Image size	How big should the image be, in pixels	width="1024" height="1024"
Geographic extent	Two pairs of coordinates defining the southwest and northeast corners of the map	bbox=-170 0, -50 90

To understand the capabilities that a wms server has, you need to make a GetCapabilities request to it. Note that different types of webservers have slightly different addresses, or have simplified the URL that you need to enter. A simple request looks like this:

<http://wms.jpl.nasa.gov/wms.cgi?request=GetCapabilities>

You can either type this as a URL in a web browser, or using something like wget. This will produce an file that you need to save as xml format and open with a text editor. When you open it up you will see information about the wms service offered by this server, such as the contact details for the person maintaining the data, the image formats it outputs in, and of course the layers that it offers. Given that information, you can manually request a layer from the server by making up the URL using the above parameters. eg:

*[http://wms.jpl.nasa.gov/wms.cgi?
request=GetMap&service=wms&version=1.1.1&srs=EPSG:4326&format=image/jp
eg&styles=&bbox=-180,-60,180,
84&width=600&height=300&layers=global_mosaic](http://wms.jpl.nasa.gov/wms.cgi?request=GetMap&service=wms&version=1.1.1&srs=EPSG:4326&format=image/jpeg&styles=&bbox=-180,-60,180,84&width=600&height=300&layers=global_mosaic)*

Mapserver has to be configured at both MAP and LAYER level to work as a WMS server. You need to add a WEB object, in which you need to add a METADATA object as follows:

MAP

...

WEB

...

METADATA

"wms_title" "My first wms server"

"wms_srs" "EPSG:4326"

"wms_onlineresource" "http://www.maths.lancs.ac.uk/cgi-bin/mapserv?

map=/web/home/cookj1/test_wms.map

END

END

...

END

Note that there are many other optional parameters such as contact information, access permissions and so on.

For each layer that you want to enable as a web layer you also need to add a METADATA object inside the LAYER object, and change a couple of the other options:

LAYER

...

STATUS ON

MINSCALE 1000

MAXSCALE 10000000

METADATA

"wms_title" "My first wms layer"

"wms_srs" "EPSG:4326"

END

END

The status option allows the layer to be turned ON and OFF. The third option DEFAULT means that it is always on and can't be turned off.

The MIN and MAXSCALE prevent the layer being drawn at unsuitable scales such as too high a resolution. They are related to the UNITS keyword, which specifies the units that the map is measured in.

You can check your map server is correctly configured by running your own GetCapabilities request. The format of the request differs slightly depending on how the server is set up, but it usually looks something like this:

```
http://localhost/cgi-bin/mapserv?  
map=/path/to/map&request=GetCapabilities&service=wms
```

or in this case:

```
http://www.maths.lancs.ac.uk/mapserv?  
map=/web/home/cookj1/test_wms.map&request=GetCapabilities&service=wms
```

If things are correctly configured, this should return a file that needs saving as an xml file as you did previously. Open it in a text editor and check that there are no errors. To check a GetMap request at the command line requires manually typing in the bounding box. You can do this if you want, or test it in Quantum GIS by adding a WMS layer and entering the GetCapabilities URL. Both methods should highlight any errors in the URL that you are using.

WFS: Web Feature Services return vector data rather than a raster image. This is more labour intensive for the server, but the results can be nicely overlaid on to other data, and you may be able to save the features as a data file for your own purposes.

Again, the capabilities of a WFS server can be determined using a URL request:

```
http://map.ns.ec.gc.ca/envdat/map.aspx?  
service=WFS&version=1.0.0&request=GetCapabilities
```

In this case the "layers" are called FEATURETYPES. Confusingly these are referred to as TYPENAMES in the GetFeature (equivalent to GetMap) request. The output will not be a map, but will be in Geography Markup Language (GML) format.

So, to manually request data from a WFS source via a URL:

```
http://map.ns.ec.gc.ca/envdat/map.aspx?  
service=WFS&version=1.0.0&request=GetFeature&typename=envirodat
```

Again, to convert your map server into a WFS server you just need to add some additional text to the WEB and LAYER objects. Note that your data can be served in both WMS and WFS format at the same time from the same map by adding both sets of parameters:

MAP

...

WEB

```

...
METADATA
  "wfs_title" "My first wfs server"
  "wfs_srs" "EPSG:4326"
  "wfs_onlineresource" "http://www.maths.lancs.ac.uk/cgi-bin/mapserv?
  map=/web/home/cookj1/test_wfs.map
  END
END
LAYER
  ...
  STATUS ON
  MINSCALE 1000
  MAXSCALE 10000000
  DUMP TRUE
  METADATA
    "wfs_title" "My first wfs layer"
    "gml_featureid" "cat"
    "gml_include_items" "all"
  END
END
...
END

```

The DUMP TRUE parameter gives mapserver permission to send the raw data to the client and works just as well for wms layers. The gml_featureid parameter specifies the field from the attribute data that you wish to use as an ID field in gml. The gml_include_items parameter is either "all", ie you wish to expose all of the attribute data in the gml output, or a comma-delimited list of fields.

You can run your own GetCapabilities request to check your server is correctly configured:

```

http://localhost/cgi-bin/mapserv?
map=/path/to/map&service=wfs&request=getcapabilities&version=1.0.0

```

or in this case:

```

http://www.maths.lancs.ac.uk/mapserv?
map=/web/home/cookj1/test_wfs.map&request=getcapabilities&version=1.0.0

```

Web API/Web Pages:

For this section we're going to work with openlayers, which is a javascript-based web application. This gives it the enormous advantage that it can be used without any additional software installed on your webserver, if you can pull data in from remote sources. If you build a web page entirely with mapserver, or some other web-mapping package, you need to have them installed on your webserver, which many commercial web hosts might not allow.

Luckily you don't really need to know much/any javascript to work openlayers in it's basic form, but you do need to respect the syntax and the order in which the components are declared and used.

See the following code:

```
<html>
  <head>
    <script src="http://www.maths.lancs.ac.uk/ol/lib/OpenLayers.js"></script>
    <script type="text/javascript">
      var map;
      function init() {
        map = new OpenLayers.Map('map');
        var wms = new OpenLayers.Layer.WMS(
          "OpenLayers WMS",
          "http://labs.metacarta.com/wms/vmap0?",
          {layers: 'basic'}
        );
        map.addLayers([wms]);
        map.zoomToMaxExtent();
      }
    </script>
  </head>
  <body onload="init()">
    <div id="map" style="width: 600px; height: 300px"></div>
  </body>
</html>
```

This works in the following way:

1. The OpenLayers library is loaded using the script tag.
2. There is a div element with the id “map” that contains the map.
3. A map is created using JavaScript.
4. A layer is created using JavaScript, and then added to the map.
5. The map zooms out as far as possible.
6. JavaScript gets called only after the body tag’s onLoad event.

Step 1 (red): Getting the OpenLayers Library

```
<script src="openlayers/lib/OpenLayers.js"></script>
```

The openlayers/lib/OpenLayers.js URL points to the location of a JavaScript file that loads OpenLayers. You can use debugging tools such as Firebug in firefox to explore the OpenLayers library from your browser.

Step 2 (orange): The Map in the Document Object Model (DOM)

```
<div id="map" style="width: 600px; height: 300px"></div>
```

This is a container for the map that we are creating in our page markup. Later, when we initialize the “map” object, we will give the id of this div element to the map’s constructor.

The size of the map is determined by the size of the element that contains it. Here, we have set the size of the map with an inline style declaration to be 600 pixels by 300 pixels. (Note that these style declarations are better placed in a <style> element - we’re just putting them directly on the div element for simplicity here).

Step 3 (yellow): The Map Object

```
map = new OpenLayers.Map('map');
```

There are lots of "maps" in this statement! The first declares the variable, to give it a short name that can be used elsewhere in the javascript code. Since the variable was declared outside of the function that initialises it, it can be used globally, which helps for debugging purposes.

The second- OpenLayers.Map is called an object, and tells the openlayers javascript that we are using the Map class, which means that all of the built-in parameters and options can be accessed. This can take other optional arguments as well, but not in this example!

The third is the name or identifier of the div that we are putting the map in, so openlayers knows where to draw the map on the page.

So this section basically creates a new OpenLayers.Map object and gives it the variable name map.

If you are using Firebug, try typing map into the console. This will allow you to explore the map object and all its properties.

Step 4 (blue): Creating a Layer

```
var wms = new OpenLayers.Layer.WMS(  
  "OpenLayers WMS",  
  "http://labs.metacarta.com/wms/vmap0?",  
  {layers: 'basic'});  
map.addLayers([wms]);
```

OpenLayers organizes a single map into several layers. This part of the code tells openlayers that this layer is to be a new WMS layer, and that it can be referred to elsewhere as "wms". Layers have several parameters that sit within the () brackets- these change depending on the type of layer you are creating. The first in this case is the name that it will be referred to in the legend on the page, and the second is the address of the wms server- this then has it's own parameters such as the layer name as per the wms getcapabilities return.

Step 5 (purple): Positioning the Map

```
map.zoomToMaxExtent();
```

This code tells the map to zoom out to its maximum extent, which by default is the entire world. It is possible to set a different maximum extent as an option in the Map constructor.

OpenLayers maps need to be told what part of the world they should display before anything will appear on the page. Calling `map.zoomToMaxExtent()` is one way to do this. Alternative ways that offer more precision include using `zoomToExtent()` and `setCenter()`.

Step 6 (green): Loading the Map

```
<body onload="init()">
```

In order to ensure that your document is ready, it is important to make sure that the entire page has been loaded before the script creating the map is executed.

We ensure this by putting the map-creating code in a function that is only called when the page's `<body>` element receives the `onload` event.

Layers in more detail

WMS Layers (oltest1.html)

A WMS layer is created in OpenLayers with four arguments (the fourth one being optional). These are as follows:

name	{String} A name for the layer
url	{String} Base url for the WMS
params	{Object} An object with key/value pairs representing the GetMap query string parameters and parameter values
options	{Object} Hashtable of extra options to tag onto the layer

The parameters defined as `{objects}` take several options and enclose them in `{}` brackets. At the very least these will include the list of layers to return from the wms:

```
var wms = new OpenLayers.Layer.WMS("NASA Global Mosaic",  
    "http://wms.jpl.nasa.gov/wms.cgi",  
    {layers: "modis,global_mosaic"});
```

Note that the you can only have one base mapping layer in openlayers. By default, WMS layers are baselayers, but if you don't want your layer to be a base layer you need to add the following:

```
{isBaseLayer: false}
```

after the `{layers:...}` parameter. Furthermore, to set the background of your wms tiles to be transparent you need to add the following key-pair WITHIN the layers parameter:

```
{(layers: ...), transparent: true}
```

Commercial Layers (oltest2.html, oltest_wms.html)

OpenLayers provides support for Google Maps, Yahoo Maps, Virtual Earth, and MultiMap, so these can be used as background mapping for your own layers. Using any of the commercial layers requires a further parameter in the <script> section of the code, to include their api key. This needs to be on one line with no spaces. Furthermore, commercial layers use spherical mercator projection, which means that layers probably have to be reprojected from their original coordinate system (say British National Grid) to overlay correctly.

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <script src='http://maps.google.com/maps?
      file=api&v=2&key=ABQIAAAjpkAC9ePGem0llq5XcMiuhr_
      wWLPFku8Ix9i2SXYRVK3e45q1BQUd_beF8dtzKET_EteAjPdGDwqp
      Q'></script>
    <script src="openlayers/lib/OpenLayers.js"></script>
    <script type="text/javascript">
```

To use (for example) a google layer as your base layer, you need to pass at least two options- the first is the name of the layer as it will appear in the legend on the map and the second is the map type (eg google satellite):

```
var satellite = new OpenLayers.Layer.Google(
  "Google Satellite" ,
  {type: G_SATELLITE_MAP}
);
```

Note that we can also add a layer switcher to allow you to toggle layers on and off, by adding a new feature called a control to the init function:

```
map.addControl(new OpenLayers.Control.LayerSwitcher());
```

To get your own layer to display correctly on top of the google layer, you will first need to set the projection that openlayers will use to call data from your webserver. This should be one of the projections listed in the wms_srs metadata in your map. In your web page add the following code:

```
var map
var options = {
  projection: new OpenLayers.Projection("EPSG:4326"),
};
```

```
function init() {
map = new OpenLayers.Map('map', options);
```

You need to get the bounds for your layer from the GetCapabilities document for your webserver. Note that an extra parameter (options) has been added to the map object to pass it this information. Finally, you need to include the option {reproject: true} as the final option in the layer description. eg:

```
var wms = new OpenLayers.Layer.WMS("Alaska",
  "http://www.maths.lancs.ac.uk/cgi-
  bin/mapservmap=/web/home/cookj1/test_wms.map",
  {layers: "alaska"},
  {'isBaseLayer': false});
{reproject: true}
);
```

Don't forget to tell openlayers to load this layer at startup as well:
map.addLayers([satellite, wms]);

You can also set the initial extent of your map so that it doesn't show the whole world by default. It can be difficult to get the values for the extent that you want, but luckily with firebug it's easy. Open your Firebug console (Ctrl+Shift+L or Cmd+Shift+L depending on your OS). In the console tab, enter the following:

```
map.getExtent();
map.getCenter();
```

Make a note of the values you get from these commands. To set the map to centre on your new location at startup, add a second argument to the options variable:

```
maxExtent: new OpenLayers.Bounds(-179.722, 33.6053, 179.679, 84.7857)
```

and after the layers load:

```
map.setCenter(new OpenLayers.LonLat(-152, 67), 3);
```

This command also sets the initial zoom level (in this case, 3).

WFS Layers

To use wfs layers in your map you may need to do some extra configuration at the server level. In your cgi-bin folder (where mapserv lives) add a proxy.cgi file (see <http://trac.openlayers.org/wiki/FrequentlyAskedQuestions#ProxyHost>), and add the following to your web page above the map variable:

```
OpenLayers.ProxyHost = "/cgi-bin/proxy.cgi?url=";
```

When this is configured correctly use the same set of parameters as for wms, eg:

```
var wfs = new OpenLayers.Layer.WFS(
  "States",
  "http://localhost/geoserver/wfs",
  {typename: "topp:states"}
);
```

Then tell openlayers to load this layer at startup as well:

```
map.addLayers([wms, wfs]);
```

To overlay vector layers such as wfs on top of commercial layers, you may need to set the projection of each layer in your web page as well as the map itself. See http://crschmidt.net/~crschmidt/spherical_mercator.html for details.

Alternative Mapping Packages

There are alternative mapping packages- for example MapGuide Open Source (<http://mapguide.osgeo.org>). This was originally a commercial product from Autodesk, but development was forked approx 3 years ago into open source and proprietary versions. Development of the open source branch feeds into the development of the proprietary branch.

Binaries and source code are available for all major platforms, but activity is mainly focused on the windows version. Installation on all but a few linux distributions requires compiling from source, which takes approximately 12 hours to work through from start to finish. It has options for using with apache or IIS on windows, and for different programming languages such as php or .net.

The end result is a more sophisticated web application, but one with considerable overhead in terms of setup time and costs- and is only really an option if you have full root access to your web server.

Finally: Cataloguing

Why? Because a catalogue is a much better way of storing your data, particularly once it's all web-enabled.

Geonetwork (<http://geonetwork.open-source.org>) is a "spatial data management system". In short, it allows you to upload, download, find and describe spatial data, belonging to you and others. It has a web-based interface, with a spatial browser and a search interface, and ensures that you fill in at least the bare minimum of metadata to make your data useful to you and others. Basically it's a content management system for spatial data.

It's cross-platform and java-based, and comes with binary installers and source code for all platforms. On windows you install the executable, and start the server from the programme menu. You then access the catalogue via <http://localhost:8080/geonetwork>.

When you upload data, you get the option to choose a particular metadata template, and assign access permissions for the data, ie is it read-only, can certain groups download it or just view it on the web etc. It's important to note that you can use geonetwork to store other types of data, such as videos, and conference proceedings.

Reference Material:

Web Mapping Illustrated (O'Reilly) by Tyler Mitchell ISBN 0-596-00865-1

Mapserver main website: <http://mapserver.gis.umn.edu/>

Mapserver 5 tutorial <http://biometry.gis.umn.edu/tutorial/>

Geoserver website: <http://geoserver.org/>

Openlayers website: <http://openlayers.org/>

Introduction to Openlayers (Workshop, FOSS4G 2008)
<http://workshops.opengeo.org/openlayers/intro/doc/>
OGC Standards (for WMS/WFS etc): <http://www.opengeospatial.org/standards>
Geonetwork website: <http://geonetwork-opensource.org/>

Joanne Cook
Senior IT Support and Development
OADigital/Oxford Archaeology
j.cook@thehumanjourney.ney
+44 (0)1524 880212

This work is licenced under the Creative Commons Attribution-Share Alike 2.0 UK: England & Wales License. To view a copy of this licence, visit <http://creativecommons.org/licenses/by-sa/2.0/uk/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.