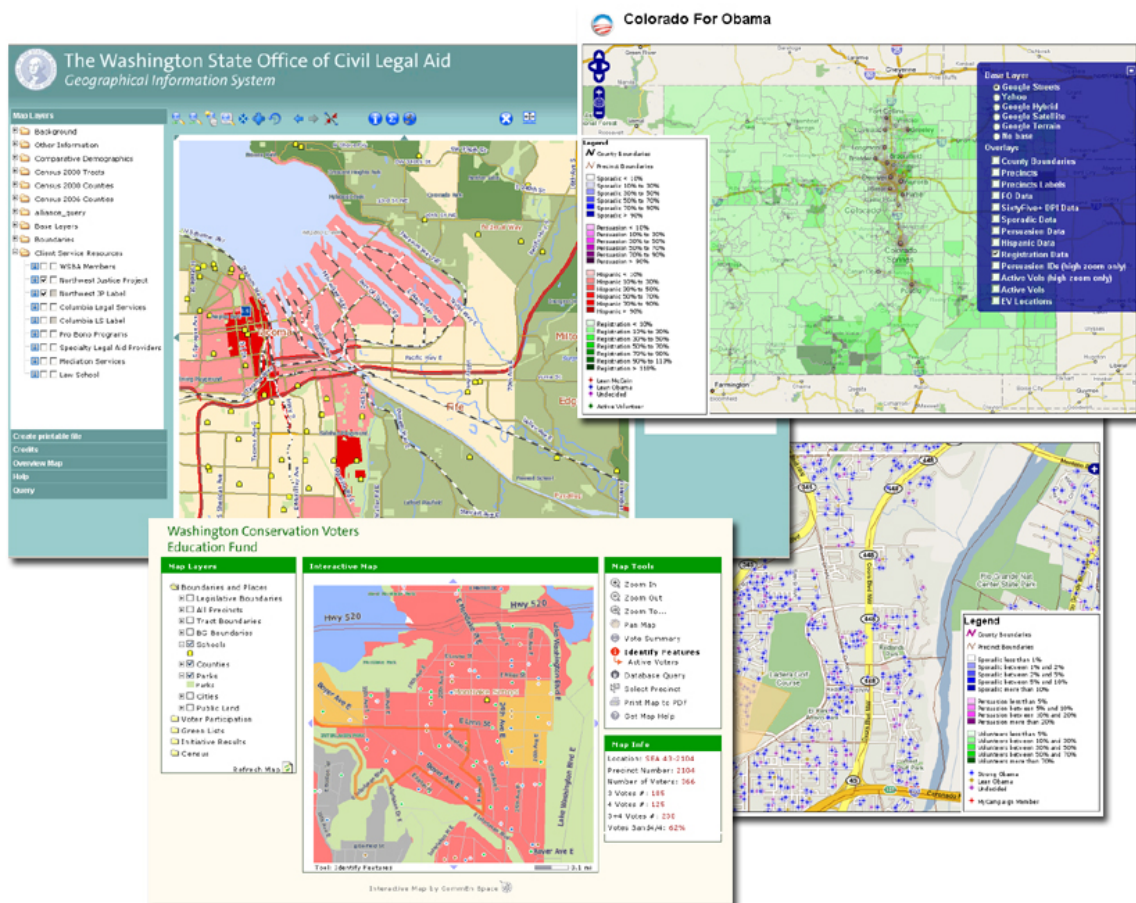


Interoperable Web GIS Solutions with Open Source Software

Introduction for intermediate and advanced GIS users



by Karsten Vennemann, Terra GIS Ltd, Seattle, WA, USA
October 14, 2016



TERRA GIS
TERRESTRIAL ENVIRONMENT REGIONAL ANALYSIS

Terra GIS LTD
2119 Boyer Ave E
Seattle, WA 98112
USA
www.terragis.net

Contents

1	FOSS Overview	1
1.1	What is Open Source (GIS)?	1
1.2	The FOSS Culture	2
1.3	History	3
1.4	licenses	4
2	FOSS Web GIS Components	6
2.1	Web GIS Engines	7
2.1.1	MapServer	7
2.1.2	GeoServer	8
2.1.3	Mapnik	10
2.2	Client Side Frameworks	12
2.2.1	OpenLayers (OL)	12
2.2.2	Leaflet	13
2.3	Client-Server Side Frameworks	14
2.3.1	Mapbender	14
2.3.2	MapFish	15
2.3.3	More Great Frameworks	17
2.4	Extending GIS Capabilities	17
2.4.1	PostGIS - The OS Spatial Database	17
2.4.2	Feature Server	18
2.4.3	TileCache	21
3	Setup of an OS Web GIS Application	22
3.1	Selecting components for a Web GIS	23
3.2	Configuration of a Server for Web Mapping (on Ubuntu Linux)	24
4	Exercises	26
4.1	Using OpenLayers	26
4.1.1	Some Important OpenLayers Objects	26
4.1.2	Commercial Layers in OpenLayers	28
4.1.3	Adding WMS data to your map	29
4.1.4	OpenLayers Exercises	31
4.1.5	OL Examples (version 2.13)	32
4.1.6	Optimizing OpenLayers	32
4.1.7	OpenLayers is also used as	33
4.2	Using MapServer	33
4.2.1	Installing MS4W - MapServer for Windows	35

4.2.2	Configuration of MapServer	37
4.2.3	MapServer Exercises	40
4.2.4	Additional Topics	41
4.2.5	Publishing Web Map Services (WMS) with MapServer	42
4.2.6	WMS Exercises	45
4.2.7	MapServer Performance Tips	46
4.2.8	Tools to work with MapServer	47
4.2.9	Notes about MapServer	49
4.3	Using PostGIS	52
4.3.1	Installing PostGIS	52
4.3.2	Interacting with PostGIS	54
4.3.3	pgAdmin - Administration of PostGIS	56
4.3.4	Utilities for PostGIS	58
4.3.5	PostGIS Exercises	62
4.3.6	Exercises - PostGIS Layers in MapServer	62
4.3.7	Using PostGIS with MapServer	63
4.3.8	Some Notes on PostGIS	64
5	Appendix	66
5.1	OS Desktop GIS Tools	66
5.2	Basic Libraries - Using PROJ, OGR, and GDAL	67
5.3	OS Desktop GIS Programs	70
5.3.1	Geographic Resources Analysis Support System (GRASS)	70
5.3.2	User friendly Desktop Internet GIS (uDig) and JGrass	72
5.3.3	Quantum GIS (QGIS)	72
5.3.4	OpenJUMP - JUMP (JAVA Unified Mapping Platform)	73
5.3.5	gvSIG - Generalitat Valencia Sistema de In- formación Geográfica	75
5.4	General Resources	76
5.4.1	Popular FOSS4G Licenses	76
5.4.2	Books	78
5.4.3	Articles	79
5.4.4	General Web Sites	79
5.4.5	Open source utilities	80
5.4.6	Web Mapping Software	81
5.4.7	Additional tools for use with OpenLayers	81
5.4.8	Map Tiling engines	82
5.4.9	Databases and Additional Software	82
5.4.10	Desktop GIS List	83
5.4.11	GIS Samplers, Live DVDs, and Compilations	83
5.4.12	Conferences	84
5.4.13	User Groups	84

5.4.14	Mailing Lists	84
5.4.15	Internet Relay Chat (IRC)	84
5.4.16	Meeting the Pros	85
5.4.17	Installing a Web GIS Server from the Ubuntu GIS Repository	86
5.4.18	GDAL and OGR Formats	88
5.5	Sample Map files for MapServer	97
5.6	OpenLayers Viewer: Examples OL 2	108
5.7	PAQs: Participant Asked Questions	118

List of Tables

1	The tribes of FOSS4G	2
2	List of common FOSS software licenses	5
3	MapServer Project	8
4	GeoServer Project	9
5	Mapnik Project	10
6	OpenLayers Project	13
7	Mapbender Project	15
8	MapFish Project	16
9	PostGIS Project	18
10	Feature Server Project	19
11	TileCache Project	21
12	MapServer utility programs	48
13	Utility programs for creating MapServer configura- tion files	49
14	OS Geospatial Desktop Programs	71
15	GRASS Project	71
16	uDig Project	72
17	Quantum GIS (QGIS) Project	73
18	OpenJUMP Project	74
19	gvSIG Project	75
20	FOSS4G web sites	79
21	Open source utilities	80
22	Web Mapping Software	81
23	OpenLayers additional tools	81
24	Map Tiling engines	82
25	Databases and Additional Software	82
26	Desktop GIS	83
27	GIS Samplers, Live DVDs, and Compilations	83
28	Free and Open Source Software for Geospatial (FOSS4G)	84
29	Mailing Lists	84
30	OS GIS Blog aggregator	85
31	GDAL Raster Formats	88
32	OGR Vector Formats	94

List of Figures

1	Schematic view of Interoperable Web GIS	6
2	Three maps rendered with Mapnik	11
3	UNHCR web mapping application based on MapFish	16
4	Schematic view of Featureserver operation. Source: http://featureserver.org	20
5	Schematic view of Web GIS Components	23
6	Client-server Architecture for geospatial web services	44
7	pgAdmin - Connecting to a DB Server	56
8	pgAdmin - Administration Tool for PostgreSQL . . .	57

1 FOSS Overview

1.1 What is Open Source (GIS)?

Open source¹ means that the source code is available to the general public for use, distribution, and modification from its original design free of charge (among a long list of other requirements). Other acronyms frequently used in the OS GIS world are FOSS² and FOSS4G³. Free in this context is meant to express that the software is free (like in freedom or liberty) and free of charge⁴. While most open source geospatial software is built on the standards of the Open Geospatial Consortium (OGC) the term "Open Source" is not synonymous with Open Standards because both proprietary and open source software can be compliant with the OGC Open Standards.

<http://www.opengeospatial.org>

The OGC is a consortium of proprietary companies, scientific organizations, government agencies and representatives of the open source software movement that develops and publishes software interface specifications (OGC standards).

OSGeo is the organization that supports the development of the highest quality open source geospatial software

<http://www.osgeo.org>.

Proprietary and open source software can be compliant with OGC standards. However, adoption of OGC standards is usually higher among OS software packages. One characteristic of OS software is that it extensively uses synergies in software development. Libraries are shared to a big extent and the high adoption level of OS software makes them highly suitable for interoperable software solutions. However, Ramsey (2007)⁵ listed the following *tribes* (referring to their communal use of software programming languages or the medium, like the web

Open Source is not synonymous with Open Standards

OGC sets standards



OSGeo supports the development of open source geospatial software



The Tribes Concept

¹in the following referred to simply as OS

²Free and Open Source software

³Free and Open Source software for Geospatial (Applications)

⁴Due to a short coming of the English language which has no word for it, it is also sometimes referred to as *free as in beer* (gratis)

⁵The State of Open Source GIS, Version September 2007

for which the packages are primarily designed.

Table 1: The tribes of FOSS4G

Tribe	FOSS4G Projects
C/C++	MapServer, GRASS, MapGuide, QGIS, PostGIS, OGR/GDAL, PROJ4, GEOS, FDO
Java	GeoTools, GeoServer, uDig, DeeGree, JUMP, gvSIG
Web	MapBender, OpenLayers
.Net	SharpMap, WorldWind, MapWindow

Reasons to use Open Source software

user control	User is in control <ul style="list-style-type: none">• Pick your favorite operating system: Windows-Linux-Solaris-Mac• No licensing issues (did we install one to many PCs with software XY?)• Vendor independency• Access to source code: don't like something, need changes to the core system, need extensions - hire somebody to change it right now
performance quality interoperability	High performance, high quality, high interoperability <ul style="list-style-type: none">• distributed programming effort, highly modular• System heterogeneity - less prone to hacker attacks and viruses. Highly interoperable, very advanced support of OGC open standards
support	Support - Commercial and non commercial <ul style="list-style-type: none">• Mailing lists, user groups, Conferences, IRC channels• Fast response times for bug fixes• typically tracked on the web, accessible and open to everybody to report or fix a bug
free	Last but not least - It is free

1.2 The FOSS Culture

give take contribute	Often the FOSS movement is referred to as not only a model on how to create, distribute and license software but rather a culture. A lot of times business people don't understand
-------------------------------------	--

why one would create something useful and just give it away instead of selling it. Thus, many times they infer that there must be a catch, something must be wrong with the product, since it is free it must have no value and other misconceptions.

There is much more to it than producing free and open software. It is a way of doing things, of working together, of collaborating, a movement of people around the globe, in short a culture. It is appreciated when people using the software are giving something back to the community. That might be helping others in the user list and online forums, writing documentation about something you learned about using the software in the online wiki pages⁶ of the project, writing new source code or customizations and sharing it with the community. The community is working like an organism and the organism does better if all parts are working together.

contribute
•**source code**
•**documentation**
•**answer questions on mailing lists**
•**support and fund OSGeo**

1.3 History

Before the 1960's it is safe to say that most software was free (*libre*). Only in the late sixties and seventies proprietary software was *born* and quickly dominated the software landscape. In 1983 Richard Stallman, a former programmer at the MIT laboratories founded the Free software Foundation and the GNU Project. One of the major goals of GNU was to create a free operating system and to establish licensing terms for FOSS.

1960's
most software is free
1983
Free software Foundation and the GNU Project

A few of the milestones for general OS software development where the first release of the Linux operating system in 1991, the release of the Apache HTTP server in 1995, and the increasing involvement of software heavy weights such as IBM (Eclipse programming platform) and SUN Microsystems (Java programming language and tools) since the late 1990's.

1991
Linux
1995
Apache HTTP server
Eclipse programming platform

In 1998 the Open Source Initiative a, non-profit corporation, was created with the goal to educate about and advocate for the benefits of open source software. On their website <http://www.opensource.org> they state:

1998
Open Source Initiative founded

"Open source is a development method for software that harnesses the power of distributed peer review and transparency of process. The promise of open source is better quality, higher reliability, more flexibility, lower cost, and an end to predatory vendor lock-in. The Open Source Initiative

⁶a web page/collection of pages that enable anyone to contribute, modify and share content

(OSI) is a non-profit corporation formed to educate about and advocate for the benefits of open source and to build bridges among different constituencies in the open-source community. One of our most important activities is as a standards body, maintaining the Open Source Definition for the good of the community. The Open Source Initiative Approved License trademark and program creates a nexus of trust around which developers, users, corporations and governments can organize open-source cooperation."

1.4 licenses

FOSS4G licensing general terms

A variety of licensing types exists for Free and Open Source software. Some of them are more restrictive than others. According to the current OSGeo president Arnulf Christl's talk at FOSS4G 2008, most of the licenses specify the user's rights to

- run any number of copies of the software
- pass on copies of the software including the license and copyright notice
- look into the code, understand it and modify it to suit one's needs
- pass on modified versions of the code

What is copyleft?

From a word play with the term *copyright* resulted the newly coined term *copyleft* which similarly quotes the following freedoms for users to use and study the work, copy and share the work with others, modify the work, and to distribute modified and therefore derivative works. The idea of copyleft is that work derived from the original OS software will have to be distributed under the same liberal or equivalent license terms.

Common FOSS software Licenses

GNU License



The most popular OS software licenses for geospatial applications are listed below. Brief descriptions for each of these licenses (quoted from Wikipedia) are included in the resource section of this booklet.

Table 2: List of common FOSS software licenses

Name	Style	software
GNU-GPL	strong copyleft license, derived works have to be available under the same copyleft	GRASS, QGIS, gvSIG, Mapbender, PostGIS, GeoServer, AveiN!
LPGL	compromise between copyleft and more permissive licenses, has copyleft restrictions on the program itself, but not on other software linking with the program.	Mapnik, MapGuide
MIT	permissive license, permits reuse within proprietary software (license has to be distributed with that software)	MapServer, GDAL/OGR, Proj4
BSD	permissive license, little restriction, close to the public domain	FeatureServer, Tile-Cache, OpenLayers
Mozilla (MPL)	hybrid of modified BSD and GPL.	MapWindow, Mozilla Firefox

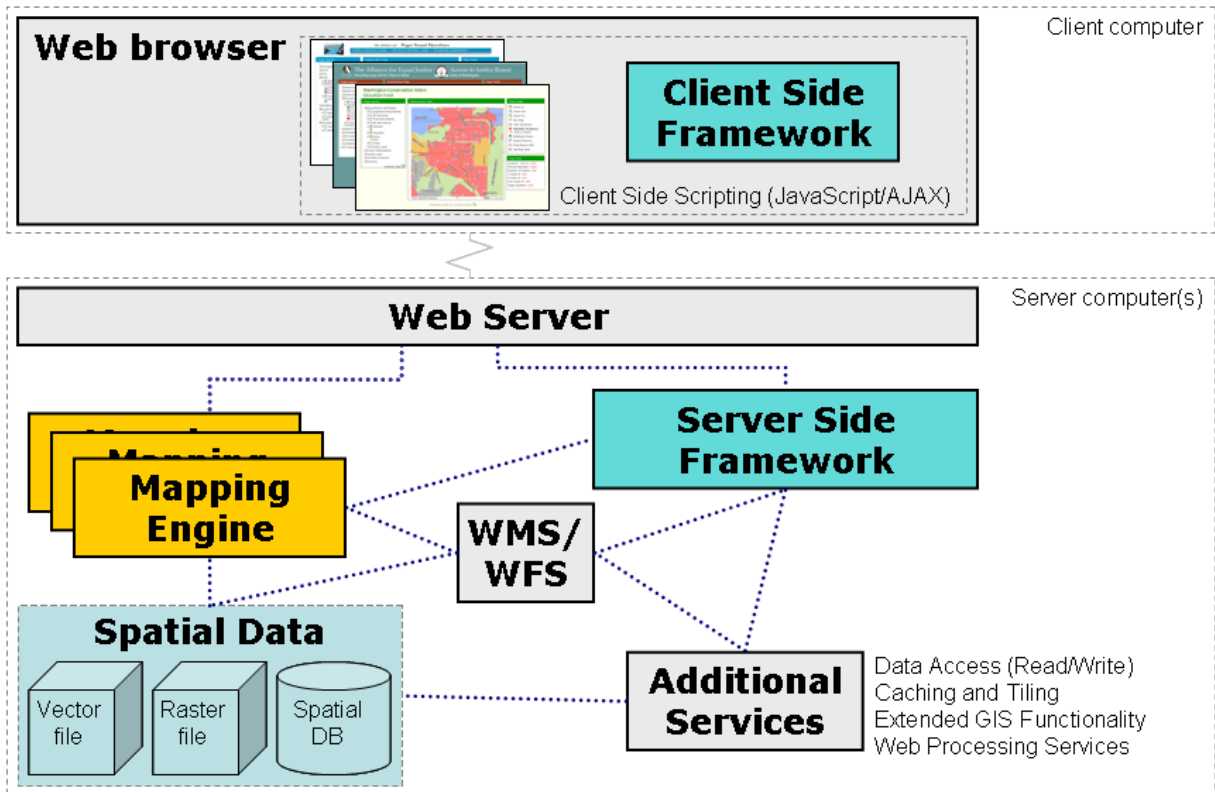


Figure 1: Schematic view of Interoperable Web GIS

2 FOSS Web GIS Components

Web GIS frameworks can either be client side based only or include a server and a client side. One pretty common client program is a modern web browser that has scripting enabled (JavaScript) and thus can run some part / or most of the application on the client computer. If an application does most of the processing on the client side it is called a thick client. Examples for thick clients are Google Earth, ArcGIS, or uDig.

If most of the processing takes place on the server and only the images are sent over to the client or are requested by the client this is called a thin client e.g. OpenLayers or Google Maps. In a simple case where all the application components are hosted on one server machine this includes at least the Spatial data such as vector and raster files and/or a spatial database. These data can then be rendered as images by

thick client
e.g.
Google Earth
ArcGIS
uDig
thin client e.g.
Openlayers
Google Maps

a Mapping engine (e.g. MapServer, GeoServer, Mapnik or ArcIMS) and served via the HTTP web server (e.g. Apache or MS IIS) to the web. Mapping engines can be used to publish as web mapping services (WMS) or web features services to the web (WFS). The server side can also be complemented by a framework or content management system that provides for functionalities such as user authentication, user access rights management, user interface customization, tools and functionality management and more. Such an interoperable web GIS system can be extended by additional server side components that provide functionality such as caching and tiling, GIS analysis functionality (e.g. via PostGIS intersections and reprojection) or web processing services (WPS).

WMS
web mapping
service
WFS
web feature
service

2.1 Web GIS Engines

2.1.1 MapServer

MapServer is a map rendering engine that was developed at the University of Minnesota (UMN), short MapServer. It is one of the most mature and most successful open source GIS projects and is implemented in C. The main focus is on rendering spatial data and in providing a development environment for spatially-enabled Internet applications. Some of the highlights about MapServer are that it supports more input data sources than most of the proprietary products, has higher performance, and the pre-compiled versions are simpler to install and set up. There are two main ways that one can interact with MapServer using it as a CGI¹ application or using one of several MapScript APIs to write your custom application. The MapScript API² is available for PHP, Python, Perl, Ruby, Java, and C#. MapServer uses a configuration text file that commonly has the .map extension. This file tells MapServer all the information the programs needs to know to render a map. The configuration includes general information about the maps like projection, size of the map in pixels, output format etc. Individual layer tags are used to determine the classification of a layer. Examples for input formats are MapInfo files, shapefiles, PostGIS, Oracle Spatial, ArcSDE, WMS, and all GDAL and OGR formats. Output formats include GIF, JPG, PNG, all GDAL formats, WFS and WMS.



MapServer has
two modes
•CGI mode or
•MapScript
mode
MapScript is
available in
PHP
Python
Perl
Ruby
Java
C#

¹Common Gateway Interface is a standard protocol for interfacing external application software with web servers.

²An application programming interface (API) is a set of functions, procedures, methods, classes or protocols.

MapServer suite

With the release of MapServer version 6.2 its source code and several related projects were bundled to form the MapServer suite. The suite includes MapServer CGI/FCGI, MapScript, Tiny OWS and MapCache. Other useful related tools are the MapServer utilities listed on page 48. Tiny OWS is an implementation of Transactional Web Feature Service (WFS-T), a specification of the OGC that has been added to the suite as a module with the release of MapServer version 6.2.

Table 3: MapServer Project

Main supporter of MapServer Core Team, Steven Lime	
MapServer	
Functionality	Map rendering engine, web mapping development environment
Operating systems	Multi platform
Project started	1996
Implementation	C
OS libraries	OGR/GDAL
PostGIS support	Yes
License	BSD

2.1.2 GeoServer

GeoServer rendering and publishing data editing data WFS-T versioning



GeoServer is a newer development than MapServer. It covers much of the functionality that MapServer does, such as rendering images from spatial data, and publishing of WMS and WFS. It has a web-based graphical configuration and management interface and stores the configuration in XML format.

Furthermore it supports transactional capabilities and shared editing (versioning) of spatial data. GeoServer also supports KML and tiled map output. Input formats include PostGIS, shapefile, ArcSDE, DB2, Oracle (soon VPF, MySQL, MapInfo, and WFS). Output formats include JPG, GIF, PNG, SVG, KM-L/KMZ, GML, shapefile, GeoJSON, GeoRSS, WFS and WMS. GeoServer can currently serve WFS on top of: Oracle Spatial, ArcSDE, PostGIS and shapefiles.

Table 4: GeoServer Project

Main supporter of The Open Planning Project (TOPP) GeoServer	
Functionality	Map rendering engine, support for web based editing of data, transactions WFS-T and versioning
Operating systems	Multi platform
Project started	2001
Implementation	Java, J2EE
OS libraries	Geotools
PostGIS support	Yes
License	GPL 2.0

2.1.3 Mapnik

**Mapnik
rendering high
quality maps**



**Map output
can be via
Python scripts
and
style sheets**

Mapnik is another map rendering engine which was started by Artem Pavlenko in 2005. It appears to have a lot of potential, even though the developer and user community is still small. Mapnik is written in C++ and already delivers really nice map output based on the AGG renderer³. Mapnik can be used to produce maps interactively on a web server or on the desktop environment. Map output can be controlled via Python scripts and style sheets. Currently the focus is on map rendering and it does not have capabilities for editing or querying spatial data. Mapnik accepts shapefiles, PostGIS and TIFF raster as input formats, and uses a styles processor to output images in PNG, JPG, SVG, PDF, and PS formats. A tutorial for making maps with Mapnik and Python can be downloaded from <http://code.google.com/p/mapnik-utils/>.

Table 5: Mapnik Project

Main supporter of Mapnik	Artem Pavlenko and others
Functionality	Map rendering engine (based on AGG), Python bindings
Operating systems	Windows, Linux, Mac
Project started	2005?
Implementation	C++
OS libraries	Boost libraries, AGG renderer http://www.antigrain.com
PostGIS support	Yes
License	LGPL

³Anti Grain Geometry renderer. An OS software graphic library, written in C++ that allows to render images from vector data.



Figure 2: Three maps rendered with Mapnik

2.2 Client Side Frameworks

The following frameworks are based on JavaScript/Ajax Libraries, have no (or optional) server side dependencies and thus can be used easily on a shared hosting environment.

2.2.1 OpenLayers (OL)

OpenLayers Geography in a browser



**smooth map
scrolling and
browsing
experience
(slippy map)**

integration
OpenLayers and
EXT JS

**included in
a web page
via JavaScript
<script> tag
reference**

OpenLayers is a pure client side mapping framework based on JavaScript and as a result is very easy to deploy. OpenLayers has no server side dependencies and provides an API that you can use independently from the data sources that supply the input data for the application. Input data sources can be tiled and pre-rendered images via WMS or commercial layers such as Bing, World Wind, Google Maps, or WMS services. Using map tiles enables a smooth map scrolling and browsing experience (slippy map) just like the commercial maps from Google, and Microsoft. Vector input for OL can come from many sources such as MapServer, GeoRSS, WFS, or KML. OpenLayers supports a variety of reusable components such as scale bars, zoom bar, layer switcher (on/off), and tools like zoom in/out and pan. Version 3 of OpenLayers (OL3) was released on August 29th 2014 and is a full re-write using modern design patterns. OL3 breaks backwards compatibility with the 2.X versions of OL. OL3 is based on Google's Closure Tools which enable advanced compression of the library (resulting in smaller size) and ensures wide cross-browser testing. Custom builds of OL3 can be easily compiled for production software using the closure compiler.

GeoEXT <http://www.geoext.org/> is a set of spatial tools extending the **Ext JS** JavaScript library and integrating them with OpenLayers components. It supplies an API to embed OpenLayers components such as a map or a legend in Ext panels and enables easy use of Ext components within the web application.

One easy way to include an OL maps in a web page is to simply add a JavaScript <script> tag reference to a copy of the OpenLayers .js file and one can even reference a hosted version of the OpenLayers library at <http://openlayers.org/api/OpenLayers.js>. OL also has tools enabling digitizing and editing ⁴of data when used in conjunction with a WFS-T⁵ enabled server like GeoServer. OL has close ties to TileCache, and Feature Server which are also projects from Metacarta Labs.

⁴e.g. OpenLayers Editor (geops) <http://ole.geops.de/>

⁵Transactional Web Feature Services support editing of vector data.

Table 6: OpenLayers Project

| | |
|---------------------------------------|---|
| Main supporter of OpenLayers 2 | v2 OpenLayers Dev Team |
| Main supporter of OL3 | v3 OpenLayers Dev Team |
| Functionality | Client side mapping framework to develop interactive mapping applications. OL3 better optimized for use on mobile devices |
| Operating systems | Windows, Linux, Mac |
| Project started | 2005, OL3 August 2014 |
| Implementation | JavaScript/Ajax |
| libraries v2: | Prototype.js and Rico library |
| libraries v3: | Google's Closure Tools (compiler+library) |
| PostGIS support | via rendering engine |
| License | BSD |

2.2.2 Leaflet

Leaflet is a lightweight [JavaScript library](#) for creating client side web maps that works efficiently on desktop and mobile platforms. Weighing in with only about 33KB size it is a small library that has many functions but is lacking the more advanced 'GIS' features that especially OpenLayers 2.X (and OL 3) caters to. It is a good choice if (download) speed is the main objective, compatibility with mobile devices is important and if more advanced features are not needed in the desired web-map. Leaflet's functionality can be extended via a large collection number of [plugins](#).

Leaflet also has notably been used in conjunction with the popular [D3 \(Data-Driven Documents\) library](#). D3 is a JavaScript library supporting many kinds of visualizations (such as charts) that are based on a combination with HTML, SVG and CSS. Mapping capabilities have been added to D3 over time, making it an interesting alternative for *in browser visualization* of geographic and non geographic objects. The [TopoJSON](#) format is an extension of GeoJSON that encodes topology of vector data for effective display in a browser. The latter format

**a lightweight
JS Library**



**D3 Java Script
library
visualization
using SVG**

D3, GeoJSON and TopoJSON

uses topology and therefore is eliminating redundancy in data (e.g. this means no duplicate adjacent boundaries but one shared boundary). This can result in much smaller vector file sizes (of geographic objects), that can be up to 80% smaller than their [GeoJSON](#) equivalents. D3 and Leaflet can be combined (TopoJSON is available in Leaflet via D3) as shown in this simple [example](#). A large collection of examples shows the capabilities of the growing world of JavaScript [Web maps and other interesting D3 Visualizations](#).

Main supporter of Vladimir Agafonkin and [others](#)
Leaflet

| | |
|-------------------|---|
| Functionality | Client side mapping framework with focus on small library size and streamlined, mobile friendly functionality |
| Operating systems | Windows, Linux, Mac, Mobile devices |
| Project started | 2011 |
| Implementation | JavaScript/Ajax |
| OS libraries | none |
| PostGIS support | via rendering engine |
| License | BSD |

2.3 Client-Server Side Frameworks

2.3.1 Mapbender

Mapbender
Mapbender

web based
administration
interface

security for
WMS and WFS
via OWS proxy
functionality

Mapbender is a comprehensive client and server side mapping framework. It is implemented in PHP, JavaScript and XML and based on an administration database that can be housed in either MySQL or PostgreSQL. Some of its functionality such as measuring distances on a map is only available if PostgreSQL/-PostGIS is used as an admin database. It features a web based administration interface to manage applications, application layout, user and group access rights, mapping services, available tool functionality and security for WMS and WFS via OWS proxy functionality. Functionality for each of the housed web mapping applications include displaying, navigating, editing and querying spatial data and maps. Input sources are solely OGC WMS compliant map services and WFS. In the near future it is envisioned to embed OpenLayers as a map viewing option

in addition to the intrinsic Mapbender viewer.

Table 7: Mapbender Project

| | |
|------------------------------------|---|
| Main supporter of Mapbender | WhereGroup |
| Functionality | Client side and server side mapping framework, based on a database (MySQL and PostgreSQL) |
| Operating systems | Windows and Linux |
| Project started | 2001 |
| Implementation based on | JavaScript/Ajax, DHTML, PHP
MySQL or PostgreSQL |
| PostGIS support | Via rendering engine |
| License | GPL |

2.3.2 MapFish

MapFish is a client-server web mapping framework. MapFish provides tools for creating web services that allows querying and editing geographic objects. It also provides a JavaScript toolbox including specific components for interacting with MapFish web services. It consists of two parts: MapFish server and MapFish client. MapFish Client is a JavaScript framework based on OpenLayers, Ext JS and GeoExt. MapFish Server is a Python framework (based on Pylons) that extends Pylons with geospatial functionality. There are two other implementations currently available: a Ruby/Rails plug-in and PHP/Symfony plug-in. Server side functionality is implemented as a Widgets and plug-ins oriented architecture. An administration tool, **MapFish Studio** can help to simplify the configuration work and will allows to create new MapFish applications. User Interface (UI) components include: layer tree, authentication, geostatistics, query, routing, and editing (the last five are server side widgets). Features include offline support (based on Google Gears), and PDF printing (features printed as vectors, rotated maps, scale bar, legends, tables).

MapFish



Adding server side framework to OpenLayers administration tool MapFish Studio

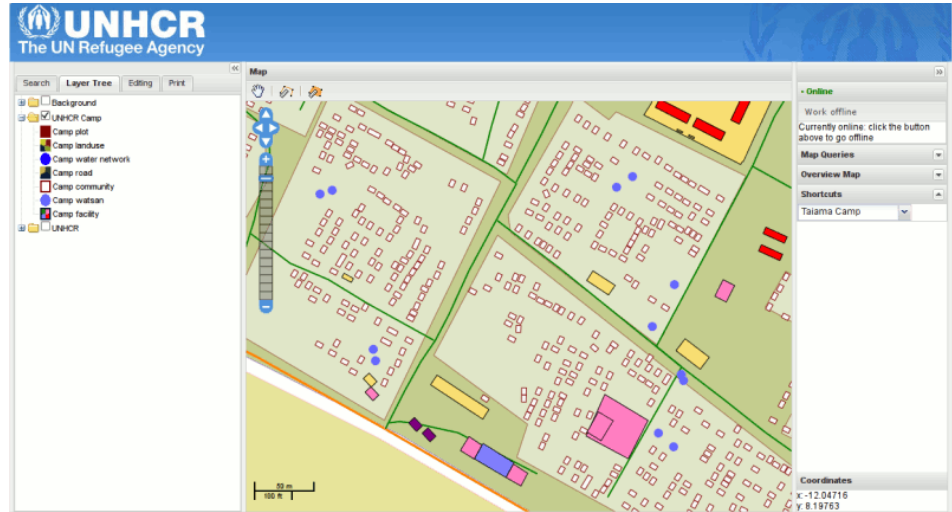


Figure 3: UNHCR web mapping application based on MapFish

Table 8: MapFish Project

| | |
|----------------------------------|--|
| Main supporter of MapFish | Campocamp |
| Functionality | Client side and server side mapping framework, client side viewer is based on OL, supplies server side widgets, offers MapFish API |
| Main Focus | Adding server side framework to OpenLayers |
| Operating systems | Windows and Linux |
| Project started | 2007? |
| Implementation | JavaScript/Ajax, DHTML, Python |
| OS libraries | Pylons, SQLAlchemy, GeoJSON, Shapely, JTS, and Ext JS |
| PostGIS support | Via rendering engine |
| License | GPL 3.0 |

2.3.3 More Great Frameworks

[Django](#) is a web development framework written in Python. It supports multiple database back-ends (PostgreSQL, MySQL, SQLite, Oracle, and MS SQL) and bundles a number of applications. Several years ago *GeoDjango* was merged into the Django main source code allowing it to be used as a web development framework for creating web based GIS applications.

2.4 Extending GIS Capabilities

Spatial data storage solutions and additional tools for web mapping applications.

2.4.1 PostGIS - The OS Spatial Database

PostGIS is an extension for PostgreSQL and adds support for geographic objects to PostgreSQL. It enables PostgreSQL server to be used as a backend spatial database for GIS. The functionality includes data import/export, storage and retrieval of spatial geometries, and spatial indexing (R-tree) of spatial objects in PostgreSQL by adding functions, casts, and storage types. It enables spatial operations and analysis by simple means of running (spatial) SQL queries in the database. In short PostGIS is a GIS in its own right and can greatly enhance web GIS applications with additional functionality. In a sense it provides similar functions as ArcSDE but operates differently. It is not a middleware sitting on top of a database, rather it is an internal extension of the database written in C. There is no difference in the use of spatial and non-spatial objects in PostGIS and all objects are accessible via SQL queries to a GIS application.

When the first version of PostGIS was released in 2001 only MapServer accepted PostGIS as an input format, however over time it has become the standard spatial database backend for all the other open source GIS tools. ArcGIS 9.3 has read and write support for PostGIS but requires an ArcSDE license to connect. Prior to version 9.3 ArcGIS was lacking native support for PostGIS. ArcGIS version 10 and higher can connect to PostGIS. For editing this still looks a little different. In order

**PostGIS
the coolest
thing GIS ever**



**one major
difference to
ArcSDE is that
PostGIS is not
a middleware
sitting on top
of a database
instead it is
an internal
extension of
the database.**

**ArcGIS 9.3 has
PostGIS
support but re-
quires
ArcSDE**

Table 9: PostGIS Project

| | |
|----------------------------------|--|
| Main supporter of PostGIS | Refractions Research, Victoria, Canada |
| Type | Spatial database. PostGIS is an extension for PostgreSQL |
| Functionality | Storage and retrieval of spatial data (geometries such as point, line, polygon, multipoint, multiline, multipolygon, geometrycollection). Spatial indexing. GIS functions via spatially enabled SQL. E.g. intersections, distance calculations, reprojection |
| Operating systems | Linux, Windows |
| Project started | 2001 |
| Implementation | C |
| OS libraries | GEOS, Proj4, OGR, and FDO |
| License | GPL |

to edit PostGIS feature in ArcGIS 10.1 (without ArcSDE) one will still need a middleware called *Spatial Data Server* (SDS) that comes with ArcGIS Server - [see this blog entry](#). There exists also a third party extension (which used to be free) [ST-Links SpatialKit](#) exists that enables display and editing of PostGIS data in ArcMap 10, 10.1 and 10.2 (version 4) and version 3.0.4 works with ArcMap 9.3 without any middleware. Another third party ArcGIS extension *zigGIS* used to be available from *Obtusesoft* but has been discontinued in 2011. In addition the *Data Interoperability Extension* also enables earlier versions of ArcGIS to use PostGIS layers.

2.4.2 Feature Server

server software for publishing, aggregating and converting geospatial data

FeatureServer is a server software (a middleware) for publishing, aggregating and converting geospatial data in a variety of different formats for the web. Published data are made available as a common web services interface. FeatureServer is implemented as a RESTful⁶ Geographic Feature Service. It uses standard HTTP commands (GET, PUT, POST and DELETE) to control operations and enables dynamic capabilities to read geographic features or feature collections. Input can come from

⁶REpresentational State Transfer (REST) - style of software architecture that can be used for the web

distributed sources. It can be used to translate geographic features between formats. For example a user can input an ESRI shapefile and open it in Google Earth. FeatureServer enables feature editing and server-side storage of spatial data using OpenLayers. The following data sources (storage) are supported: DBM, BerkleyDB, PostGIS, WFS, OGR, Flickr, and OSM (Open Street Map). Service input/output formats are GeoJSON (points, lines, and polygons with rings), GeoRSS Atom (Simple), KML, GML/WFS (output-only), HTML, and OSM (output-only as Open Street Map *.osm files). Four different server operation modes are available: CGI, mod_Python, Standalone wsgi HTTP server, and FastCGI.

Table 10: Feature Server Project

| | |
|---|---|
| Main supporter of Feature-Server | None (formerly it was Metacarta Labs) |
| Functionality | Server software for converting, tiling and caching of geospatial data |
| Operating systems | Windows, Linux, Mac |
| Project started | 2007? |
| Implementation | Python |
| License | BSD |

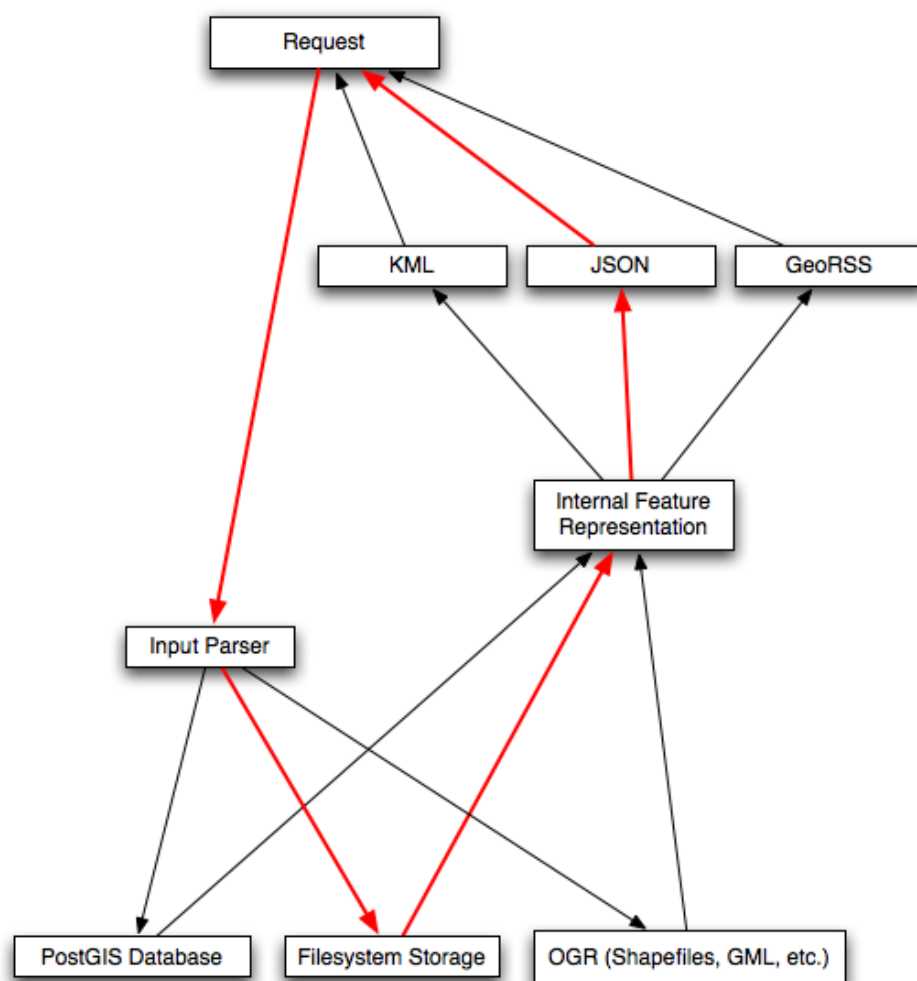


Figure 4: Schematic view of Featureserver operation. Source: <http://featureserver.org>

2.4.3 TileCache

TileCache is a server software solution with caching and rendering capabilities to create your own local disk-based cache of a WMS server. The tiles can be used in a variety of clients e.g. OpenLayers, Leaflet, Google Maps, MS Bing, and WorldKit. **GeoWebCache** is a similar product but is written in Java (and is part of the [OpenGeoSuite](#)). TileCache sits in front of map rendering engines such as MapServer, GeoServer, Mapnik or ArcGIS Server, and converts requests from front end viewers (such as OpenLayers or World Wind) into requests for maps (more exactly requests for a collection of tiles that make up a map). By caching tiles on disk the performance of a web based GIS viewer such as OpenLayers can be greatly increased. Tiles are pre-rendered and there is no need for a mapping engine to create a new image at the time of a new map request. Tile size and discrete zoom levels can be configured. Tiles can either be pre-rendered for all geographic areas (tile seeding) or only be created when requested. The use of tiles (generated by TileCache) enables smooth map scrolling experiences in applications (*slippy map feeling*) similar to those of the commercial products such as Google Maps, Bing, and MapQuest Maps. Other software packages with similar functionalities are MapProxy, MapCache (part of the MapServer suite), and TileMill, (a tiling tool for non GIS experts) - see page 82. In order to set up the MapCache module of the MapServer suite in your ms4w installation you will need to add some configuration documented [here](#).

creating image tiles from vector data caching on disk

Table 11: TileCache Project

| | |
|------------------------------------|---|
| Main supporter of TileCache | Metacarta Labs |
| Functionality | Server middleware for converting, tiling and caching of geospatial data |
| Operating systems | Windows, Linux, Mac |
| Project started | 2007? |
| Implementation | Python |
| License | BSD |

3 Setup of an OS Web GIS Application

Server hosting options

- dedicated server
- virtual cloud
- shared hosting environment

There is a variety of different options for building an OS web mapping application.

Short Overview of Server Hosting Options

Depending on the goals and needs for such an undertaking one has to consider what kind of server will be available for hosting the application. For example:

- in house server with administrative access rights
- dedicated server (third party hosted machine) with root access
- shared hosting environment without root access (no administrator rights)
- hosting on a virtual cloud of computers such as Amazon EC2 or others

How to setup a GIS on a shared hosting environment

Dedicated servers can either be virtual servers (multiple virtual servers sharing one physical server computer) or real physical machines. While it is possible to install a web GIS application on a shared hosting environment this will not be the most common use case. For those interested in installing a GIS web mapping application on such an environment read on Aaron Racicot's Blog REPROJECTED [GIS on a shared hosting environment](#). Hosting of a web GIS application on a virtual cloud of computers (such as Amazon EC2 or others) can be a good choice for certain use cases. Those environments allow for preparing images on a complete Linux server workstation installation (including GIS software stacks). Thus, those environments can be a good choice for setting up temporary servers. They can also serve as a resource for GIS analysis that has to be performed quickly. Because of pre-configured server images it is possible to start/-boot multiple machines/servers simultaneously within minutes - with all the required software installed. Those solutions are highly scalable and are available for a relatively low cost, even if you run multiple identical server machines simultaneously over a short time period.

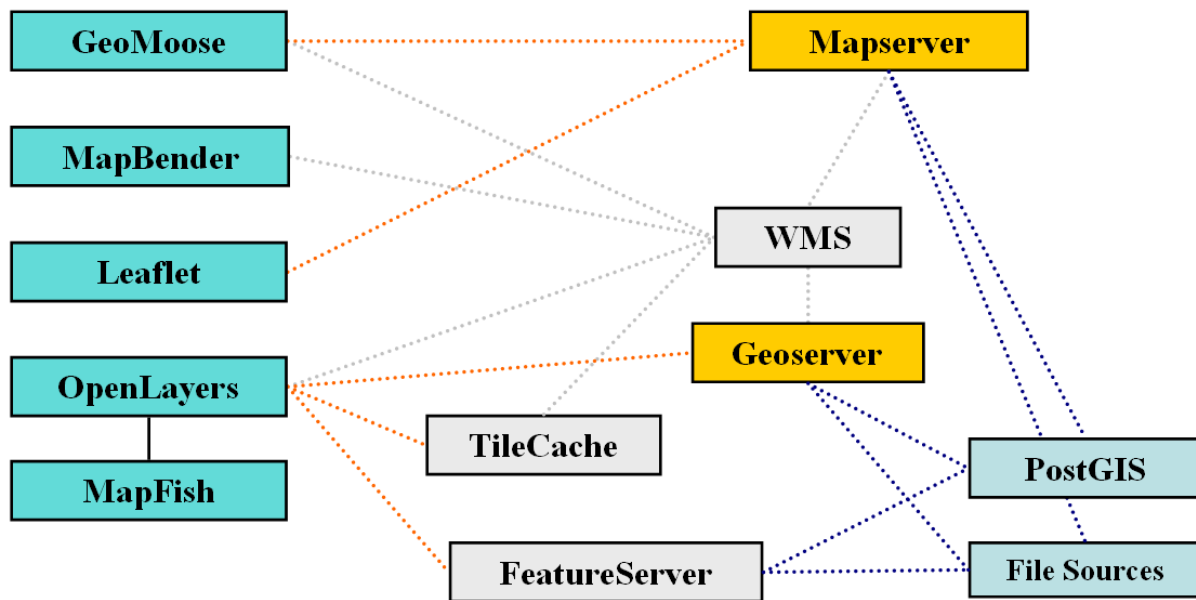


Figure 5: Schematic view of Web GIS Components

3.1 Selecting components for a Web GIS

While this can be an involved and difficult process, there is a couple of basic choices to be made. Figure 5 shows the main building blocks of an interoperable web GIS solution. The data sources (in light blue) are the first building block. Data can be stored in file format or in databases. The second building block is a map rendering engine (shown as orange boxes) that can take in the data and produce some kind of map output (image or vector), which can then be served to the web either as a complete map or as a service, such as WMS or WFS (gray box) to be used in another application on the client side. Capabilities of map rendering engines can be similar to some degree, but many offer differing functionalities.

GeoServer would be the engine of choice if we need editing capabilities or need support for shared editing (versioning). MapServer might be the choice if speed is the main concern and you need very high performance to render images. Some kind of mapping framework can complement the application and make the development, management and deployment of the web GIS easier. On the framework side we have to choose if we want a client side framework only (e.g. because we want to use it on a shared hosting environment - we could use OpenLayers for that), or if we have some type of dedicated server available and we need to build a complex web application. In that case we would e.g. choose Mapbender

interoperable web GIS

- Data
- Map rendering engine
- Mapping framework
- Map services

GeoServer has editing and data versioning support

MapServer is fast Framework client side or client-server side ?

skill set of the project team

or MapFish-OpenLayers. Which one of the frameworks we will choose depends also on the knowledge of the person (or team) working on the project (e.g. programming and scripting skills). If several of the frameworks offer the functionality needed for the project then the choice may well depend on the existing skill set of the project team. Thus, if I am already proficient with Python I might pick MapFish as my framework of choice, whereas when I am most proficient with PHP I could go with Mapbender.

TileCache and FeatureServer add tiling and caching data aggregation and translation PostGIS adds GIS capabilities

Of course the framework has to enable the functionality needed for the project - or be closer to the needed functionality if new capabilities need to be developed. Many frameworks also support multiple sources of data and data intake from multiple map rendering engines to be used in the same applications. Mapbender for example can take in and manage any WMS service no matter whether it is served by GeoServer, Mapnik, MapGuide, ArcIMS, ArcGis Server, or MapServer. So the advantages and strength of several map rendering engines can be used in one framework for the greatest benefit and flexibility in the application. Additional functionalities such as the production and the caching of map image tiles, editing, aggregating, and translating of spatial data can be added by using TileCache and FeatureServer respectively. Furthermore, custom functionalities can be added using the APIs of the frameworks and the map rendering engines. PostGIS has the potential to supply GIS analysis functions to any of the frameworks. For more complicated tasks, even desktop programs such as GRASS and R (Statistical software), can be linked to the web application and potentially enable the development of a full featured web based GIS application.

3.2 Configuration of a Server for Web Mapping (on Ubuntu Linux)

compile software from source is one option

When setting up a web server based on Linux operating systems one can naturally install your software components from scratch, including having to compile from source . This can be good choice when you want to have the most up to date software or a specific software version. However, there are other options that often will be more elegant and time efficient. Using pre-compiled components for your operating system (and version) can save you a great deal of time and potential compilation trouble. The UbuntuGIS project provides easy access to relatively new software versions for many FOSS4G software

projects. The pre-compiled packages can be installed on your Ubuntu machine using the **apt** package manager. To use apt to install OS GIS software components successfully on your Ubuntu machine there are following prerequisites

pre-compiled packages are a viable alternative

- the apt package manager needs to be installed (usually included with your Ubuntu operating system)
- the software package location has to be included in the list of repositories ¹ (this is needed for the package manager to establish access to the software packages download)

Below are examples how GIS components can be installed from the bash command line using apt. The system will prompt you when additional software libraries (dependencies) will need to be installed in order to make the desired component operational.

```
# install gdal
sudo apt-get install gdal-bin
```

```
# install proj
sudo apt-get install proj
```

On **page 86** there is an example of how to install components using the Ubuntu and **Ubuntu GIS** repositories.

example how to install Ubuntu GIS

¹e.g. <http://trac.osgeo.org/ubuntugis/wiki/UbuntuGISRepository>



4 Exercises



4.1 Using OpenLayers

Use Firefox with Firebug for web development

When doing web development some tools come in handy to debug and verify JavaScript, review requests to the server and more. Using the Firefox browser an excellent tool Firebug is available for this purpose <http://getfirebug.com/downloads>. Firebug makes it much easier to verify CSS style sheets, the DOM in web pages, review requests made to the server, debugging JavaScript and more. This can be very useful when working with OpenLayers (OL) because it is a JavaScript library and API to track down errors during coding. For example we can review the HTTP requests that OL sends to WMS services we add to a OL map.

tilled vs. un-tilled sources

OpenLayers accepts a large variety of input sources and you can use it to combine data layers coming from many different locations, servers and software. OL can take in tiled or untiled data. For non-tiled vector data one has to specify the *singleTile:true* tag in the layer specification. This will perform faster for this use case than tiled layers. Tiled data is provided by all the commercial maps (Google, MS) or can be rendered and stored/cached via TileCache from you own WMS sources. In that use case this will be much faster since the tiles already exist on the server and don't have to be produced on the fly by a rendering engine with each new map request. The OL website has resources to learn about the API and also a nice collection of examples on how to use or setup functionalities with OL. The list of examples can be found here: [OpenLayers 2](#) and [OpenLayers 3](#).

4.1.1 Some Important OpenLayers Objects

OpenLayers.Map the OL map object

The OpenLayers API architecture uses an object oriented approach. The following section will introduce a few of the main components that make up a typical OL viewer web page. The main component in OL is the **map object**. Once the map object is defined, other components, such as data layers and tools (map controls) can be added to it.

The following JavaScript snippet (taken from `ol_map_simple.html`)

shows a simple definition of a map object defining the variable called **map** including the *options* variable as parameters. In this example the parameters include the map projection, map units, and map extent. Note that a JS function *window.onload* is used to initiate the OpenLayers objects when the page is first loaded in a browser. You can find the local class live example [here](#).

```
var map;
var options = {
  projection: new OpenLayers.Projection("EPSG:3857"),
  units: "m",
  maxResolution: 156543.0339,
  maxExtent: new OpenLayers.Bounds(-13776237,5870705,-13270618,6177605)
};
```

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

The following section is defining a variety of map layer objects: **Map.Layer objects** Counties and Rails as WMS layers, and empty base layer, and two Google layers. The last line of JS code below adds the layer objects to the map (object).

```
var Counties = new OpenLayers.Layer.WMS( "counties",
  "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/wacounties_wms.map",
  {layers: 'Counties', 'transparent': true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility': false, singleTile:true} );
var Rail = new OpenLayers.Layer.WMS( "rail",
  "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/wacounties_wms.map",
  {layers: 'Rail', 'transparent': true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility': true, singleTile:true} );
var Cities = new OpenLayers.Layer.WMS( "cities",
  "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/wacounties_wms.map",
  {layers: 'Cities', 'transparent': true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility': false, singleTile:true} );
var Uninhabited = new OpenLayers.Layer.WMS( "uninhabited",
  "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/wacounties_wms.map",
  {layers: 'Uninhabited', 'transparent': true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility': false, singleTile:true} );

var base_empty = new OpenLayers.Layer("No Background", {isBaseLayer: true, numZoomLevels: 20, 'displayInLayerSwitcher': true, basename: "empty"});
map.addLayer(base_empty);
var g_street = new OpenLayers.Layer.Google("Google Streets", {'maxZoomLevel':20} ); map.addLayer(g_street);
```

```
var g_satellite = new OpenLayers.Layer.Google("Google Satellite",
    google.maps.MapTypeId.SATELLITE, {maxZoomLevel:20} );
map.addLayers([ g_street, g_satellite, base_empty, Uninhabited,
    Counties ] );
```

**OpenLayers
Control objects**

In the code section below two map controls (tools) are added to the map. First the *LayerSwitcher* (to turn layers on and off) is added, then a second control *MousePosition* (enabling the display of the mouse cursor position in map coordinates) is defined and added to the map (last line of JS code below).

```
var bounds = new OpenLayers.Bounds(-13776237,5870705,
    -13270618, 6177605);
map.zoomToExtent(bounds);

map.addControl( new OpenLayers.Control.LayerSwitcher() );
mp=new OpenLayers.Control.MousePosition();

mp.displayProjection = new OpenLayers.Projection
    ("EPSG:3857");
map.addControl(mp);
}; // end bracket of 'window.onload' function
```

**important
map control
objects**

The OL API documentation can help to expand your knowledge about the API basics described in the paragraphs above. One important section of the API docs is the part about map controls. A lot of functionality is included in the `OpenLayers.Control` (map control) objects <http://dev.openlayers.org/docs/files/OpenLayers/Control-js.html>.

**integrate commercial
map layers**

Google
MapQuest
Bing

4.1.2 Commercial Layers in OpenLayers

Commercial map layers such as those from Google, MapQuest and Bing (Microsoft) can be directly used in OpenLayers as *base* (background) layers for your map. Those layer are based on scale dependent tiles (small images) that OpenLayers (or other viewers) stitch together to provide an entire map. The idea behind this architecture that the transfer of the already existing (pre-rendered) tiles is faster than requesting a new image that is rendered (image file that is generated) on the fly by a mapping engine each time the map changes (for example in a pan or zoom request by a user). Some of the commercial map providers offer their own API that can be used stand alone without OpenLayers to make map. The benefit to use OpenLayers is being able to easily switch between base layers and the added comfort of the comprehensive OpenLayers API providing many tools to add and manipulate your own data. For

some of those providers that also supply a mapping API we will need to include a reference to the JavaScript of the provider to enable the map layer to work as expected (Google, Bing)¹. Note that for many cases those layers can be legally used without charge, however be aware that certain restrictions apply which may differ by provider. For example the Google terms of use indicate that the layers can be used in your maps (and in OpenLayers) free of charge, but when you want to put the same map behind a password protection you will need to purchase a professional license to comply with the terms of use. Note that the Yahoo API was officially discontinued in 2011 and that consequently maps based on it will cease working in the near future. Regarding Google API version 2 of it required to apply for a map API key that needed to be referenced in your map page (see page 108). For mapping layers of Google API version 3 no key is needed and **the syntax also changes slightly**. MapQuest started in 2011 to base their map tiles entirely on data from Open Street Map project. On page 108 see code snippets on how to include Google v3, MapQuest, Open Street Map, and Bing layers. There are multiple maps rendered from open street map data such as one based on Mapnik and one focusing on [suitable bicycle routes](#).

Google API

Open Street
Map

4.1.3 Adding WMS data to your map

To use Google map tiles together with our own MapServer layers in OpenLayers we define the map projection as "Spherical Mercator" (that is the projection of the Google and other commercial tiled data sources). This is necessary because MapServer can re-project our data to any projection no matter what the source projection is (on the fly), but tiles offered by the tiled data sources mentioned above have a fixed map projection that we have to match to align the map tiles and our own data correctly. We set the projection in the general map definition:

```
PROJECTION
    "init=epsg:3857"
END
```

In order for MapServer to be able to decode this epsg code it has to be specified in the epsg file (that is used by Proj4). The epsg is a text file that stores map projections as a list of projection strings. So let's go ahead and add that projection to

adding the
projection to
the epsg file

¹see page 108 for an example of an HTML page that includes such a reference

our epsg file (the file should be located in the c:

ms4w

proj

nad directory). Basically the code has to be added to the file if it does not exist – best is as the first entry that speeds up MapServer to retrieve it. In Notepad++ add and save this line on top of the file:

Spherical Mercator

```
<3857> +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0 +x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null +no_defs <
```

SQL to add the Spherical Mercator projection to the PostGIS table spatial_ref_sys

In PostGIS similarly this is also used and all possible EPSG codes (here called SRID for Spatial Reference ID) are stored in the "spatial_ref_sys" table. We can add the SRID also to the spatial_ref_sys table (e.g. for bounding box re-projections in PostGIS from decimal degrees to Spherical Mercator) via SQL like this. Insert Google Spherical Mercator (EPSG 3857):

```
INSERT into spatial_ref_sys (srid,auth_name,auth_srid,proj4text,srtext) values (3857,'spatialreference.org',3857,'+proj=merc+a=6378137+b=6378137+lat_ts=0.0+lon_0=0.0+x_0=0.0+y_0=0+k=1.0+units=m+nadgrids=@null+wktext+no_defs','PROJCS["unnamed",GEOGCS["unnamedellipse",DATUM["unknown",SPHEROID["unnamed",6378137,0]],PRIMEM["Greenwich",0],UNIT["degree",0.0174532925199433]],PROJECTION["Mercator_2SP"],PARAMETER["standard_parallel_1",0],PARAMETER["central_meridian",0],PARAMETER["false_easting",0],PARAMETER["false_northing",0],UNIT["Meter",1],EXTENSION["PROJ4","+proj=merc+a=6378137+b=6378137+lat_ts=0.0+lon_0=0.0+x_0=0.0+y_0=0+k=1.0+units=m+nadgrids=@null+wktext+no_defs"]') )
```



The Open Street Map (OSM) projection

The Open Street Map (OSM) projection is very similar to the Google Spherical Mercator projection and sometimes also called "Google Mercator". To add it to the epsg file:

SR-ORG Projection 6627 – Open Street Map (OSM)

```
<6627> +proj=merc +lon_0=0 +k=1 +x_0=0 +y_0=0 +ellps=WGS84 +datum=WGS84 +units=m +no_defs <
```

Insert into PostGIS:

```
INSERT into spatial_ref_sys (srid,auth_name,auth_srid,proj4text,srtext) values (96627,'sr-org',6627,'+proj=merc+lon_0=0+k=1+x_0=0+y_0=0+ellps=WGS84+datum=WGS84+units=m+no_defs','PROJCS["GoogleMercator",GEOGCS["WGS84",DATUM["WorldGeodeticSystem1984",
```

```
SPHEROID["WGS84",6378137.0,298.257223563,AUTHORITY["
EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["
Greenwich",0.0,AUTHORITY["EPSG","8901"]],UNIT["
degree",0.017453292519943295],AXIS["Geodeticlatitude
",NORTH],AXIS["Geodeticlongitude",EAST],AUTHORITY["
EPSG","4326"]],PROJECTION["Mercator_1SP"],PARAMETER
["semi_minor",6378137.0],PARAMETER["
latitude_of_origin",0.0],PARAMETER["central_meridian
",0.0],PARAMETER["scale_factor",1.0],PARAMETER["
false_easting",0.0],PARAMETER["false_northing",0.0],
UNIT["m",1.0],AXIS["Easting",EAST],AXIS["Northing",
NORTH],AUTHORITY["EPSG","3857"]]' );
```

Tip

There are two web sites spatialreference.org and epsg.io that are making it easier to look-up many map projections and display their definitions in multiple formats such as ESRI *prj* files, PostGIS insert statements, *Proj4 epsg codes*, MapServer map file definitions, and others.

**Projection
look-up made
easy**

4.1.4 OpenLayers Exercises

Viewing a basic Map in OL

1. Review the examples in the Appendix of this booklet
2. Create a new HTML page in Notepad++ (you can start with the file `osgis/ol_map_simple.html` as save it as `osgis/firstmap.html`, see the local class example live [here](#))
3. Add additional *base layers* such as **Google Streets**, **Google Hybrid** and **Bing** to the map. You can review the syntax in `alllayers.html` for help, see the live example [here](#).
4. Add two existing MapServer WMS layers to the OL map (e.g. `rails` and `uninhabited land`). Refer to `osgis/ol_map_simple.html` to see the syntax for adding a WMS layer. Make sure the layer is **not** configured as a *base layer* and that is switched off by default. The basic syntax of a WMS layer object in OL is documented [here](#) and has the parameters *name*, *url*, *params*, and *options*.
5. Add a MapServer legend to the page. You can use `osgis/one_wms_layer.html` as an [example](#). To do this you can add a HTML image tag `` to the



page and reference a GetLegendGraphic request to a MapServer WMS layer. You may want also to review the syntax in the chapter about Web Map Services. Note how HTML <div> elements and CSS styles definitions are used to layout the page.

6. Optional and advanced: Review the *Code Listing for os-gis/identify/map.html* in the back of this booklet on page 112 and add the identify feature functionality for one WMS layer of your choice to your map. Edit the file to enable identifying features in your WMS layer. View live example [here](#).

4.1.5 OL Examples (version 2.13)

See also the more detailed list in the resources section on page 112 "OL examples included on the class DVD".

- Identify Features Example (map.html on DVD)
- Digitize Polygons Example (mapserver_wa_digitize.php and save_polygon.php on DVD and [live example](#))
- Measure Distance and Area [Example](#) and [control documentation](#)
- Click Event Handler [Example](#). This event can be used to trigger any action to write inside click event script. For example we could query PostGIS, retrieve feature attributes and open a pop-up or web page.
- Review alllayers.html from the class DVD as an example how individual Commercial Layers - Google, Bing, MapQuest - can be added to a OpenLayers map along with Open Street Map tiles.

4.1.6 Optimizing OpenLayers

Over the course of the last years the OpenLayers library grew and had a lot of new functionalities added to it. While this is great and allows to build more and more complex web application using the API, the library roughly doubled in size. Here are a couple of ways one can go about optimizing OL performance.

**Apache
HTTP
compression** **com-**

- A general approach for your web (mapping) application can be to use HTTP compression for Apache. More information on how to configure this [here](#).

**om built
Layers**

- Another approach is to reduce the functionality of the OL library in a production environment to just the functionality needed and to strip everything else. In many cases this can result in a OL library that is 50% or more smaller than the standard OL (version 2) built.

4.1.7 OpenLayers is also used as

- data viewer in Spatial Data Integrator (Spatial ETL)
- optional viewer in Mapbender
- the default map viewer in Django (a content management system)
- a data viewer in the web based GeoServer configuration interface

4.2 Using MapServer

Mapserv.exe is the CGI portion for the MapServer package. It handles user input and directs image creation or query requests. The program accepts input via either "GET" or "POST" methods and can be used in an interactive manner or as an image engine. The documentation is available online in the [reference manuals](#) and in PDF format [MapServer manual download \(850+ pages\)](#). Note that the system user that is running Apache (or your HTTP Server) needs read/write permissions for the log file to make this work. You may need to create an empty log file and set the correct permissions. On Linux this is usually the user called *www-data*. Another method to enable logging is by setting an environment variable `MS_ERRORFILE` (see the MapServer manual, Chapter 5, page 86).

In this class we are using MapServer in CGI mode via HTTP requests only (not MapScript)

To run MapServer and successfully use it interactively in a browser, several prerequisites exist:

Prerequisites to run MapServer

- HTTP server is installed and running \Rightarrow Apache2
- Apache configuration is setup correctly \Rightarrow Aliases, web directory, permissions
- If we are using a scripting language this has to be configured correctly \Rightarrow PHP installed , php.ini setup correctly

- Libraries MapServer is depending on are installed (several have to be compiled with MapServer). GDAL, OGR, Proj etc.
- Fonts, symbol files that are referenced in your map file need to exist and be located in the correct directory
- A temp directory for the map output needs to be specified

Note that windows fonts can be used with MapServer

Many of those prerequisites are already taken care of when we install a pre-configured package such as MapServer for Windows (MS4W).

4.2.1 Installing MS4W - MapServer for Windows

Installation instructions for MS4W ² are available on the [web](#) and the original package download can be found [here](#)



The paragraphs in the next section are quoted from (the older web page) maptools.org.

Extracting MS4W for the first time

Please read through the following instructions before starting your installation.

1. To install the MS4W .zip file, use a compression program (e.g. SevenZip) to extract the package at the root of a drive, e.g., drive C:. If successful, you should have a new directory named 'ms4w' at the root of the drive you chose (e.g. C:/ms4w).
2. Start your MS4W Apache Web Server by running c:/ms4w/apache-install.bat (at the command line or by double-clicking it). This file installs Apache as a Windows service (called "Apache Web Server") so that it starts whenever your machine is restarted. When executed, a DOS window should pop up with the following message:

**unzip the
package to
C:/ms4w**

**Start Apache
HTTP Server**

*Installing the Apache MS4W Web Server service
The Apache MS4W Web Server service is successfully
installed.
Testing httpd.conf....
Errors reported here must be corrected before the service
can be started.
The Apache MS4W Web Server service is starting.
The Apache MS4W Web Server service was started suc-
cessfully.
This means that Apache is running and installed as a
service.*

In order to run the apache-install.bat file in **Microsoft Windows Vista, and Windows 7,8,9 and 10** follow these instructions:

**Notes for
MS VISTA
and MS 7,8,10**

- a) In Windows Explorer, go to the location of your cmd.exe file (C:/Windows/System32)

²Please be aware that the MS4W package may be replaced over time with the new comprehensive package OSGEO4W and its download manager <http://trac.osgeo.org/osgeo4w/>

b) Right-click the cmd.exe executable and choose **Run as Administrator**

c) Navigate to your ms4w folder in the command prompt window and run apache-install.bat

**Test Apache
in your web
browser
http://localhost**

3. To test that Apache is running properly, open your Web browser and find your local host Web service by entering one of the following URLs:

http://localhost or http://127.0.0.1 You should now see the main MS4W page in your Web browser. This gives you general information about your install along with configuration information. If this is your first time using MS4W it is very important that you review the listed "Features" installed within MS4W, and test them by selecting each link found on this page.

**MS4W is in-
stalled!**

4. Technically, at this point, MS4W is installed! However, as you may have noticed from the MS4W main index.html page, there are no applications running. What this means is that there are no fully configured Web applications such as e.g. GeoMoose or MapBender found within MS4W's Web-accessible directory, /ms4w/apps/. The MS4W-configured Web applications can be found on <http://www.ms4w.com/download.html> as separate zip files.

**Optional web
applications
not used in
this class**

5. To, install these Web applications into /ms4w/apps/ all that is required is to download and unzip the Web application compressed file at the same root directory as MS4W (e.g., C:). We will be working for the purposes of this class with MapServer only. Thus the following details of this paragraph are for reference only. Two things should happen when uncompressing this file. First, the Web application directory should appear within /ms4w/apps/. Second, a new httpd_*.conf file should be added to /ms4w/httpd.d/httpd_*.conf. (The /httpd.d/ directory contains Apache configuration files that define which files on your computer/server are Web-accessible. For each Web application that you install, a new configuration file will be found.)

**Configure
Aliases for
use by Apache
⇒ add "osgis"**

6. The definitions of these Web-accessible directories are called Web Aliases. In order to activate a Web Alias you must restart Apache. To test your latest installed application, go to the MS4W main index.html page (i.e., http://localhost/). In the applications section you should



now find a link to the application you just installed. Select the link to the recently installed application to see if it is configured correctly. Another option is find the Web Alias in the Apache configuration file³ for your application and call it from your Web browser directly.

7. To enable the use of a number of utility programs that usually are distributed with MapServer, the environment path (for your operating system) to the c:/ms4w/Apache/cgi-bin directory needs to be set (more information on page 47). You can
- set the path manually in your Environment Variables permanently or
 - run the batch file /ms4w/setenv.bat to include this directory in your path (temporarily until next reboot)

**Set
environment
variable path
for MapServer
utility pro-
grams**

Installation of Notepad++

- Notepad++: software/notepad/npp.xyz.Installer.exe
- Setup syntax highlighting for MapServer .map files
⇒ Copy **mapfile.api** to
InstallationPath/Notepad++/plug-ins/APIs
- **windows 7 and 8**
⇒ Copy **userDefineLang.xml** to
*C:/Users/
thecurrentuser/AppData/Roaming/Notepad++/*
⇒ Copy **mapfile.api** to
InstallationPath/Notepad++/plugins/APIs/

**syntax high-
lighting for
MapServer
.map files**

2 Tips for using Notepad++:

- Press CTRL+Space for auto completion.
- Make use of the effective **column edit mode** - using Alt + Mouse dragging.

4.2.2 Configuration of MapServer

- The CGI interface can be tested at the command line by using the "QUERY_STRING" switch, such as:

³c:/ms4w/Apache/conf/httpd.conf

```
mapserv "QUERY_STRING=map=c:/class/data  
/mapfiles/wacounties_wms.map&mode=map"
```

- To save the output into an image file, use the pipe command such as:

```
mapserv -nh "QUERY_STRING=map=c:/class/data  
/mapfiles/wacounties_wms.map&mode=map" > test.png
```

Debugging

It can be very useful to switch on debugging for MapServer and write the output to a log file. One method to do this is to add two tags to your map file:

```
CONFIG "MS_ERRORFILE" "c:/tmp/mapserver.log" # path to  
log file  
DEBUG 5 # debug level 0–5 (5 is most output)
```

Configuration of MapServer output is done via a map file

Configuration of map output with MapServer is done in the map file. The map file is not an XML file (however it probably would be one if MapServer was invented today). It uses tags similar to XML to configure the output and functionalities of MapServer. The Map file online references are located here:

MapServer reference guides

Official MapServer site
<http://mapserver.org/mapfile/index.html#mapfile>

be aware of map file syntax changes for MapServer version 7

Note that in MapServer version 7 some of the map file syntax and data handling by MapServer changed from previous versions. This is especially important when using example map files from earlier MapServer versions that can be found in forums or elsewhere on the internet. To review the changes consult the migration guide http://mapserver.org/MIGRATION_GUIDE.html#mapserver-migration-guide.

A basic map file

Example of a map file

```
MAP #Start of map file (wacounties.map)  
NAME "Washington Counties"  
EXTENT 600000 -800000 2750000 1000000 # bounding box (map  
extent)  
SIZE 600 300 # size of output  
map in pixels  
PROJECTION  
"init=epsg:2285" # Map Projection WA State Plane N in  
feet  
END  
LAYER # Data Layer object
```

```

NAME Counties      # name of layer used as a reference
                    by MapServer
TYPE POLYGON       # spatial type
STATUS ON          # status (on/off/default)
DATA "c:/class/data/layers/counties2008" # input
                    data source (shape file in this case)
CLASS              # classification
STYLE
    COLOR 255 128 128
    OUTLINECOLOR 96 96 96
    WIDTH 1
END
END # Class END
END # Data Layer END
END # Map File

```

Create map images on the desktop: shp2img

To create a map image from a map file on the desktop shp2img can be used. The output can either be PNG or GIF. This is useful to test your map file. If all goes well an image will be returned. If the image cannot be created an error message on the command line will refer to a line number in the map file. More information at <http://www.mapserver.org/utilities/shp2img.html>.

**map images on
the desktop
shp2img**

```

shp2img -m mapfile [-o image] [-e minx miny maxx maxy] [-s
    sizex sizey]
    [-l "layer1 [layers2...]" ] [-i format]
    [-all_debug n] [-map_debug n] [-layer_debug n]
    [-p n] [-c n] [-d
    layername datavalue]
-m mapfile: Map file to operate on — required
-l format: Override the IMAGETYPE value to pick output
    format
-o image: output filename (stdout if not provided)
-e minx miny maxx maxy: extents to render
-s sizex sizey: output image size
-l layers: layers to enable — make sure they are quoted
    and space
    separated if more than one listed
-all_debug n: Set debug level for map and all layers
-map_debug n: Set map debug level
-layer_debug layer_name n: Set layer debug level
-c n: draw map n number of times
-p n: pause for n seconds after reading the map
-d layername datavalue: change DATA value for layer

```

For example to create an image with the Counties and Rail layer in *wacounties_wms.map*:

```
shp2img -m c:/class/data/mapfiles/wacounties_wms.map  
-l "Counties Rail" -o counties_rail_map.png
```



4.2.3 MapServer Exercises

4.2.3.1 Making a basic Map

1. Review the basic map file *wacounties.map* and create a png map image output using *shp2img*
2. Open the file *../data/map files/wacounties.map* in Notepad and add the layers: *citynames*, *rails*, and *uninhabited areas*
3. Review your configuration using *shp2img*. Troubleshoot any errors
4. Use the reference guides and sample map files (in the end of this booklet) to find the correct syntax for classification of layers by attributes.
5. Classify one of the layers you added to *wacounties.map* using the **EXPRESSION** map tag assigning different colors for its attributes (or attribute ranges). Review one example for classification of attribute ranges using the **EXPRESSION** tag in the map file starting on page 101.

4.2.3.2 Making your own Map

**reference
guides and
sample map
files for help**

Use the MapServer reference and sample map files in this guide for help. Review your progress periodically using the *shp2img* utility.

1. Either use the map file created in the above or create a new one. You may use your own data if desired.
2. Add at least five data layers and classify their cartographic representation using class tags.
3. Add scale dependency to the layers in the map file (**MAXSCALE/MINSCALE** or **MAXDENOMSCALE/MINDENOMSCALE**) so that they are only available at certain defined scales
4. Use scale dependency for a layer to change the cartographic output for small scales (overview) and large scales (detailed) for the map output.

5. Add labels to the city layer (or another layer you have defined)
6. Add a tif image file to your map using the RASTER type layer. You may use any of the tif files located in the *data/layers/catalog/topography* directory. An example of a RASTER Layer (*Topography*) can also be found in the map file starting on page 101
7. Add a tiled vector layer as data source to the map using the TILEITEM and TILEINDEX tags. The *data/layers/catalog* directory contains *tiled* shape files and their respective tile index shape files (containing the boundaries of the cut shape files) for lakes and rivers on multiple scales.
8. Review how you can tile shape files (and create the tile index files) yourself using the utilities shp2tile⁴ and tile4ms⁵ (see page 47. One example can be found in the map file *data/mapfiles/map_file_library/alliance_base_layers.map* for the *50k Tiled Layer Rivers_and_Streams*.

4.2.4 Additional Topics

Review the online documentation for these interesting topics:

- Map Templates
- Query Templates
- Variable Substitution

A simple way to embed your map output in a web page are MapServer templates. This is an example "Help me a spider" of a simple HTML file and a respective map file (quoted from the book by Bill Kroppla (2005): *Beginning MapServer: Open Source GIS Development* -Chapter two). Note that MapServer can replace variables that are embedded in brackets[] during run time. In this case MapServer will replace the **[img]** variable and return the rendered image.

variable substitution in MapServer templates

data/mapfiles/helpmespider.html:

```
<!-- MapServer Template -->
<html>
```

⁴cut a big shape into a grid of adjacent shape files

⁵create tile index shapefile for use with MapServer's TILEINDEX feature. This will create a shapefile of rectangular extents of all shapefiles listed. Note that the utility *ogrtindex* of the OGR-package has similar functionality


```
<head> <title>MapServer Easy Map</title></head>
<body>
  <form method=POST action="/cgi-bin/mapserv.exe">
    <input type="submit" value="Click_Me">
    <input type="hidden" name="map" value="c:/class/
      data/mapfiles/helpmepider.map">
  </form>
  <IMG SRC="[img]" width=800 height=400 border=0>
  <p>[version]</p>
</body>
</html>
```

data/mapfiles/helpmepider.map:

```
MAP #Start of map f i l e
NAME "Washington Counties"
EXTENT 630000 -540000 2700000 780000 # bounding box (map
  extent )
SIZE 600 300 # size of output map in pixels
  WEB
  TEMPLATE "/ms4w/apps/osgis/helpmepider.html"
  IMAGEPATH "/ms4w/tmp/ms_tmp/"
  IMAGEURL "/ms_tmp/"
  END
  LAYER # Data Layer object
  NAME Counties # name of layer used as a reference
    by MapServer
  TYPE POLYGON # spatial type
  STATUS DEFAULT # status (on/ off / default )
  DATA "c:/class/data/layers/counties2008" # input
    data source (shapefile in this case)
    CLASS # classification
    OUTLINECOLOR 100 100 100 # color
      for outline boundary
    END # Class END
  END # Data Layer END
END # Map File END
```



4.2.5 Publishing Web Map Services (WMS) with MapServer

To publish a layer or layers as a WMS some additional information is required. One or more layers of a map file can be published as part of a WMS. Depending on the settings, the layers can be accessed by a WMS client (e.g. ArcMap or uDig) independently or as an entirely configured map. A WMS will

**One or more
layers of a
map file can
be published
as part of a
WMS
GetMap
request
example**

give back an image if it gets the appropriate **GetMap** request. Layers configured as DEFAULT will always be output to any incoming request even if they were not requested. Below is an example of a **GetMap** request asking for

- a map in **PNG** format
- with the layer **States**
- in projection EPSG 2285 (Washington State Plane North)
- with bounding box (extent) 630000,-540000,2700000,780000
- in 400 by 300 pixels size

http://terra2.terraxis.net/cgi-bin/mapserv?map=/var/www/sites/mapdata/projects/alliance/alliance_background.map&version=1.1.1&service=WMS&request=GetMap&layers=States&srs=EPSG:2285&bbox=630000,-540000,2700000,780000&format=image/png&width=400&height=300&styles=default

The OGC standard defines the type of requests that a WMS supports. The following requests are defined by the standard:

- GetCapabilities <http://<computername>/mapserver?map=../demo.map&VERSION=1.1.1&REQUEST=getCapabilities&SERVICE=wms>
⇒ What can the WMS offer ? Returns service-level meta-data in XML format
- GetMap (example above) ⇒ returns a map with well-defined geographic and dimensional parameters
- getFeatureInfo (optional)
⇒ returns information about features (identify attributes)
- getLegendURL (optional)
e.g. http://terra2.terraxis.net/cgi-bin/mapserv?map=/var/www/sites/mapdata/projects/alliance/alliance_boundaries.map&version=1.1.1&service=wms&request=getlegendgraphic&layer=Tract_Boundaries&format=image/png
⇒ returns a legend graphic

This figure illustrates a client-server architecture for geospatial web services (geoservices). Green color represents read and write paths. Dotted arrowed lines indicate mostly read-only data flow (this figure is freely available from [Wikimedia Commons](#)).

**client-server
architecture
for geospatial
web services**

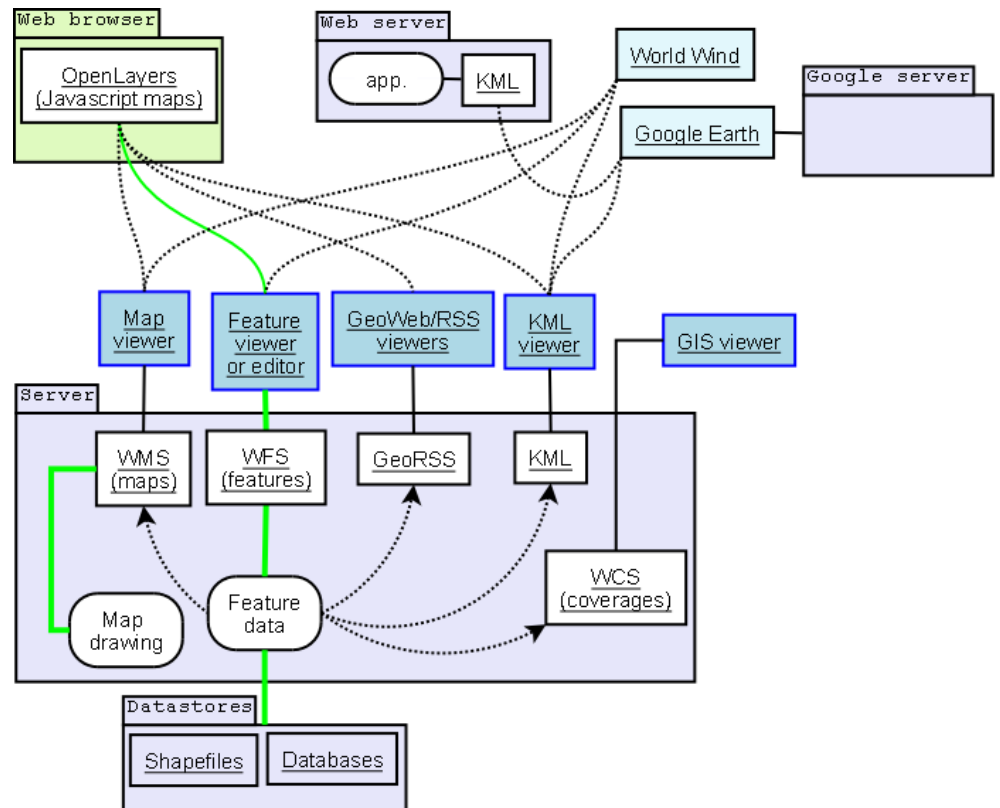


Figure 6: Client-server Architecture for geospatial web services

To change a regular layer in a map file into a WMS layer we need to define several METADATA tags for it (see example below). The wms_srs tag defines the output projection(s) that are provided by MapServer and that a client (e.g. OpenLayers) can request:

METADATA

```
"wms_title" "PrecinctTargets"      # Name of WMS
"wms_srs" "epsg:3857 epsg:4326"    # Spherical
                                   Mercator and Geographic Projection
"wms_feature_info_mime_type" "text/html" #Format of
                                   Query (GetFeatureInfo requests)
"ows_enable_request" "*"           # needed in MS 6
                                   and higher to enable WMS requests
```

END

Note that with MapServer you need to use the MapServer layer name (defined in the map file under LAYER, NAME to request it in a HTTP request and not with the wms_title as you might expect). *"The latter is used to produce the title element for the layer in a WMS GetCapabilities request and is usually displayed by WMS clients as the human-readable caption of the layer. The MapServer LAYER NAME is mapped to the <Name> element in the WMS GetCapabilities which is the unique name/id of the layer, this is the identifier by which client and server software refer to this layer in GetMap and other requests"* (Daniel Morissette communication 2009). The MapServer documentation has more details about the use of WMS with MapServer: http://mapserver.org/ogc/wms_server.html.

**WMS meta-
data tags**

**in WMS re-
quest use the
MapServer
layer name -
defined in the
map file under
LAYER, NAME**

4.2.6 WMS Exercises

Publishing your own WMS

1. Use the map files written during the earlier MapServer exercises and publish the layers as Web Map Services
2. Verify your results via HTTP requests in a web browser (examples for requests are in the section about WMS standards)
3. Try different kind of requests - **GetMap** and **GetLegend** and save the results as files on your laptop. Use the reference guides and this booklet to find the correct syntax



4. Install any or all of the desktop GIS systems on your laptop (gvSIG, QGIS, uDig) and open your WMS on the desktop.

4.2.7 MapServer Performance Tips

MapServer Performance Tips

A good resource to find more information about tuning the performance of MapServer is the reference at <http://www.mapserver.org/optimization/index.html#optimization>

4.2.7.1 Tuning your map file for performance:

Use inline projection parameter definitions (*spelled out* parameters) in place of EPSG codes. If you want to use EPSG codes, remove all unneeded projection definition records from the Proj4 database. Or put the most used on top of the file.

For every layer in a map file that has a status of ON or DEFAULT, MapServer will load that layer and prepare it for display, even if that layer never gets displayed \Rightarrow Switch rarely used layers off (don't use DEFAULT for them).

4.2.7.2 Optimizing the performance of vector data sources

Splitting your data (use of filters is slower) with ogr2ogr utility (select on certain features from a data source, and save them to a new data source).

Shapefiles

Use shptree to generate a spatial index on your shapefile. This is quick and easy ("shptree foo.shp") and generates a .qix file. MapServer will automatically detect an index and use it. In addition to using Tileindex the data can at the same time also be indexed with shptree.

Use the sortshp utility This reorganizes a shapefile, sorting it according to the values in one of its columns. If you're commonly filtering by criteria and it's almost always by a specific column, this can make the process slightly more efficient.

PostGIS

Add indexed primary key

```
ALTER TABLE locations ADD CONSTRAINT locations_pkey_id PRIMARY KEY(id);
```

Spatial indexing (GIST index)

```
create index locations_gidx on locations using gist(geom);
```

Reorganization

```
CLUSTER the_geom ON table;
```

Do not close DB connection by default

```
PROCESSING "CLOSE_CONNECTION=DEFER"
```

Maximum Image size influences the time needed for map rendering. Default is 2048. This can be increased by setting in the map file *MAXSIZE [integer]* in pixels.

4.2.8 Tools to work with MapServer

Most MapServer distributions come with a number of utility programs (see table below). To run the mapserv.exe utilities, the environment path to the c:/ms4w/Apache/cgi-bin directory must be set. You can

- run the batch file c:/ms4w/setenv.bat to include this directory in your path
- or even better permanently set the path manually in your windows Environment Variables (-> Control Panel -> System and Security -> System -> Advanced system settings -> Environment variables) and add to **PATH** after all that is in there already: ;c:\Program Files\PostgreSQL\9.4\bin;c:\ms4w\Apache\cgi-bin;c:\ms4w\tools\gdal-ogr;c:\ms4w\tools\mapserv;c:\ms4w\tools\shapelib;c:\ms4w\proj\bin;c:\ms4w\tools\shp2tile;c:\ms4w\tools\shpdiff;c:\ms4w\tools\avce00;c:\ms4w\python\gdal;c:\ms4w\tools\php;c:\ms4w\tools\mapcache;c:\ms4w\tools\berkeley-db;c:\ms4w\tools\sqlite;c:\ms4w\tools\spatialite;c:\ms4w\tools\unixutils;

For more information read the online reference guide at <http://www.mapserver.org/utilities/>.

**MapServer
utility
programs**

Additional utilities are included with MS4W in the *../m4w/tools/shapelib* directory. The tools coming with *shapelib* provide tools to work with shapefiles (read/write/update) and to write your own C programs. To mention one of them *shprewind* is a utility that can correct the winding order (sequence of vertices) in a shapefile, and can be used to repair defective shapefiles. Another

Table 12: MapServer utility programs

| Utility name | description |
|--------------|--|
| legend | create legend |
| msencrypt | encrypt encryption key for connection parameters |
| scalebar | create scalebar |
| shp2img | create mapoutput |
| shptree | create quadtree-based spatial index for a shapefile (a .qix file) |
| shptreevis | view quadtree quadrants that are part of a .qix file |
| sortshp | sort the records of a shapefile based on a single attribute column (ascending/descending order) |
| sym2img | create dump of symbol file in PNG or GIF format |
| tile4ms | create tile index shapefile for use with MapServer's TILEINDEX feature. This will create a shapefile of rectangular extents of all shapefiles listed |

handy tool is *shp2tile* that divides a (large) shapefile into multiple smaller tiled shapefiles. For more information read the user guide at <http://shapelib.maptools.org/shapelib-tools.html>. The GDAL utilities provide tools to manipulate raster data. To split large raster files into tiles the python utility *gdal2tiles* comes in handy. From the gdal homepage: *"This utility generates a directory with small tiles and metadata, following OSGeo Tile Map Service Specification. Simple web pages with viewers based on Google Maps and OpenLayers are generated as well - so anybody can comfortably explore your maps on-line and you do not need to install or configure any special software (like MapServer) and the map displays very fast in the web browser. You only need to upload generated directory into a web server."* MapTiler is a graphical interface for the *gdal2tiles* utility <http://www.maptiler.org/>.

divide large shapefiles into a set of smaller "shapefiles"

Tools to create MapServer configuration files

map files can be created manually or using a tool

You can either write your map file manually using your favorite text editor (e.g. we have seen Notepad++ has syntax highlighting support, Textpad <http://www.textpad.com/> syntax file: http://www.textpad.com/add-ons/files/syntax/map_

[40.zip](#), and UltraEdit (which itself is not free however). The utilities listed below can help you to write and manage map files:

Table 13: Utility programs for creating MapServer configuration files

| Utility name | description |
|-------------------|---|
| MXD2map | is a Java based command line utility to convert ArcGIS documents (.mxd) files to .map files. Download from http://www.mxd2map.org/index.html |
| AveiN! and AMeiN! | are two utility programs for ArcView 3 and ArcMap respectively that allow you to create a map and to export the result as a .map MapServer configuration file. AMeiN! translates into "ArcMap einfach ins Netz!" which is German for something like: "Loading ArcMap projects into the web made easy!". Download from http://sourceforge.net/projects/avein/ |
| QGIS 2.X | can export a QGIS project as a .map file with the RT MapServer export plug-in (install via OSgeo4W as a python MapScript installation is needed) |
| Scribe | A set of utilities (GUI and Python scripts) that facilitate the creation of MapServer configuration files. More information can be found on the Github page and in initial article . |

4.2.9 Notes about MapServer

The following paragraphs are a short collection of handy notes about MapServer and by no means are intended to represent a complete guide to using MapServer. You are encouraged to explore by yourself and revert to the comprehensive MapServer manual (on DVD) and also available on www.mapserver.org.

Migration from older versions

With each new release of MapServer new features will likely be added to it and bugs fixed, however with mayor version jumps (e.g. from 5.6 to 6.0) there will likely (unfortunately) be some changes to the map file syntax. New map file tags

**migrating to
new
MapServer
versions**

will be introduced and old ones can be replaced and deprecated. Map files obtained from the Internet or other sources can be of great help to build a cartography map file library for yourself. When you are using such map file with newer MapServer versions it will be a good idea to read the migration guide that is published in the MapServer web site. This way you can find out about potential changes that might be required to have your map file working as expected and which changes to your map file might be required. For version 6 (and for earlier versions) the document is located at http://mapserver.org/MIGRATION_GUIDE.html#migration.

MapServer Image Rendering Options

cartographic output configuration for MapServer

MapServer includes a variety of options for producing cartographic output (generating an image file such as jpg, png or gif formats for example). Starting with MapServer version 5 a new rendering engine called AGG (*Anti Grain Geometry* <http://www.antigrain.com>) was included. Earlier versions of MapServer had only the 'GD' rendering available which has been removed in version 6 of MapServer. With AGG in newer MapServer versions all output is antialiased⁶ which enables very fine cartographic output. Some symbols such as cartographic lines (that where needed to enable anti aliasing in earlier versions of MapServer) are now deprecated (no longer supported). For more information see <http://mapserver.org/output/agg.html#introduction>. Below is an example for a fairly nice map output configuration balancing rendering time , file size and cartographic quality. Another good comparison of map rendering settings and out for MapServer can be found at <http://linfiniti.com/2010/03/comparing-renderers-in-mapserver>.

OUTPUTFORMAT

```
NAME 'AGG_Q'      # arbitrary name, call in the map file
DRIVER AGG/PNG    # (see below)
IMAGEMODE RGBA
TRANSPARENT OFF
FORMATOPTION "INTERLACE=OFF"    # for tilecache this
                                needs to be set to off
FORMATOPTION "QUANTIZE_DITHER=OFF"
```

⁶On Wikipedia states '*Font rasterization is the process of converting text from a vector description (as found in scalable fonts such as TrueType fonts) to a raster or bitmap description. This often involves some anti-aliasing on screen text to make it smoother and easier to read*' see http://en.wikipedia.org/wiki/Font_rasterization

```
FORMATOPTION "QUANTIZE_COLORS=256"  
MIMETYPE "image/png" #  
END
```

To use the option above add to your map file in the general MAP tag

```
IMAGETYPE AGG_Q
```

Using True Type Fonts in MapServer on Windows

On many windows operating system versions you can find (true type) fonts (with the *.ttf file extension) on the file system at c:\Windows\Fonts\verdana.ttf. Those fonts can be used by MapServer for rendering labels (or even for using font characters as map symbols). In order to use the files MapServer will need to know about the location when rendering. This information can be indicated in the map file as the location of a text file that lists the fonts you are intending to use and their location (see syntax below). For example see the content of such a file *list.txt* below.

```
FONTSET "c:/class/data/mapfiles/fonts/fonts.list"
```

If the fonts are in the same directory (copy and list only the ones that you are really always using) the content of the the text file **list.txt** could look as shown below to enable fonts *verdana*, *verdana-italic*, and *wingdng3* (a windows font with many simple symbols)

```
verdana      verdana.ttf  
verdana-italic  verdanai.ttf  
wingdng3 WINGDNG3.TTF
```

**using windows
fonts with
MapServer**

4.3 Using PostGIS

When to use a spatial database instead of files

One consideration in the beginning: When to use a spatial database instead of files ? Basically a database should only be used in particular use cases:

- editing (multiple users)
- unified storage and access
- bulk processing

Prerequisites : PostgreSQL and PostGIS installed



PostGIS brings the power of spatial operations to PostgreSQL



4.3.1 Installing PostGIS

Chapter two of the document "Introduction to PostGIS" (in the workshop materials: [Postgis-introduction.pdf](#)) outlines the installation of PostgreSQL and PostGIS. Instead of PostgreSQL version 8.0 in our workshop we are installing newer versions of PostgreSQL and PostGIS accordingly. Note that there were considerable changes from PostGIS version 1.5.X to PostGIS 2.X that also affect the installation process. In the sections below the PostGIS version to which they apply is indicated. In our workshop exercises we will create a database called "osgis" instead of "BC" (as named in the document "Introduction to PostGIS").

- Install PostgreSQL
- Install PostGIS
- Create plpgsql language
on the SQL console: `createlang plpgsql mydb`

Changes from PostGIS 1.5.X to versions 2.X

Some highlights of the new functionalities in PostGIS 2.0 and up include raster data support (storage and spatial and analysis functions), comprehensive support for handling 3D data with surfaces and relationships, and the introduction of topology for vector data. Raster files (such as tif files e.g.) can be loaded into the data base using the command line utility [raster2pgsql](#) which has similar syntax as [shape2pgsql](#) for loading ESRI shapefiles. Raster data can be stored and used in two ways with PostGIS - *inside* and *outside* the data base. Inside stored raster are loaded into the database as a *blob*, whereas

out-db rasters are images located on the file system and referenced in a table from within the database ([see -R switch in the command line utility reference](#)). This workshop will not cover details of working with PostGIS rasters.

Note that to enable deprecated functionalities in PostGIS 2.X (database functions available in prior versions of PostGIS but not by default in PostGIS 2.X) you can run specific SQL queries in order to enable those. This might be relevant especially if importing data from a PostGIS 1.5 system into PostGIS 2.X after an upgrade (in order to not have to rewrite many of the functions used in your application).

```
# change to SQL script directory
cd C:/Program Files/PostgreSQL/9.4/share/contrib/postgis-2.1/
# enable legacy functions in PostGIS 2.X
psql -d osgis -p 5432 -f legacy.sql
```

Setup of PostGIS versions 1.5.X (and older)

The initial step will always be to create a new database. This can be accomplished by either

using [PgAdmin](#) (the GUI based administration tool)

on the command line: *createdb osgis -U postgres*

In order to load and enable all PostGIS functionality in PostgreSQL we need to run the following two SQL scripts:

Loading PostGIS functions, casts etc into the db

```
c:/Program Files (x86)/PostgreSQL/9.4/share/contrib/postgis-1.5/postgis.sql
```

Loading the spatial reference table into PostGIS

```
c:/Program Files (x86)/PostgreSQL/9.4/share/contrib/postgis-1.5/spatial_ref_sys.sql
```

Instead of running these scripts above we also could create a spatially enabled database from a template that has the tables and functions already loaded.

Setup of PostGIS versions 2.X (and newer)

Again the initial step will be to create a new database. This can be accomplished by either

using [PgAdmin](#) (the GUI based administration tool)

on the command line: *createdb osgis -U postgres*

Starting with version 2.0 PostGIS is a regular Extension of PostgreSQL and can be installed in the same way as other additions to the database as follows:

Install PostGIS into a database on the command line

```
CREATE EXTENSION postgis;
```

Install topology support and functions

```
CREATE EXTENSION postgis_topology;
```

Install tiger data based geocoder

```
CREATE EXTENSION postgis_tiger_geocoder;
```

Enable fuzzy string match functions for geocoder

```
CREATE EXTENSION fuzzystrmatch;
```

4.3.2 Interacting with PostGIS



There are at least two options to interact with PostGIS. One can either use the command line utilities from a DOS prompt, or the Administration tool pgAdminIII that comes with PostGIS. For big data sets it is advisable that we use the command line since that can handle larger datasets than pgAdmin seems to be able to digest.

PostGIS - command line

There are manuals covering this topic available so we list here some examples only:

Creating a database `createdb -U postgres osgis`

Deleting a database `dropdb -U postgres osgis`

connecting to a db

```
psql -U postgres -d osgis
```

after that any sql or psql command can be used

psql command to list all tables `\dt`

psql command list all column of table counties2008

```
\d counties2008
```

Running SQL from file

```
psql -U postgres -d osgis -f thepath/myfile.sql
```

PostGIS - Simple Spatial SQL Queries

Some of these examples are taken from the sql file postgis-introduction.sql

— *Distance Query*

```
create table points ( pt geometry, name varchar );
insert into points values ( 'POINT(0_0)', 'Origin' );
insert into points values ( 'POINT(5_0)', 'X_Axis' );
insert into points values ( 'POINT(0_5)', 'Y_Axis' );
select name, st_astext(pt), st_distance(pt, 'POINT(5_5)') from points
;
drop table points;
```

—*Creating a spatial index*

```
create index counties2008_gidx on counties2008 using gist (geom);
```

—*Intersection*

```
select NAME from counties2008 where counties2008.geom && transform(((
    setsrid((MakePoint(−122.206834, 47.611421)),4326))),2285) and
    intersects (counties2008.geom,transform((( setsrid((MakePoint
    (−122.206834, 47.611421)),4326))),2285));
```

—*Select one spatial feature and reproject:*

```
select st_astext(st_transform(geom,4326)) from counties2008 limit 1;
```

— *reproject two points from geographic (SRID 4326) to spherical
Mercator projection (SRID 3857) and output as test*

```
select st_astext(transform((SETSRID(Makepoint(−88.2,37.7),4326))
    ,3857)), st_astext(transform((SETSRID(Makepoint(84.6,41.8),4326)
    ),3857));
```

— *select all points within specified counties*

```
select counties2008.name, locations.id, locations.name from locations
    , counties2008 where
st_intersects(counties2008.geom,st_transform(locations.geom,2285))
    and counties2008.name IN ( 'King', 'Pierce') order by locations.id
;
```

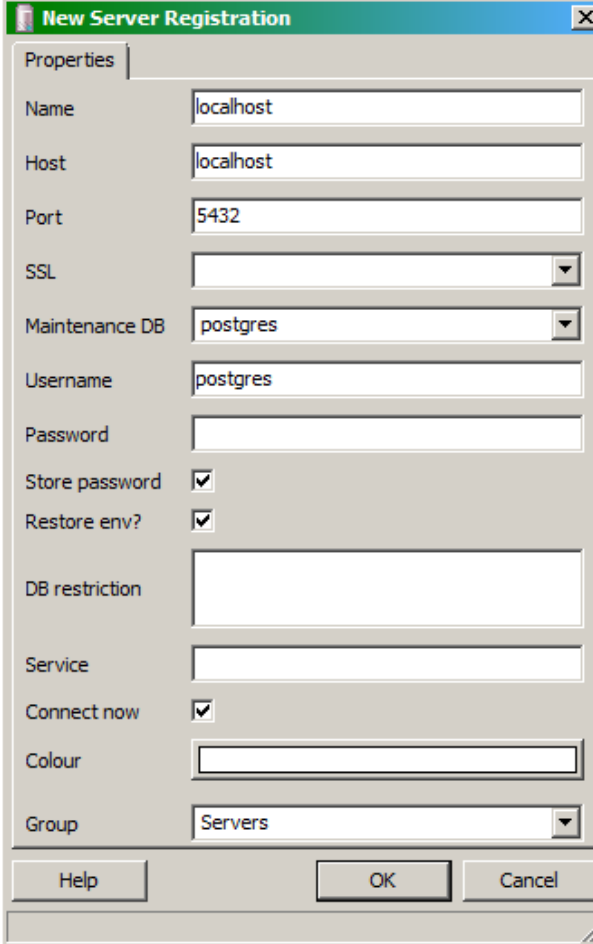
— *Update Postgresql's internal query planner*

```
VACUUM ANALYZE;
```

4.3.3 pgAdmin - Administration of PostGIS

**PostgreSQL
database
connection
parameters**

Connecting to a PostgreSQL database in pgAdmin:



The screenshot shows the 'New Server Registration' dialog box in pgAdmin. The 'Properties' tab is selected. The fields are as follows:

| Field | Value |
|----------------|-------------------------------------|
| Name | localhost |
| Host | localhost |
| Port | 5432 |
| SSL | |
| Maintenance DB | postgres |
| Username | postgres |
| Password | |
| Store password | <input checked="" type="checkbox"/> |
| Restore env? | <input checked="" type="checkbox"/> |
| DB restriction | |
| Service | |
| Connect now | <input checked="" type="checkbox"/> |
| Colour | |
| Group | Servers |

Buttons at the bottom: Help, OK, Cancel.

Figure 7: pgAdmin - Connecting to a DB Server

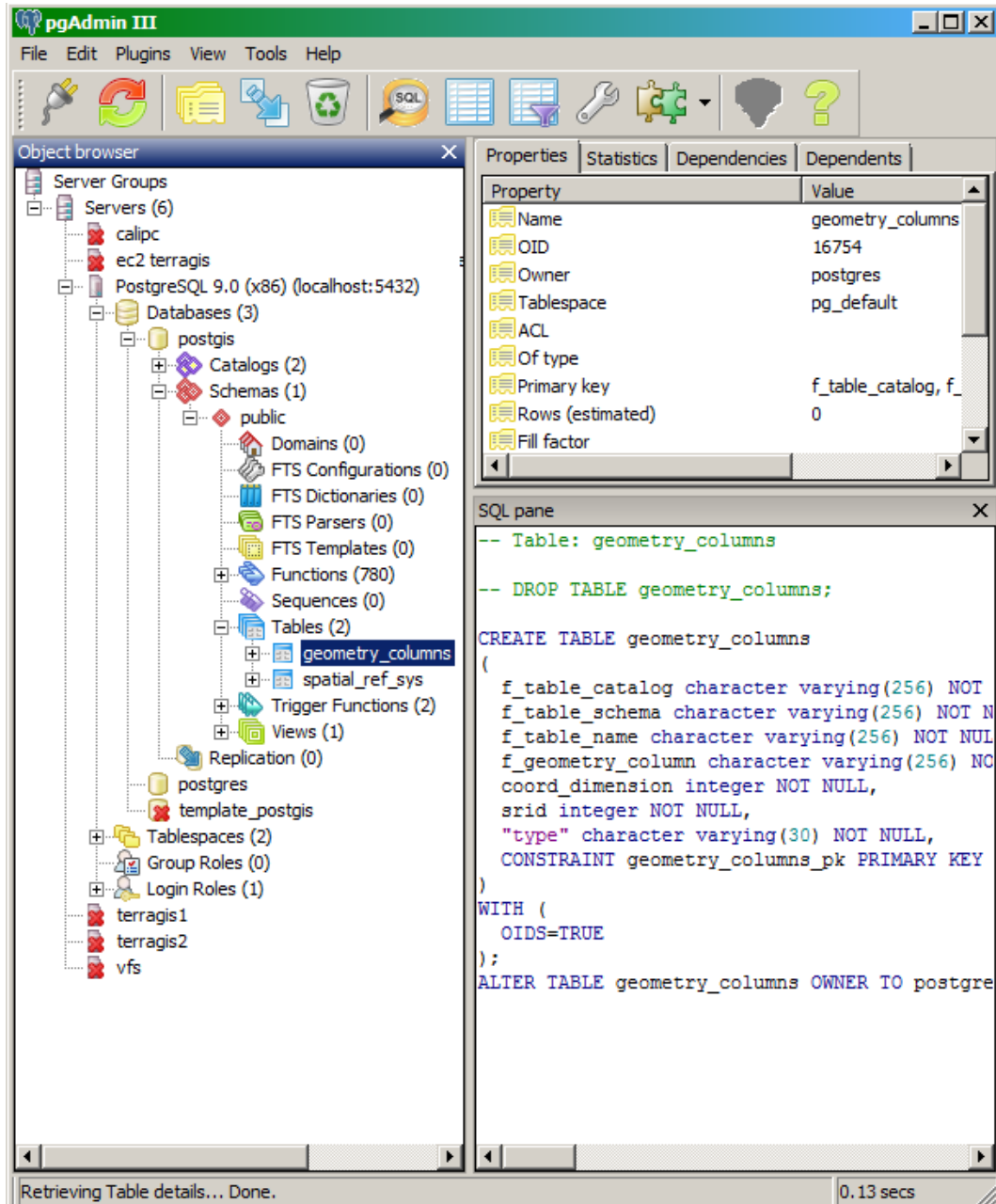


Figure 8: pgAdmin - Administration Tool for PostgreSQL

4.3.4 Utilities for PostGIS

PostGIS - Import and Export

import and export utilities shp2pgsql and pgsql2shp import and ex- port of shape- files

Two utilities that come with PostGIS allow the import and export of shapefiles from and to PostGIS. Using shp2pgsql one can convert a shapefile into a SQL statement that can be loaded into PostGIS. Using pgsql2shp, a PostGIS layer can be exported and written to a shapefile.

Located in
c:/Program Files (x86)/PostgreSQL/9.4/bin/shp2pgsql.exe

For example to convert the shapefile theimport_2008.shp (projection is decimal degrees - SRID 4326) and write to a SQL file "theimport_2008.sql" use this command :

```
shp2pgsql -I -s 4326 theimport_2008.shp in_2008 > in_2008.sql
```

Notes on optional switches to use on the command line:

- **-I** option is to create a spatial index -> this is absolutely essential to make queries perform fast
- **-i** option is to use short integers as data format for the new data base table
- **-a** append mode (to add to an existing data base table)
- **-W latin1** for encoding (use with attribute data in languages that have non-English alphabet characters. In such cases you could get the *ERROR: invalid byte sequence* without this switch.

Now to load the data into PostGIS either copy the contents of in_2008.sql into the SQL window of pgAdmin and run or run the SQL from the command line:

```
psql -U postgres -d osgis -f thepath/in_2008.sql
```

Or you can perform the above import operation in one step using the pipe command '|' to import the 2008 Washington County data in a regional map projection (WA State Plain North in feet):

```
shp2pgsql -I -s 2285 counties2008.shp counties2008 | psql  
-U osgis -d osgis
```

Note that the map projection used for the -s switch has to match the projection the shape file to import is in. In order to load GIS data sources other than shapefiles the ogr2ogr utility can be used (see the table in the resource section for supported data formats). Another option to import data into PostGIS is the **OGR Converter Tool**, a plug-in that you can install with QGIS). This is an example of loading a dataset from an ESRI Personal Geodatabase into PostGIS using ogr2ogr on the command line:

**importing
formats other
than shapefile**

```
ogr2ogr -f "PostgreSQL" PG:"host=localhost user=postgres  
port=5432 dbname=postgis_in_action password=mypassword"  
gadm_v0dot9.mdb -lco GEOMETRY_NAME=the_geom  
-where "ISO='USA'" -t_srs "EPSG:2163" -nln "us.admin_boundaries"  
gadm
```

Loading data from a file into PostGIS and making them spatial

The following work flow is an example to create a table in PostGIS, to load data into the empty table and then converting it into a spatial data set.

— *Creating and empty table called locations via SQL*

```
CREATE TABLE locations
(
  id int,
  name varchar(30),
  latitude float8,
  longitude float8,
  gid serial);
```

— *make user osgis the owner of the table*

```
ALTER TABLE locations OWNER TO osgis;
```

— *Load data with the copy command*

```
copy locations (id, name, latitude, longitude) from 'c:/class/data/
  layers/point/locations.csv' using delimiters ',' with null as '
  ';
```

— *Create spatial features from the new data:*

— *To add a geometry column (only really needed for PostGIS 1.5 and earlier, but also can still be used in PostGIS 2.X !):*

```
select AddGeometryColumn('', 'locations', 'geom', 4326, 'POINT', 2);
```

— *when using PostGIS 2.0 and up one can use instead a column for the geometry in the table definition or add the column subsequently*

```
ALTER TABLE locations ADD COLUMN geom geometry(Point, 4326);
```

— *Update new column with features generated by the Makepoint function, with EPSG (SRID) 4326 (that's decimal degrees)*

```
update locations set geom = ST_SETSRID(st_Makepoint(longitude,
  latitude), 4326);
```

— *Create a spatial index (very IMPORTANT) on the geometry column:*

```
create index locations_gidx on locations using gist(geom);
```

— *create unique id field*

— *Add Primary Key on id*

```
ALTER TABLE locations ADD CONSTRAINT pkey_id PRIMARY KEY(id);
```

And finally delete the records from the imported table where we could not build a geometry. We just delete entries where geometry is empty: delete from locations where geom is null;

Built internal query index for PostgreSQL using the vacuum command:

```
VACUUM ANALYZE;
```

Now finally the newly loaded data is usable as input for MapServer.



4.3.5 PostGIS Exercises

1. Review the examples in this chapter
2. Create a database with the name *osgis* and a database user *osgis* using pgAdmin
3. spatially enable the new *osgis* data base (create PostGIS extension). Verify if the PostGIS spatial functions are available.
4. Import two shapefiles into PostGIS using shp2pgsql (as described in section 4.3.4 on page 58).
5. Run all the examples listed in Section "Simple Spatial SQL Queries" on page 55 as listed to see the functionalities of PostGIS in action for yourself, or use your own spatial data in similar queries
6. Following the example in section "Loading data from a file into PostGIS and making them spatial" on page 60 load data from a text file into PostGIS and convert them into a spatial data set. You can use the text file `/data/layers/point/locations.csv` for this exercise.
7. How could we efficiently store spatial data in multiple projections in PostGIS?
What would be a simple way to do this?
8. Write a spatial SQL *Select* query that uses a SQL *WHERE* clause with the intersection (true/false) of two features (a point location and a County polygon e.g.) to return the name of the County polygon at the point location. Review this section of the PostGIS documentation for [help](#).



4.3.6 Exercises - PostGIS Layers in MapServer

Additional topics to be covered in the MapServer exercises after completing the chapter on PostGIS:

1. Import a shape file layer of your choice into PostGIS using shp2pgsql (or the shape file loader in pgAdmin)
2. Add the new PostGIS layer to the map file (note use the CONNECTIONTYPE and CONNECTION tags). An example of a PostGIS Layer (*Census Tracts Query*) can also be found in the map file starting on page 101

4.3.7 Using PostGIS with MapServer

Tips from Paul Ramsey <http://blog.cleverelephant.ca/2008/10/mapserverpostgis-performance-tips.html>

The data statement for a PostGIS layer in a map file can be very simple:

DATA "the_geom from the_table"

Very simple, but: how does MapServer know what primary key to use in queries? And what SRID to use when creating the bounding box selection for drawing maps? The answer is, it asks the database for that information. With two extra queries. Every time it processes the layer. However, if you are explicit about your unique key and SRID in configuration, MapServer can, and does, skip querying the back-end for that information.

DATA "the_geom from the_table using unique gid using srid=4326"

Another good thing is to specify that the open DB connection should not be closed by default (useful for fast-cgi mode and for multiple PostGIS layers in one map file)

**keep database
connections
open**

PROCESSING "CLOSE_CONNECTION=DEFER"

4.3.8 Some Notes on PostGIS

To identify which versions of PostGIS and PostgreSQL you are running the following SQL queries can be used:

```
select postgis_lib_version();  
— lists PostGIS version (short) : "1.4.0"
```

```
select postgis_full_version();  
— lists PostGIS version including GEOS and PROJ version: "POSTGIS  
  = "1.4.0" GEOS="3.1.1-CAPI-1.6.0" PROJ="Rel. 4.6.1, 21 August  
  2008" USE_STATS"
```

```
select version();  
— lists PostgreSQL version: "POSTGIS="1.4.0" GEOS="3.1.1-CAPI-1.6.0"  
  PROJ="Rel. 4.6.1, 21 August 2008" USE_STATS"
```

Information about tuning a PostGIS installation is included in the document "**Postgis-for-power-users.ppt**" included on the DVD.

The following SQL statements illustrate the use of PostGIS spatial functions: ST_DWithin, ST_Line_Locate_Point, and ST_Line_Interpolate_Point. The queries are extracts from PHP scripts and contain some PHP variables; e.g. \$xcoord.

```
— st_within example  
— get closest road in tiger data  
SELECT fullname, tlid, st_astext(the_geom2) FROM roads WHERE  
  ST_DWithin(roads.the_geom2, ST_GeomFromText( 'POINT($xcoord_  
  $ycoord)', 9102003), 50) ORDER BY ST_Distance(roads.the_geom2,  
  ST_GeomFromText( 'POINT($xcoord_$ycoord)', 9102003)) limit 1;  
  
— find closest point on tiger data road segment  
SELECT ST_Line_Locate_Point(ST_LineFromText( 'LINESTRING(  
  $thestreet_wkt_extract_coords)', 9102003), ST_GeomFromText( 'POINT(  
  $xcoord_$ycoord)', 9102003));  
  
— calculate point location ratio along tiger data road segment  
SELECT st_astext(ST_Line_Interpolate_Point(ST_LineFromText( '  
  LINESTRING($thestreet_wkt_extract_coords)', 9102003),  
  $theratio));
```

The following example illustrates the powerful aggregate functions in PostGIS to create new datasets via union of polygons.

```
— Query 1: Union all counties of the county polygon data set "  
  us_counties" to create the data set "us_border". This operation  
  unions all individual data sets into one polygon encompassing
```

the area of the entire US. You can find the US County shape file in c:/class/data/layers/catalog/us_counties.shp and need to import it to PostGIS prior to performing the following geoprocessing task:

```
select st_union(geom)
into us_border
from us_counties ;
```

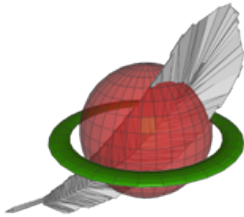
— Query 2: *Union all counties of the county polygon data set "us_counties" to create the data set "us_states". This operation unions all individual data sets and groups them by states (that is union all County polygon features into one polygon hat belong to the same state).*

```
select st_union(geom) , state_name
into us_states
from us_counties
group by state_name;
```


5 Appendix

5.1 OS Desktop GIS Tools

**file based
DBMS
light weight
portable**



**Mapnik as an
alternative
rendering engine in QGIS**

Spatialite is a spatial DBMS built on top of **SQLite**. Both formats are file based and thus are light weight and portable. The spatial components depend on the PROJ and GEOS libraries. Related tools include the **RasterLite** library to handle Raster data and **spatialite-gis** (a minimalistic GIS tool). Spatialite has the potential to replace shapefiles as a simple data exchange format. Starting with version 1.1 QGIS can read the format, support by OGR/GDAL was included since version 1.7.0.

Quantumnik is a plug-in for QGIS that enables the use of Mapnik as an alternative rendering engine in QGIS. This is useful because Mapnik is said to render the nicest looking maps of all web mapping engines. It also supports on-the-fly translation of QGIS layers and styles into Mapnik objects, enables the export of QGIS projects into Mapnik XML map files, and the loading of Mapnik XML for dynamic rendering in QGIS. The plug-in also requires a working installation of Mapnik. Download [here](#).

5.2 Basic Libraries - Using PROJ, OGR, and GDAL



GDAL - the Geospatial Data Abstraction Library (raster) and OGR - the OpenGIS Simple Features Reference Implementation (vector) are a set of tools for reading, writing and processing of raster and vector data sets in many different formats (tables are in the resource section). They are the base for many Desktop GIS systems, e.g. **ArcGIS**. OGR greatly extends the input formats MapServer can read: Oracle Spatial, ESRI Geodatabase (MDB), TIGER, and MapInfo. PROJ4 is a library for cartographic projection routines. It includes a stand alone projection utility *proj* and includes support for more than 2500 projections (epsg and esri lists).

OGR and GDAL are used in many Desktop GIS systems:

- **ArcGIS**
- **GRASS**
- **Quantum GIS**

About Proj4

The Proj4 project is hosted at <https://github.com/OSGeo/proj.4>.

The processing of data involving projection routines in OGR and GDAL are based on the Proj library. The Proj4 utilities include:

- Program proj.exe and library proj.dll
- Projection definition files epsg¹ and esri (in proj_lib dir)

Proj4 utilities program proj.exe and library proj.dll projection definition files

Using OGR

OGR includes a collection of utility programs that can be used on the command line to process vector data.

- **ogrinfo** <http://www.gdal.org/ogrinfo.html>
The ogrinfo utility lists various information about an OGR supported data source.
- **ogr2ogr** <http://www.gdal.org/ogr2ogr.html>
This program can be used to convert simple features data between file formats performing various operations during the process such as spatial or attribute selections, reducing the set of attributes, setting the output coordinate system or even reprojecting the features during translation.

collection of utility programs

- **ogrinfo**
- **ogr2ogr**
- **ogrindex**

¹EPSG - The European Petroleum Survey Group <http://www.epsg.org>

- **ogrindex** <http://www.gdal.org/ogrindex.html>

The ogrindex program can be used to create a tile index - a file containing a list of the identities of a bunch of other files along with their spatial extents. This is primarily intended to be used with the UMN MapServer for tiled access to layers using the OGR connection type.

**where is my
ArcCatalog?
get informa-
tion about a
shapefile**



**ogrinfo
<filename>**

Where is my ArcCatalog? To get meta information about a vector file you can use **ogrinfo**. For example on the command line we can find out what type of data is stored in counties2008.shp:

```
ogrinfo counties2008.shp
```

This gives us:

```
INFO: Open of 'counties2008.shp'
      using driver 'ESRI Shapefile' successful.
1: counties2008 (Polygon)
```

What do the following commands give us ?

```
ogrinfo counties2008.shp counties2008 --so
```

or connecting to PostGIS layer *counties*:

```
ogrinfo PG:"host=127.0.0.1 user=postgres password=
postgres dbname=ogis port=5432" counties --summary
```

Determine which spatial types (point, line polygon..) the following data sets are:

- data1.shp
- sunarea.tab
- whatisit.kml
- data4.shp



Type in **ogr2ogr** at the command line. This will list the options available with the utility. This gives us

```
Usage: ogr2ogr [—help—general] [—skipfailures] [—append]
      [—update] [—gt n]
      [—select field_list] [—where
        restricted_where]
      [—sql <sql statement>]
      [—spat xmin ymin xmax ymax] [—preserve_fid
        ] [—fid FID]
      [—a_srs srs_def] [—t_srs srs_def] [—s_srs
        srs_def]
      [—f format_name] [—overwrite] [[—dsco NAME
        =VALUE] ...]
      [—segmentize max_dist]
      dst_datasource_name src_datasource_name
      [—lco NAME=VALUE] [—nln name] [—nlt type]
      [layer [layer ...]]
```

—f format_name: output file format name, possible values are: , "ESRI Shapefile", "MapInfo File", "TIGER", "S57", "DGN", "Memory", "BNA", "CSV", "GML", "GPX", "KML", "GeoJSON", "Interlis 1", "Interlis 2", "G", "SQLite", "ODBC", "PostgreSQL", "MySQL", "Geoconcept" ...



- Use ogr2ogr to reproject one of the files listed above to a different projection of your choice.
Note: use -t_srs to specify the target projection.
- Produce a shapefile containing all uninhabited areas that are bigger than 500000000 (square feet) using ogr2ogr: Use a select statement with ogr2ogr. Base data is *uninhabited.shp*, the attribute containing the square feet values is "AREA".
- How can we at the same time convert the output result to a different format - like Geographic Markup Language (GML) or Census Tiger files (TIGER)?

- Create a batch file to convert 3 shapefiles to MapInfo files at once.

Using ogrtindex we can create tile indexes: A shapefile that contains the shapes of data that is split into small tiles which is good practice for larger datasets for use with MapServer.



gdalinfo
<filename>
gdal_translate

Using GDAL

Using gdalinfo we can get meta information about raster files.

- Use **gdalinfo** to retrieve information about "wa_shade_1km.tif". What information does this give us?
- Convert the tiff raster wa_shade_1km.tif into an ECW image using gdal_translate.
- Reproject wa_shade_1km.tif using gdal_translate into geographic projection(epsg:4326). Verify that the output projection is indeed geographic.

5.3 OS Desktop GIS Programs

OS software for almost any geospatial task

There is software that supports almost any task one can think of related to the processing, the analysis and the publishing of geospatial data. The following lists software for a variety of common applications in GIS, Remote Sensing and data analysis.

5.3.1 Geographic Resources Analysis Support System (GRASS)

GRASS - Powerful desktop GIS and image analysis for Remote Sensing



GRASS is comparable with Arc/Info

GRASS was originally started in 1982 by the US Army to implement functionality not available in other GIS systems at the time. After it was discontinued in 1995, its revival started in 1997 by Baylor University. Since 2001 the main development is housed at ITC. In 2008 the first standalone windows version was released in April with version 6.2.3. The system was originally started as a Raster GIS and Remote Sensing image analysis software but over the years extensive vector support was added. The capabilities are comparable with Arc/Info and in addition covers image analysis.

Table 14: OS Geospatial Desktop Programs

| Name | Type |
|--|--|
| Geographic Resources Analysis Support System (GRASS) | Desktop GIS |
| User friendly Desktop Internet GIS (uDig) and JGrass | |
| Quantum GIS (QGIS) and Open Ocean Map | |
| OpenJUMP | |
| gvSIG | |
| MapWindow | |
| The Generic Mapping Tools (GMT) | Raster map automation |
| Spatial Data Integrator | Extract Translate Load (ETL) |
| Open Source software Image Map (OSSIM) | Remote Sensing |
| The R Project for Statistical Computing (R) | Statistical software, scripting language |

Table 15: GRASS Project

| | |
|--------------------------------|---|
| Main supporter of GRASS | GRASS Development Team, ITC, Trento, Italy (since 2001) |
| Functionality | Comprehensive Desktop GIS and Image analysis |
| Operating systems | Unix/Linux, Windows (2008) |
| Project started | 1982, open source since 1999 |
| Implementation | C |
| OS libraries | OGR/GDAL |
| PostGIS support | Yes |
| License | GPL |

The Masters Thesis by Buchanan compares the functionality of Arc/Info 9.0 and GRASS 6.0 (see in references). One conclusion is that GRASS has some weaknesses in the ease of use of the user interface. However, this can be mediated by using QGIS (with the GRASS plug-in) or JGrass/uDig to remote

control its functionality.

5.3.2 User friendly Desktop Internet GIS (uDig) and JGrass

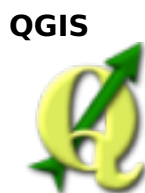


One of the goals of uDig, the "User-friendly Desktop Internet GIS" is to bring Internet resources to the desktop GIS user. It is a desktop GIS with editing capabilities based on the GeoTools library and has strong capabilities to connect to Internet resources such as WMS/WFS and other network resources. Thus it combines the strength of a Desktop viewer, editor and general GIS with access to web based services, databases and other resources. JGrass is a uDig based GIS for hydrological and geomorphological analysis. uDig is the UI for JGrass. The functions in JGrass are either GRASS functions rewritten in Java or wrapped GRASS analysis functions. To install it, use a package that includes uDig from the JGrass website.

Table 16: uDig Project

| | |
|-------------------------------|---|
| Main supporter of uDig | Refractions Research, Victoria, Canada |
| Type | Multilingual Desktop GIS |
| Functionality | Strong WMS/WFS support, Styled Layer Descriptor (SLD) support, vector editing, advanced map theming using ColorBrewer |
| Operating systems | Multi platform, Unix/Linux, Windows (2008) |
| Implementation | Java, using Eclipse for development |
| OS libraries | GeoTools |
| PostGIS support | Yes |
| License | LGPL |

5.3.3 Quantum GIS (QGIS)



QGIS was originally developed as a GIS viewing environment for the Linux desktop but is available for Solaris, Windows and Mac operating systems. Data sources for Quantum GIS can be PostGIS and shapefiles as vector data sources. Internally, QGIS uses the OGR library and therefore supports all OGR raster formats e.g. DEM, ArcGrid, ERDAS, SDTS, and GeoTIFF.

Table 17: Quantum GIS (QGIS) Project

| | |
|--------------------------------------|---|
| Main supporter of Quantum GIS | Gary Sherman and others |
| Type | Desktop GIS Viewer |
| Functionality | Can be used as a UI to GRASS GIS with GRASS Plug-in, Python bindings allow for programmatic interaction |
| Operating systems | Multi platform |
| Project started | 2002 |
| Implementation | C++, Depends on QT widget |
| OS libraries | OGR/GDAL |
| PostGIS support | Yes |
| License | GPL |

5.3.4 OpenJUMP - JUMP (JAVA Unified Mapping Platform)

JUMP is the "JUMP Unified Mapping Platform". It did experience some ups and downs during its development and as a result is available in a variety of "flavors" such as JUMP, OpenJUMP, and Kosmo. It was designed as an environment extensible platform into which spatial data conflation could be embedded (which was a major initial goal for its support by the British Columbia Ministry of Sustainable Resource Management). It provides functionality for viewing, and processing of spatial data. GML or "Geography Markup Language" is used for its main data format, but also can read and write shapefile, dxf and PostGIS vector data formats. Notably OpenJUMP is a great tool for editing, QA and correction of spatial data with problem solving capabilities.

OpenJUMP



OPEN JUMP

Table 18: OpenJUMP Project

| | |
|---|--|
| Main supporter of Vivid Solutions and Kosmo-SAIG
OpenJUMP | |
| Type | Desktop GIS in a Variety of "Flavors" (JUMP / OpenJUMP / Kosmo) |
| Functionality | Desktop GIS - Viewer - Analysis, powerful editing and QA environment, e.g. shape-file problem resolving capabilities |
| Operating systems | Multi platform |
| Project started | 2002 |
| Implementation | Java |
| OS libraries | JTS Topology Suite |
| PostGIS support | Yes |
| License | LGPL |

5.3.5 gvSIG - Generalitat Valencia Sistema de Información Geográfica

gvSIG is a project of the Spanish province of Valencia. The goals of the project are to provide an open source GIS that is platform independent and based on open source standards. Basically the capabilities should be comprehensive enough to replace ESRI's ArcView 3 desktop GIS. The user interface and functionalities of gvSIG are similar to ArcView 3, but in addition has capabilities to connect to Internet mapping services. The integration of a new labeling library "PAL" enables better labeling and brings gvSIG closer to having the same labeling support that is available in proprietary software such as ArcGIS. Another Java based project of the autonomous region of Extremadura called **Sextante** can be installed as a plug-in and offers more than 300 spatial analysis functions.

gvSIG



improved labeling support with the new PAL library - better suited for cartography

Table 19: gvSIG Project

| | |
|--------------------------------|---|
| Main supporter of gvSIG | Generalitat Valencia (GVA) - Province of Valencia, Spain |
| Type | Desktop GIS |
| Functionality | Multilingual Desktop GIS - Viewer - Analysis, analysis functions can be greatly extended when installing Sextante |
| Operating systems | Unix/Linux, Windows |
| Project started | 2003 |
| Implementation | Java |
| OS libraries | GeoTools and JTS |
| PostGIS support | Yes |
| License | GPL |

5.4 General Resources

5.4.1 Popular FOSS4G Licenses

GNU-GPL

Quoted from http://en.wikipedia.org/wiki/GNU_General_Public_License: The GNU General Public License (GNU GPL or simply GPL) is a widely used free software license, originally written by Richard Stallman for the GNU project. The GPL is the most popular and well-known example of the type of strong copyleft license that requires derived works to be available under the same copyleft. Under this philosophy, the GPL is said to grant the recipients of a computer program the rights of the free software definition and uses copyleft to ensure the freedoms are preserved, even when the work is changed or added to. This is in distinction to permissive free software licenses, of which the BSD licenses are the standard examples. The GNU Lesser General Public License (LGPL) is a modified, more permissive, version of the GPL, originally intended for some software libraries. There is also a GNU Free Documentation License, which was originally intended for use with documentation for GNU software, but has also been adopted for other uses, such as the Wikipedia project. The Affero General Public License (GNU AGPL) is a similar license with a focus on networking server software. The GNU AGPL is similar to the GNU General Public License, except that it additionally covers the use of the software over a computer network, requiring that the complete source code be made available to any network user of the AGPLed work, for example a web application. The Free Software Foundation recommends that this license is considered for any software that will commonly be run over the network.

LPGL

Quoted from <http://en.wikipedia.org/wiki/LGPL>: The GNU Lesser General Public License (formerly the GNU Library General Public License) or LGPL is a free software license published by the Free Software Foundation. It was designed as a compromise between the strong-copyleft GNU General Public License or GPL and permissive licenses such as the BSD licenses and the MIT License. The GNU Lesser General Public License was written in 1991 (and updated in 1999, and again in 2007) by Richard Stallman, with legal advice from Eben Moglen. The LGPL places copyleft restrictions on the program itself but does not apply these restrictions to other software that merely links with the program. There are, however, certain other restrictions on this software. The LGPL is primarily used for software libraries, although it is also used by some stand-alone applications, most notably Mozilla and OpenOffice.org.

MIT

Quoted from http://en.wikipedia.org/wiki/MIT_license: The MIT License is a free software license originating at the Massachusetts Institute of Technology (MIT), used by the MIT X Consortium. It is a permissive license, meaning that it permits reuse within proprietary software on the condition that the license is distributed with that software, and GPL-compatible, meaning that

the GPL permits combination and redistribution with software that uses the MIT License. According to the Free Software Foundation, the MIT License is more accurately called the X11 license, since MIT has used many licenses for software and the license was first drafted for the X Window System. Software packages that use the MIT License include Expat, PuTTY, Mono development platform class libraries, Ruby on Rails, Lua 5.0 onwards and the X Window System, for which the license was written. Some software packages dual license their products under the MIT License, such as older versions of the cURL library, which allowed recipients to choose either the Mozilla Public License or the MIT License.

BSD

Quoted from http://en.wikipedia.org/wiki/BSD_license: BSD licenses represent a family of permissive free software licenses. The original was used for the Berkeley Software Distribution (BSD), a Unix-like operating system for which the license is named. The original owners of BSD were the Regents of the University of California because BSD was first written at the University of California, Berkeley. The first version of the license was revised, and the resulting licenses are more properly called modified BSD licenses. Permissive licenses, sometimes with important differences pertaining to license compatibility, are referred to as "BSD-style licenses". Several BSD-like licenses, including the New BSD license, have been vetted by the Open Source Initiative as meeting their definition of open source. The licenses have few restrictions compared to other free software licenses such as the GNU General Public License or even the default restrictions provided by copyright, putting it relatively closer to the public domain.

Mozilla

Quoted from http://en.wikipedia.org/wiki/Mozilla_Public_License: The Mozilla Public License (MPL) is a free and open source software license. Version 1.0 was developed by Mitchell Baker when she worked as a lawyer at Netscape Communications Corporation and version 1.1 at the Mozilla Foundation.[3] The MPL is characterized as a hybridization of the modified BSD license and GNU General Public License. The MPL is the license for the Mozilla Application Suite, Mozilla Firefox, Mozilla Thunderbird and other Mozilla software. The MPL has been adapted by others as a license for their software, most notably Sun Microsystems, as the Common Development and Distribution License for OpenSolaris, the open source version of the Solaris 10 operating system, and by Adobe, as the license for its Flex product line.

5.4.2 Books

- Adams, T. & Jansen, M. (2010)** OpenLayers. Webentwicklung mit dynamischen Karten und Geodaten (in German). 344 pages. ISBN 978-3-937514-92-5
- Erle, S. & Gibson, R. & Walsh, J. (2005):** Mapping Hacks: Tips & Tools for Electronic Cartography (Hacks).
- Gratier, T., Spencer, P. & Hazzard, E. (2015):** OpenLayers 3 Beginner's Guide. <https://www.packtpub.com/web-development/openlayers-3-beginner%E2%80%99s-guide> Packt Publishing.
- Hall, G. B. & Leahy, M. G. (2008):** Open Source Approaches in Spatial Data Handling. Advances in Geographic Information Science. Springer.
- Hazzard, E. (2011)** OpenLayers 2.10. Beginner's Guide. 372 pages. ISBN 978-1-84951-412-5.
- Hengl, T. (2009)** A Practical Guide to Geostatistical Mapping. 293 pages. ISBN 978-90-9024981-0.
- Iacovella, S. & Youngblood, B. (2013):** GeoServer Beginner's Guide. 350 pages. ISBN 1849516685.
- Kropla, B. (2005):** Beginning MapServer: Open Source GIS Development.
- Maclean, M. (2014):** Leaflet Tips and Tricks. Interactive Maps Made Easy. Leanpub. [ebook](#)
- Mitchell, T. & Emde, A. & Christl, A. (2008):** Web Mapping Illustrated: Using Open Source GIS Toolkits. Revised German Ed., O'Reilly.
- Mitchell, T. (2005):** Web Mapping Illustrated: Using Open Source GIS Toolkits.
- Mitchell, T. & GDAL Developers (2013):** Geospatial Power Tools. GDAL/OGR Command Line Utilities. Locate Press. (available only as ebook so far)
- Neteler, M. & Mitasova, H. (2007):** Open Source GIS: A GRASS GIS Approach. 3rd Ed., Springer.
- Obe, R. & Hsu, L. (2011):** PostGIS in Action. Manning. 520 pages, [http://www.manning.com/obe/ISBN 9781935182269](http://www.manning.com/obe/ISBN%209781935182269)
- Obe, R. & Hsu, L. (2015):** PostGIS in Action. Second Edition. Manning. 600 pages, ISBN 9781617291395. <https://www.manning.com/books/postgis-in-action-second-edition>
- OSGEO (eds.) (2008):** Libro SIG: Sistemas de Informacion Geografica (Free GIS book in spanish), 800+ pages. http://wiki.osgeo.org/wiki/Libro_SIG
- Perez, A. S. (2012):** OpenLayers Cookbook. 300 pages. ISBN 1849517843
- Santiago, A. (2014):** The book of OpenLayers 3. Theory & Practice. Leanpub [ebook](#).

Sherman, G.E. (2012): The Geospatial Desktop. Open Source GIS & Mapping. Locate Press.

Sherman, G.E. (2014): The PyQGIS Programmers Guide. Extending Quantum GIS with Python. Locate Press.

5.4.3 Articles

The State of Open Source GIS, Version 2006. By Paul Ramsey, formerly Refrations Research, Victoria. 49 pages.

Comparison Of Geographic Information System Software (ArcGIS 9.0 And GRASS 6.0): Implementation And Case Study MS Thesis by Todd R. Buchanan, Fort Hays State University. 100 pages.

Geospatial Interoperability Return on Investment Study (2005). National Aeronautics and Space Administration, Geospatial Interoperability Office. 80 pages.

Current state of technology and potential of Smart Map Browsing (2007) in web browsers using the example of the Free web mapping application OpenLayers by Emanuel Schütze. 128 pages.

An Overview on Current Free and Open Source Desktop GIS Developments (2008) by Steiniger, S. & Bocher, E. Accepted for publication in Int. J. of Geographical Information Science, 1. Revision, Sept. 5th 2008. 24 pages. (included on class DVD)

5.4.4 General Web Sites

Table 20: FOSS4G web sites

| Name | URL |
|----------------------|---|
| Free GIS Project | http://www.freegis.org |
| Open source GIS list | http://opensourcegis.org |
| Map Tools | http://maptools.org |
| OSGeo | http://www.osgeo.org |

5.4.5 Open source utilities

Table 21: Open source utilities

| Name | URL |
|--|---|
| Simple Feature Library (OGR) | www.gdal.org/ogr |
| Geospatial Data Abstraction Library (GDAL) | www.gdal.org |
| GeoTools | http://geotools.org/ |
| PROJ4 (Cartographic Projections Library) | https://github.com/OSGeo/proj.4 |
| FWTools (utility collection) | http://fwtools.maptools.org |
| MapTiler (GUI to tile Rasters) | http://www.maptiler.org/ |
| GMT | http://gmt.soest.hawaii.edu |
| TerraLib | http://www.terralib.org |
| Spatial Data Integrator (ETL tool) | www.spatialdataintegrator.com |
| Geokettle (ETL tool) | http://www.spatialytics.org/projects/geokettle/ |
| Open Source Software Image Map (OSSIM) | http://www.ossim.org |
| The R Project for Statistical Computing | http://www.r-project.org |

5.4.6 Web Mapping Software

Table 22: Web Mapping Software

| Name | URL |
|---------------------------------------|---|
| MapServer | http://mapserver.org |
| MapServer and Node.js | https://github.com/pagameba/node-mapserver |
| GeoServer | http://geoserver.org |
| Mapnik | http://www.mapnik.org |
| Mapnik and Node.js | https://github.com/mapnik/node-mapnik |
| Deegree | http://www.deegree.org |
| MapGuide | http://mapguide.osgeo.org |
| OpenLayers | www.openlayers.org |
| Polymaps (JS, SVG) | http://polymaps.org/ |
| Leaflet (lightweight JS Library) | http://leaflet.cloudmade.com |
| Modestmaps (lightweight JS Library) | http://modestmaps.com |
| Mapstraction (lightweight JS Library) | http://mapstraction.com |
| Mapbender | www.mapbender.org |
| MapFish | http://www.mapfish.org |
| GeoDjango (now merged into Django) | https://www.djangoproject.com/ |
| D3 (Data-Driven Documents) | http://d3js.org/ |

5.4.7 Additional tools for use with OpenLayers

Table 23: OpenLayers additional tools

| Name | URL |
|---|---|
| i2maps (geocomputing environment to use with OpenLayers) | https://github.com/NCG/i2maps |
| Acidmaps (interpolated map images with Geoserver/OpenLayers) | http://ams.xoomcode.com/ |
| OpenLayers Editor (from Geops) | https://github.com/geops/ole |

5.4.8 Map Tiling engines

Table 24: Map Tiling engines

| Name | URL |
|--|--|
| TileCache | www.tilecache.org |
| GeoWebCache | http://geowebcache.org |
| MapProxy (tiles, server proxy, security) | http://mapproxy.org |
| MapCache | http://www.mapserver.org/
mapcache/ |
| Tilemill (map styling) | http://mapbox.com/tilemill |

5.4.9 Databases and Additional Software

Table 25: Databases and Additional Software

| Name | URL |
|---|--|
| PostgreSQL | www.postgresql.org |
| PostGIS | http://postgis.refractory.net |
| Spatialite | http://www.gaia-gis.it/spatialite |
| Geocouch (document based db) | https://github.com/couchbase/geocouch |
| Proj4js | http://proj4js.org |
| FeatureServer | http://featureserver.org/ |
| WAS,WSS, and WSC (52N Security Community) | http://52north.org |
| GeoPrisma (security) | http://geoprisma.org |
| GeoShield (security) | http://sites.google.com/site/
geoshieldproject |
| GeoExt | http://www.geoext.org |
| Ext JS | http://www.sencha.com |

5.4.10 Desktop GIS List

Table 26: Desktop GIS

| Name | URL |
|-----------|---|
| MapWindow | www.mapwindow.org |
| OpenJump | http://openjump.org |
| GRASS | http://grass.osgeo.org |
| QGIS | http://www.qgis.org |
| gvSIG | http://www.gvsig.org |
| gvSIG CE | http://gvsigce.org/ |
| uDig | http://udig.refractions.net |
| TerraLib | http://www.terralib.org |

5.4.11 GIS Samplers, Live DVDs, and Compilations

Table 27: GIS Samplers, Live DVDs, and Compilations

| Name | URL |
|---|---|
| OSGEO Live DVD image | http://live.osgeo.org |
| GIS Virtual Machine | http://www.gisvm.com |
| MapServer for Windows | http://www.ms4w.com/ |
| OSGEO for Windows | http://trac.osgeo.org/osgeo4w |
| Ubuntu GIS | https://wiki.ubuntu.com/UbuntuGIS |
| OpenGeo Community Suite (PostGIS, GeoServer, GeoWebCache, OpenLayers, GeoExt) | http://boundlessgeo.com/solutions/opengeo-suite/download/ |

5.4.12 Conferences

Table 28: Free and Open Source Software for Geospatial (FOSS4G)

| Year | Name | Dates | URL |
|------|-------------------------|-----------------------|---|
| 2016 | Bonn, Germany | August 24-26, 2016 | http://2016.foss4g.org |
| 2015 | Seoul, Korea | September 14-19, 2015 | http://2015.foss4g.org |
| 2014 | Portland, USA | September 8-13, 2014 | http://2014.foss4g.org |
| 2013 | Nottingham, UK | September 17-21, 2013 | http://2013.foss4g.org |
| 2011 | Denver, USA | September 12-16, 2011 | http://2011.foss4g.org |
| 2010 | Barcelona, Spain | September 6-9, 2010 | http://2010.foss4g.org |
| 2009 | Sydney, Australia | October 20-23, 2009 | http://2009.foss4g.org |
| 2008 | Cape Town, South Africa | Sep/Oct 2008 | www.foss4g2008.org |
| 2007 | Victoria, Canada | September 2007 | http://www.foss4g2007.org |
| 2006 | Lausanne, Switzerland | September 2006 | http://www.foss4g2006.org |

5.4.13 User Groups

Cascadia Users of Geospatial Open Source: Seattle based GIS user group. Monthly meeting every 3rd Wednesday, 6:00 pm. <http://www.cugos.org>

5.4.14 Mailing Lists

Table 29: Mailing Lists

| List | email or URL |
|-----------------------------|--|
| List archives | http://osgeo-org.1560.x6.nabble.com/ |
| MapServer for Windows(ms4w) | ms4w-users@lists.ms4w.com |
| MapServer | mapserver-users@lists.osgeo.org |
| OpenLayers | users@OpenLayers.org |
| PostGIS | postgis-users@postgis.refractory.net |
| PostgreSQL | pgsql-general@postgresql.org |
| GeoServer | geoserver-users@lists.sourceforge.net |
| Mapbender | mapbender_users@lists.osgeo.org |
| TileCache | tilecache@OpenLayers.org |
| MapFish | users@mapfish.org |

5.4.15 Internet Relay Chat (IRC)

URL: <irc://freenode.net> Channels for Mapbender, PostGIS, PostgreSQL, MapServer, GeoServer, OpenLayers, and MapFish. One client to use for IRC is Chatzilla. [Chatzilla is an add-on](#) to

Mozilla Firefox. More information [here](#)

5.4.16 Meeting the Pros

Table 30: OS GIS Blog aggregator

| Name | Focus | blog/website/info |
|--------------------------------|-----------|---|
| Planet OSGeo (Blog Aggregator) | All mixed | http://planet.osgeo.org |

5.4.17 Installing a Web GIS Server from the Ubuntu GIS Repository

Note that this is not a comprehensive guide but in short one needs to add the [Ubuntu repositories](#) to the server apt list, which on a typical system is located at `/etc/apt/sources.list`. It is a good idea to determine exactly which operating system version the server is running (if you not already aware of it) for example see below you are running Ubuntu *Hardy Heron*.

```
lsb_release -c
```

```
> hardy
```

% this means you are running the **hardy** version of Ubuntu

The following lines need to be added to the `sources.list` using a text editor:

```
deb http://ppa.launchpad.net/ubuntugis/ppa/ubuntu hardy main
deb-src http://ppa.launchpad.net/ubuntugis/ppa/ubuntu hardy main
```

The next step is to authenticate the repository (314DF160 is the OpenPGP key of the Ubuntu GIS repository):

```
sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 314
DF160
```

Starting with Ubuntu 9.10 (Karmic Koala) the Launchpad PPA (Personal Package Archive) repositories can also be added via command line (without manually editing the `sources.list` file and no need to authenticate the repository):

```
sudo apt-get install python-software-properties
sudo add-apt-repository ppa:ubuntugis/ppa
```

After updating your apt utility software package list you will be ready to start the GIS installation

```
sudo apt-get update
```

The following is an example for an installation sequence to set-up a Web Mapping enabled GIS Server. You see that all available packages can be installed (and also later be removed if needed) not only GIS components.

```
# install apache version 2
```

```
sudo apt-get install apache2
```

```
# install php
```

```
# see \url{https://help.ubuntu.com/8.04/serverguide/C/php5.html}
```

```
sudo apt-get install php5 libapache2-mod-php5
```

```
sudo apt-get install php5-cli
```

```
sudo apt-get install php5-cgi
```

```
sudo apt-get install php5-mysql # for MySQL
```

```
sudo apt-get install php5-pgsql # for PostgreSQL
```

```
# install gdal
```

```
sudo apt-get install gdal-bin
```

```
# install proj
sudo apt-get install proj
```

```
# install PostgreSQL and PostGIS
sudo apt-get install postgresql-8.4.4
apt-get install postgresql-server-dev-8.4 libpq-dev
apt-get install libgeos-dev
sudo apt-get install postgis-1.5
```

At this point you may take care of initial PostgreSQL housekeeping tasks (but yo can also save that for later)

```
# To set a password for the postgres database user this on the
command line
```

```
psql -c"ALTER user postgres WITH PASSWORD 'postgres'"
```

```
# Switch to postgres user
```

```
sudo su - postgres
```

```
# Create the plpgsql language inside the database template:
```

```
createlang -d template_postgis plpgsql
```

Continue with the installation of Mapserver and Tilecache:

```
# install mapserver
sudo apt-get install cgi-mapserver
sudo apt-get install mapserver-bin
```

```
# install python image library
sudo apt-get install libjpeg62 libjpeg62-dev zlib1g-dev
```

```
# install tilecache
sudo apt-get install tilecache
```

5.4.18 GDAL and OGR Formats

Table 31: GDAL Raster Formats

| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default |
|--|---------|----------|---------|----------|----------------------|
| Arc/Info ASCII Grid | AAIGrid | Yes | Yes | 2GB | Yes |
| ACE2 | ACE2 | No | Yes | – | Yes |
| ADRG/ARC Digitized Raster Graphics (.gen/.thf) | ADRG | Yes | Yes | – | Yes |
| Arc/Info Binary Grid (.adf) | AIG | No | Yes | – | Yes |
| AIRSAR Polarimetric | AIRSAR | No | No | – | Yes |
| Magellan BLX Topo (.blx, .xlb) | BLX | Yes | Yes | – | Yes |
| Bathymetry Attributed Grid (.bag) | BAG | No | Yes | 2GiB | No, needs libhdf5 |
| Microsoft Windows Device Independent Bitmap (.bmp) | BMP | Yes | Yes | 4GiB | Yes |
| BSB Nautical Chart Format (.kap) | BSB | No | Yes | – | Yes, can be disabled |
| VTP Binary Terrain Format (.bt) | BT | Yes | Yes | – | Yes |
| CEOS (Spot for instance) | CEOS | No | No | – | Yes |
| DRDC COASP SAR Processor Raster | COASP | No | No | – | Yes |
| TerraSAR-X Complex SAR Data Product | COSAR | No | No | – | Yes |
| Convair PolGASP data | CPG | No | Yes | – | Yes |
| USGS LULC Composite Theme Grid | CTG | No | Yes | – | Yes |
| Spot DIMAP (meta-data.dim) | DIMAP | No | Yes | – | Yes |
| ELAS DIPEX | DIPEX | No | Yes | – | Yes |
| DODS / OPeNDAP | DODS | No | Yes | – | No, needs libdap |
| First Generation USGS DOQ (.doq) | DOQ1 | No | Yes | – | Yes |
| New Labelled USGS DOQ (.doq) | DOQ2 | No | Yes | – | Yes |

GDAL Raster Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default |
|--|------------------|----------|---------|-----------|-------------------------------------|
| Military Elevation Data (.dt0, .dt1, .dt2) | DTED | Yes | Yes | – | Yes |
| Arc/Info Export GRID | E00 E00GRID | No | Yes | – | Yes |
| ECRG Table Of Contents (TOC.xml) | ECRGTOC | No | Yes | – | Yes |
| ERDAS Compressed Wavelets (.ecw) | ECW | Yes | Yes | | No, needs ECW SDK |
| ESRI .hdr Labelled | EHdr | Yes | Yes | No limits | Yes |
| Erdas Imagine Raw | EIR | No | Yes | – | Yes |
| NASA ELAS | ELAS | Yes | Yes | – | Yes |
| ENVI .hdr Labelled Raster | ENVI | Yes | Yes | No limits | Yes |
| Epsilon - Wavelet compressed images | EPSILON | Yes | No | – | No, needs EPSILON library |
| ERMapper (.ers) | ERS | Yes | Yes | | Yes |
| Envisat Image Product (.n1) | ESAT | No | No | – | Yes |
| EOSAT FAST Format | FAST | No | Yes | – | Yes |
| FIT | FIT | Yes | No | – | Yes |
| FITS (.fits) | FITS | Yes | No | – | No, needs libcfitsio |
| Fuji BAS Scanner Image | FujiBAS | No | No | – | Yes |
| Generic Binary (.hdr Labelled) | GENBIN | No | No | – | Yes |
| Oracle Spatial Geo-Raster | GEORASTER | Yes | Yes | – | No, needs Oracle client libraries |
| GSat File Format | GFF | No | No | – | Yes |
| Graphics Interchange Format (.gif) | GIF | Yes | No | 2GB | Yes (internal GIF library provided) |
| WMO GRIB1/GRIB2 (.grb) | GRIB | No | Yes | 2GB | Yes, can be disabled |
| GMT Compatible netCDF | GMT | Yes | Yes | 2GB | No, needs libnetcdf |
| GRASS Rasters | GRASS | No | Yes | – | No, needs libgrass |
| GRASS ASCII Grid | GRASS ASCII Grid | No | Yes | – | Yes |

| GDAL Raster Formats continued from previous page | | | | | | |
|--|------------|----------|---------|---|--|--|
| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default | |
| Golden Software ASCII Grid | GSAG | Yes | No | – | Yes | |
| Golden Software Binary Grid | GSBG | Yes | No | 4GB | Yes | |
| Golden Software Surfer 7 Binary Grid | GS7BG | No | No | 4GiB | Yes | |
| GSC Geogrid | GSC | Yes | No | – | Yes | |
| TIFF / BigTIFF / GeoTIFF (.tif) | GTiff | Yes | Yes | 4GiB for classical TIFF / No limits for BigTIFF | Yes (internal libtiff and libgeotiff provided) | |
| NOAA .gtx vertical datum shift | GTX | Yes | Yes | | Yes | |
| GXF - Grid eXchange File | GXF | No | Yes | 4GiB | Yes | |
| Hierarchical Data Format Release 4 (HDF4) | HDF4 | Yes | Yes | 2GiB | No, needs libdf | |
| Hierarchical Data Format Release 5 (HDF5) | HDF5 | No | Yes | 2GiB | No, needs libhdf5 | |
| HF2/HFZ heightfield raster | HF2 | Yes | Yes | - | Yes | |
| Erdas Imagine (.img) | HFA | Yes | Yes | No limits2 | Yes | |
| Image Display and Analysis (WinDisp) | IDA | Yes | Yes | 2GB | Yes | |
| ILWIS Raster Map (.mpr,.mpl) | ILWIS | Yes | Yes | – | Yes | |
| Intergraph Raster | INGR | Yes | Yes | 2GiB | Yes | |
| USGS Astrogeology ISIS cube (Version 2) | ISIS2 | Yes | Yes | – | Yes | |
| USGS Astrogeology ISIS cube (Version 3) | ISIS3 | No | Yes | – | Yes | |
| JAXA PALSAR Product Reader (Level 1.1/1.5) | JAXAPALSAR | No | No | – | Yes | |
| Japanese DEM (.mem) | JDEM | No | Yes | – | Yes | |
| JPEG JFIF (.jpg) | JPEG | Yes | Yes | 4GB | Yes (internal libjpeg provided) | |
| JPEG-LS | JPEGLS | Yes | No | – | No, needs CharLS library | |

GDAL Raster Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default |
|--|-------------------------|-----------------|----------------|-----------------|---------------------------------|
| JPEG2000 (.jp2, .j2k) | JPEG2000 | Yes | Yes | 2GiB | No, needs libjasper |
| JPEG2000 (.jp2, .j2k) | JP2ECW | Yes | Yes | 500MB | No, needs ECW SDK |
| JPEG2000 (.jp2, .j2k) | JP2KAK | Yes | Yes | No limits | No, needs Kakadu library |
| JPEG2000 (.jp2, .j2k) | JP2MrSID | Yes | Yes | | No, needs MrSID SDK |
| JPEG2000 (.jp2, .j2k) | JP2Open JPEG | Yes | Yes | | No, needs OpenJPEG library (v2) |
| JPIP (based on Kakadu) | JPIPKAK | No | Yes | | No, needs Kakadu library |
| KMLSUPEROVERLAY | KML
SUPER
OVERLAY | Yes | Yes | | Yes |
| NOAA Polar Orbiter Level 1b Data Set (AVHRR) | L1B | No | Yes | – | Yes |
| Erdas 7.x .LAN and .GIS | LAN | No | Yes | 2GB | Yes |
| FARSITE v.4 LCP Format | LCP | No | Yes | No limits | Yes |
| Daylon Leveller Height-field | Leveller | No | Yes | 2GB | Yes |
| NADCON .los/.las Datum Grid Shift | LOSLAS | No | Yes | | Yes |
| In Memory Raster | MEM | Yes | Yes | | Yes |
| Vexcel MFF | MFF | Yes | Yes | No limits | Yes |
| Vexcel MFF2 | MFF2 (HKV) | Yes | Yes | No limits | Yes |
| MG4 Encoded Lidar | MG4Lidar | No | Yes | – | No, needs LI-DAR SDK |
| Multi-resolution Seamless Image Database | MrSID | No | Yes | – | No, needs MrSID SDK |
| Meteosat Second Generation | MSG | No | Yes | | No, needs msg library |
| EUMETSAT Archive native (.nat) | MSGN | No | Yes | | Yes |
| NLAPS Data Format | NDF | No | Yes | No limits | Yes |

GDAL Raster Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default |
|---|----------------|----------|---------|-----------|-----------------------------------|
| NITF (.ntf, .nsf, .gn?, .hr?, .ja?, .jg?, .jn?, .lf?, .on?, .tl?, .tp?, etc.) | NITF | Yes | Yes | 10GB | Yes |
| NetCDF | netCDF | Yes | Yes | 2GB | No, needs libnetcdf |
| NTv2 Datum Grid Shift | NTv2 | Yes | Yes | | Yes |
| Northwood/ VerticalMapper Classified Grid Format .grc/.tab | NWT_GRC | No | Yes | – | Yes |
| Northwood/ VerticalMapper Numeric Grid Format .grd/.tab | NWT_GRD | No | Yes | – | Yes |
| OGDI Bridge | OGDI | No | Yes | – | No, needs OGDI library |
| OZI OZF2/OZFX3 | OZI | No | Yes | – | No |
| PCI .aux Labelled | PAux | Yes | No | No limits | Yes |
| PCI Geomatics Database File | PCIDSK | Yes | Yes | No limits | Yes |
| PCRaster | PCRaster | Yes | Yes | | Yes (internal libcsf provided) |
| Geospatial PDF | PDF | No | Yes | – | No, needs libpoppler or libpodofo |
| NASA Planetary Data System | PDS | No | Yes | – | Yes |
| Portable Network Graphics (.png) | PNG | Yes | No | | Yes (internal libpng provided) |
| PostGIS Raster (previously WKTRaster) | PostGIS Raster | No | Yes | – | No, needs PostgreSQL library |
| Netpbm (.ppm,.pgm) | PNM | Yes | No | No limits | Yes |
| R Object Data Store | R | Yes | No | – | Yes |
| Rasdaman | RASDAMAN | No | No | No limits | No (needs raslib) |
| Rasterlite - Rasters in SQLite DB | Rasterlite | Yes | Yes | – | No (needs OGR SQLite driver) |

GDAL Raster Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Max.size | Compiled by default |
|--|-------------|-----------------|----------------|-----------------|--|
| Swedish Grid RIK (.rik) | RIK | No | Yes | 4GB | Yes (internal zlib is used if necessary) |
| Raster Matrix Format (*.rsw, .mtw) | RMF | Yes | Yes | 4GB | Yes |
| Raster Product Format/RPF (CADRG, CIB) | RPFTOC | No | Yes | – | Yes |
| RadarSat2 XML (product.xml) | RS2 | No | Yes | 4GB | Yes |
| Idrisi Raster | RST | Yes | Yes | No limits | Yes |
| SAGA GIS Binary format | SAGA | Yes | Yes | – | Yes |
| SAR CEOS | SAR_CEOS | No | Yes | – | Yes |
| ArcSDE Raster | SDE | No | Yes | – | No, needs ESRI SDE |
| USGS SDTS DEM (*.CATD.DDF) | SDTS | No | Yes | – | Yes |
| SGI Image Format | SGI | Yes | Yes | – | Yes |
| Snow Data Assimilation System | SNODAS | No | Yes | – | Yes |
| Standard Raster Product (ASRP/USRP) | SRP | No | Yes | 2GB | Yes |
| SRTM HGT Format | SRTMHGT | Yes | Yes | – | Yes |
| Terragen Heightfield (.ter) | TERRAGEN | Yes | No | – | Yes |
| EarthWatch/DigitalGlobe .TIL | TIL | No | No | – | Yes |
| TerraSAR-X Product | TSX | Yes | No | – | Yes |
| USGS ASCII DEM / CDED (.dem) | USGSDEM | Yes | Yes | – | Yes |
| GDAL Virtual (.vrt) | VRT | Yes | Yes | – | Yes |
| OGC Web Coverage Service | WCS | No | Yes | – | No, needs libcurl |
| WEBP | WEBP | Yes | No | – | No, needs libwebp |
| OGC Web Map Service | WMS | No | Yes | – | No, needs libcurl |
| X11 Pixmap (.xpm) | XPM | Yes | No | – | Yes |
| ASCII Gridded XYZ | XYZ | Yes | Yes | – | Yes |
| ZMap Plus Grid | ZMap | Yes | Yes | – | Yes |

Table 32: OGR Vector Formats

| Format Name | Code | Creation | Georef. | Compiled by default |
|--------------------------------|--------------------|-----------------|----------------|---|
| Aeronav FAA files | AeronavFAA | No | Yes | Yes |
| ESRI ArcObjects | ArcObjects | No | Yes | No, needs ESRI ArcObjects |
| Arc/Info Binary Coverage | AVCBin | No | Yes | Yes |
| Arc/Info .E00 (ASCII) Coverage | AVCE00 | No | Yes | Yes |
| Atlas BNA | BNA | Yes | No | Yes |
| AutoCAD DXF | DXF | Yes | No | Yes |
| Comma Separated Value (.csv) | CSV | Yes | No | Yes |
| CouchDB / GeoCouch | CouchDB | Yes | Yes | No, needs libcurl |
| DODS/OPeNDAP | DODS | No | Yes | No, needs libdap |
| EDIGEO | EDIGEO | No | Yes | Yes |
| ESRI FileGDB | FileGDB | Yes | Yes | No, needs FileGDB API library |
| ESRI Personal Geo-Database | PGeo | No | Yes | No, needs ODBC library |
| ESRI ArcSDE | SDE | No | Yes | No, needs ESRI SDE |
| ESRI Shapefile | ESRI Shapefile | Yes | Yes | Yes |
| FMEObjects Gateway | FMEObjects Gateway | No | Yes | No, needs FME |
| GeoJSON | GeoJSON | Yes | Yes | Yes |
| Géoconcept Export | Geoconcept | Yes | Yes | Yes |
| Geomedia .mdb | Geomedia | No | No | No, needs ODBC library |
| GeoRSS | GeoRSS | Yes | Yes | Yes (read support needs libexpat) |
| Google Fusion Tables | GFT | Yes | Yes | No, needs libcurl |
| GML | GML | Yes | Yes | Yes (read support needs Xerces or libexpat) |
| GMT | GMT | Yes | Yes | Yes |
| GPSTransfer | GPSTransfer | Yes | Yes | Yes (needs GPSTransfer and GPX driver) |
| GPX | GPX | Yes | Yes | Yes (read support needs libexpat) |
| GRASS | GRASS | No | Yes | No, needs libgrass |
| GPSTrackMaker (.gtm, .gtz) | GPS TrackMaker | Yes | Yes | Yes |
| Hydrographic Format | HTF | No | Yes | Yes |

OGR Vector Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Compiled by default |
|--|-------------------------------|----------|---------|---|
| Idrisi Vector (.VCT) | Idrisi | No | Yes | Yes |
| Informix DataBlade | IDB | Yes | Yes | No, needs Informix DataBlade |
| INTERLIS | "Interlis 1" and "Interlis 2" | Yes | Yes | No, needs Xerces (INTERLIS model reading needs ili2c.jar) |
| INGRES | INGRES | Yes | No | No, needs INGRESS |
| KML | KML | Yes | Yes | Yes (read support needs libexpat) |
| LIBKML | LIBKML | Yes | Yes | No, needs libkml |
| Mapinfo File | MapInfo File | Yes | Yes | Yes |
| Microstation DGN | DGN | Yes | No | Yes |
| Access MDB (PGeo and Geomedia capable) | MDB | No | Yes | No, needs JDK/JRE |
| Memory | Memory | Yes | Yes | Yes |
| MySQL | MySQL | No | Yes | No, needs MySQL library |
| NAS - ALKIS | NAS | No | Yes | No, needs Xerces |
| Oracle Spatial | OCI | Yes | Yes | No, needs OCI library |
| ODBC | ODBC | No | Yes | No, needs ODBC library |
| MS SQL Spatial | MSSQLSpatial | Yes | Yes | No, needs ODBC library |
| OGDI Vectors (VPF, VMAP, DCW) | OGDI | No | Yes | No, needs OGDI library |
| OpenAir | OpenAir | No | Yes | Yes |
| PCI Geomatics Database File | PCIDSK | No | No | Yes, using internal PCIDSK SDK (from GDAL 1.7.0) |
| PDS | PDS | No | Yes | Yes |
| PGDump | PostgreSQL SQL dump | Yes | Yes | Yes |
| PostgreSQL/PostGIS | PostgreSQL/PostGIS | Yes | Yes | No, needs PostgreSQL client library (libpq) |
| EPIInfo .REC | REC | No | No | Yes |
| S-57 (ENC) | S57 | No | Yes | Yes |
| SDTS | SDTS | No | Yes | Yes |
| Norwegian SOSI Standard | SOSI | No | Yes | No, needs FYBA library |
| SQLite/Spatialite | SQLite | Yes | Yes | No, needs libsqlite3 or libspatialite |
| SUA | SUA | No | Yes | Yes |
| SVG | SVG | No | Yes | No, needs libexpat |
| UK .NTF | UK. NTF | No | Yes | Yes |
| U.S. Census TIGER/Line | TIGER | No | Yes | Yes |

OGR Vector Formats continued from previous page

| Long Format Name | Code | Creation | Georef. | Compiled by default |
|---|-------------|-----------------|----------------|----------------------------|
| VFK data | VFK | No | Yes | Yes |
| VRT - Virtual Datasource | VRT | No | Yes | Yes |
| OGC WFS (Web Feature Service) | WFS | Yes | Yes | No, needs libcurl |
| X-Plane/Flighgear
aero-nautical data | XPLANE | No | Yes | Yes |

5.5 Sample Map files for MapServer

A basic map file

```
MAP #Start of map file
NAME "Washington Counties"
EXTENT 600000 -800000 2750000 1000000 # bounding box (map
    extent)
SIZE 600 300 # size of output
    map in pixels
LAYER # Data Layer object
    NAME Counties # name of layer used as a reference
        by MapServer
    TYPE POLYGON # spatial type
    STATUS DEFAULT # status (on/off/default)
    DATA "../layers/counties2008" # input data source (
        shapefile in this case)
    CLASS # classification
        OUTLINECOLOR 100 100 100 # color for outline
            boundary
    END # Class END
END # Data Layer END
END # Map FileD
END
```

An intermediate map file

```
MAP # Start of map file
NAME "alliance"
EXTENT 630000 -540000 2700000 780000 # Bounding box
    coordinates in Projection set below (epsg:2285)
FONTSET "../../../fonts/fonts.txt" # Fonts files for
    true type fonts
SIZE 400 400 # output size of the
    map in pixels
UNITS FEET # map units
IMAGECOLOR 204 221 255 #
IMAGETYPE PNG # output file type
INTERLACE ON # enable interlacing
    for png image (faster loading)

    PROJECTION # start
        projection of the map
```



```

        "init=epsg:2285"                # epsg
        :2285 from epsg coming with Proj4 is
        Washington Sate Plane North
    END                                # projection
    end tag

LEGEND                                #
    keysize 15 15                      # size of legend
    boxes
    keyspacing 5 2                     # spacing
    between legend boxes
END                                    # legend end tag

# Start of web interface definition
#####

WEB                                    # start general web
    definitions
        METADATA                      # Metadata tag
            start
            "wms_title" "Layers"       # name of WMS
            service, below location of the configuration map
            file
            "wms_onlineresource" "http://localhost/mapserver/
            mapserv.exe?map=d:/web/mapdata/projects/alliance/
            alliance.map"
            "wms_srs" "epsg:2285 epsg:4326" # list of spatial
            reference systems, multiple entries enable
            multiple outputs to be requested
            "wms_feature_info_mime_type" "text/html" # format of
            the GetFeatureInfo request when identifying
            feature attributes
            "ows_enable_request" "*"    # for MapServer version
            >= 6 WMS output needs to be enabled explicitly
        END                            # end Metadata
    tag
END                                    # end tag general
    web definitions

# Start of symbol definitions
#####

```

```

SYMBOL                                     # start vector
    symbol definitions
        NAME 'point'                       # name of
            symbol
        TYPE ELLIPSE                       # type of
            symbol
        POINTS 1 1 END                     # size
        FILLED TRUE                         # filled or
            not filled
END                                         # end vector symbol
    definitions

# Start of layer definitions
#####

LAYER                                     # start layer tag
    NAME "States"                         # layer name
    TYPE POLYGON                          # layer type (
        POINT, LINE, POLYGON)
    STATUS ON                             # status (ON,
        OFF, DEFAULT)
    DATA "../wa_states"                  #
    TEMPLATE "templates/wa_states_template.html" #
        location of query template in relative
        relation to map file location
    CLASS
        # start class
        definition
            COLOR 240 240 230
                # Color (in this case
                polygon fill)
        END                                # end class
    definitions
    DUMP true                             #
        parameter that is required to enable WMS
        output
    PROJECTION                             # start
        projection of the layer
            "init=epsg:2285"               # epsg
            :2285 from epsg coming with Proj4 is
            Washington Sate Plane North
    END                                    # layer
        projection end tag

```

```
METADATA                                # Metadata tag
    start
"wms_title" "States"                    # name of this
    individual WMS layer
"wms_srs" "epsg:2285 epsg:4326" # list of spatial
    reference systems this layer is available in
"wms_feature_info_mime_type" "text/html" #
    format of the GetFeatureInfo request
"ows_enable_request" "*" # for MapServer version
    >= 6 WMS output needs to be enabled explicitly
END                                     # Metadata
    tag start
END                                     # end layer tag

#####
END                                     # End of map file
```

A more complex map file with a PostGIS layer

```
MAP # Start of map file
NAME "alliance_all2"
EXTENT 630000 -540000 2700000 780000
FONTSET "c:/class/data/mapfiles/fonts/fonts.list"
SIZE 400 400
MAXSIZE 16384
UNITS FEET
IMAGECOLOR 204 221 255
IMAGETYPE PNG_1
# IMAGETYPE PNG
# INTERLACE ON

OUTPUTFORMAT
  NAME 'AGG'
  DRIVER AGG/PNG
  IMAGEMODE RGB
  FORMATOPTION "INTERLACE=ON"
END

OUTPUTFORMAT
  NAME 'AGG_Q'
  DRIVER AGG/PNG
  IMAGEMODE RGB
  FORMATOPTION "QUANTIZE_FORCE=ON"
  FORMATOPTION "QUANTIZE_DITHER=OFF"
  FORMATOPTION "QUANTIZE_COLORS=256"
  FORMATOPTION "INTERLACE=ON"
END

OUTPUTFORMAT
  NAME 'PNG_1'
  DRIVER "GD/PNG"
  MIMETYPE "image/png"
  IMAGEMODE PC256
  EXTENSION "png"
END

#### Start of projection definition #####
# NAD83 / Washington North (ftUS) epsg <2285>
# PROJECTION
#   "proj=lcc"
#   "lat_1=48.73333333333333"
```

```
# "lat_2=47.5"
# "lat_0=47"
# "lon_0=-120.83333333333333"
# "x_0=500000.0001016001"
# "y_0=0"
# "ellps=GRS80"
# "datum=NAD83"
# "to_meter=0.3048006096012192"
# END

    PROJECTION
        "init=epsg:2285"
    END

LEGEND
    keysize 16 12
    keyspacing 5 2
    TRANSPARENT ON
    LABEL
        PARTIALS FALSE
        TYPE TRUE
        FONT 'verdana'
        SIZE 8
        COLOR 1 1 1
    END
END

# Start of web interface definition
#####

WEB
    HEADER "templates/general_header_template.html"
    FOOTER "templates/general_footer_template.html"
    ERROR "http://alliance.terrakis.net/
        mapserver_error.html"
    EMPTY "http://alliance.terrakis.net/query_empty.
        html"
    LOG "/var/www/sites/mapbender/log/
        mapserver_oca.log"
    METADATA
        "wms_title" "alliance_all"
        "wms_onlineresource" "http://alliance.terrakis.net/
            cgi-bin/mapserv.fcgi?map=/var/www/mapdata/
            projects/alliance/alliance2.map"
```

```
"wms_srs" "epsg:2285 epsg:4326"
"wms_feature_info_mime_type" "text/html"
"ows_enable_request" "*" # for MapServer version
    >= 6 WMS output needs to be enabled explicitly
END
END

# Start of symbol definitions
#####
SYMBOL
    NAME 'point'
    TYPE ELLIPSE
    POINTS 1 1 END
    FILLED TRUE
END

# Start of layer definitions
#####

LAYER
    NAME "States"
    # GROUP "Boundaries"
    TYPE POLYGON
    STATUS ON
    DATA "mapdata/catalog/political/wa_states"
    HEADER "templates/table_header_template.html"
    FOOTER "templates/table_footer_template.html"
    TEMPLATE "templates/wa_states_template.html"
    CLASS
        COLOR 240 240 230
    END
    DUMP true
    METADATA
        "wms_title" "States"
        "wms_srs" "epsg:2285 epsg:4326"
        "wms_feature_info_mime_type" "text/html"
    END
END

LAYER
    NAME "Topography"
    TYPE RASTER
```

```
STATUS ON
MINSCALE 500000
DATA "mapdata/catalog/topography/wa_shade_1km.tif"
"
TRANSPARENCY 15
DUMP true
METADATA
  "wms_title" "Topography"
  "wms_srs" "epsg:2285"
  "wms_feature_info_mime_type" "text/html"
END
END

LAYER
  NAME "People_in_Institutions"
  # GROUP "Census 2000 (Counties)"
  TYPE POLYGON
  DATA "mapdata/projects/alliance/layers/counties"
  DUMP true
  METADATA
    "wms_title" "People_in_Institutions"
    "wms_srs" "epsg:2285 epsg:4326"
    "wms_feature_info_mime_type" "text/html"
  END

  HEADER "templates/table_header_template.html"
  FOOTER "templates/table_footer_template.html"
  TEMPLATE "templates/
    county_people_in_institutions_template.html"
  TOLERANCE 0
  STATUS OFF
  # MINSCALE 1000000
  MAXSCALE 3500000
  CLASS
    NAME "Up to 400"
    EXPRESSION ([INSTITUTN] >= 31 AND [
      INSTITUTN] <= 400)
    STYLE
      COLOR 112 153 89
    END
  END
  CLASS
    NAME "401 – 1,500"
```

```

        EXPRESSION ([INSTITUTN] > 400 AND [
            INSTITUTN] <= 1500)
        STYLE
            COLOR 178 194 141
        END
    END
    CLASS
        NAME "1,501 – 4,000"
        EXPRESSION ([INSTITUTN] > 1500 AND [
            INSTITUTN] <= 4000)
        STYLE
            COLOR 255 240 204
        END
    END
    CLASS
        NAME "4,001 – 8,500"
        EXPRESSION ([INSTITUTN] > 4000 AND [
            INSTITUTN] <= 8500)
        STYLE
            COLOR 255 153 153
        END
    END
    CLASS
        NAME "More than 8.500"
        EXPRESSION ([INSTITUTN] > 8500)
        STYLE
            COLOR 230 0 0
        END
    END
END

LAYER
    NAME "Census Tracts Query"
    CONNECTIONTYPE postgis
    CONNECTION "user=test password=test dbname=test host=
        localhost"
    PROCESSING "CLOSE_CONNECTION=DEFER"
    DATA "geom from (select geom, id, fipsstco, trt2000,
        stfid, tractid, area, acres, sq_miles, pop_100,
        name, gid from wa_tracts %replacementvalue%) as
        temp using unique gid using SRID=2285"
    TYPE POLYGON
    # TRANSPARENCY 50
    STATUS ON

```



```
HEADER "templates/table_header_template.html"
FOOTER "templates/table_footer_template.html"
TEMPLATE "templates/county_template.html"
  DUMP true
  METADATA
    "wms_title" "Census_Tracts_Query"
    "wms_srs" "epsg:2285"
    "gml_featureid" "id"
    "gml_include_items" "all" #optional (gives all
      available parameters back
  END
  CLASS
    NAME "tracts"
    STYLE
      COLOR 0 38 115
    END
    STYLE
      SYMBOL "POINT"
      SIZE 0
      OUTLINECOLOR 110 110 110
    END
  END
END
END
END # Map File
```

A map file layer with a raster tile index

```
# Start of layer definitions
#####
LAYER
  NAME "Orthos2008"
  TILEINDEX "/data/ortho_image_index" # shapefile
    of the raster tile extents\
  # can be produced using gdaltindex (builds a
    shapefile with a record for each input
    raster file , an attribute containing the
    file name, and a polygon geometry outlining
    the raster.)
  TILEITEM "location" # attribute containing the
    file name (in dbf file)
  TYPE RASTER
  STATUS OFF
  PROJECTION
```

```
        "init=esri:4326"
    END
    DUMP true
    METADATA
    "wms_title" "Orthos2008"
    "wms_srs" "epsg:4326 epsg:3857"
    "wms_feature_info_mime_type" "text/html"
    "ows_enable_request" "*" # for MapServer version
    >= 6 WMS output needs to be enabled explicitly
    END
END
```

5.6 OpenLayers Viewer: Examples OL 2

Commercial background layers and MapServer WMS

Note that this example using Google maps v3.2 (work with OpenLayers 2.10 and higher). Compare the example on the DVD: `data/openlayers/ol_map_simple.html`.

```
<html>
<head>
  <title>MapServer WMS Layer on Top of Commercial maps</title>
  <script src="../../openlayers/OpenLayers.js"></script>
  <script src="http://maps.google.com/maps/api/js?v=3.2&sensor=false"></script>
  <script type="text/javascript">
    var map;
    var options = {
      projection: new OpenLayers.Projection("EPSG:3857"),
      units: "m",
      maxResolution: 156543.0339,
      maxExtent: new OpenLayers.Bounds(-13776237, 5870705,
        -13270618,6177605)
    };

    window.onload = function(){
      map = new OpenLayers.Map( 'map', options );
      var Counties = new OpenLayers.Layer.WMS( "counties",
        "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/
        wacounties_wms.map", {layers: 'Counties', 'transparent':
          true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility':
          false, singleTile:true} );
      var Rail = new OpenLayers.Layer.WMS( "rail",
        "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/
        wacounties_wms.map", {layers: 'Rail', 'transparent': true
          }, {isBaseLayer: false, 'opacity': 0.7, 'visibility': true
          , singleTile:true} );
      var Cities = new OpenLayers.Layer.WMS( "cities",
        "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/
        wacounties_wms.map", {layers: 'Cities', 'transparent':
          true}, {isBaseLayer: false, 'opacity': 0.7, 'visibility':
          false, singleTile:true} );
      var Uninhabited = new OpenLayers.Layer.WMS( "uninhabited",
        "/cgi-bin/mapserv.exe?map=C:/class/data/mapfiles/
        wacounties_wms.map", {layers: 'Uninhabited', '

```

```
transparent': true}, {isBaseLayer: false, 'opacity':
0.7, 'visibility': false, singleTile:true} );

var base_empty = new OpenLayers.Layer("No Background", {isBaseLayer:
true, numZoomLevels: 20, 'displayInLayerSwitcher': true, basename:
"empty"});
map.addLayer(base_empty);
var g_street = new OpenLayers.Layer.Google("Google Streets" ,{
maxZoomLevel:20} ); map.addLayer(g_street);
var g_satellite = new OpenLayers.Layer.Google("Google Satellite" ,{
type: google.maps.MapTypeId.SATELLITE, 'maxZoomLevel':20} );
var g_hybrid = new OpenLayers.Layer.Google("Google Hybrid" ,{
type: google.maps.MapTypeId.HYBRID, 'maxZoomLevel':20} );
var g_physical = new OpenLayers.Layer.Google("Google Physical" ,{
type: google.maps.MapTypeId.TERRAIN, 'maxZoomLevel':20} );
map.addLayers([ g_street, g_hybrid, g_satellite, g_physical, base_empty,
Uninhabited, Cities, Rail, Counties]);

var bounds = new OpenLayers.Bounds(-13776237, 5870705, -13270618,
6177605);
map.zoomToExtent(bounds);

map.addControl( new OpenLayers.Control.LayerSwitcher() );
mp=new OpenLayers.Control.MousePosition();

mp.displayProjection = new OpenLayers.Projection( "EPSG:3857" );
map.addControl(mp);
}; // end bracket of window.onload function
</script>
</head>
<body>
<div id="title"> <h1 class="style2">Simple OL Map – OS Web GIS Class</h1
></div>
<div id="map"></div>
</body>
</html>
```

Commercial layers in OpenLayers 2

Below are some HTML/Javascript code snippets to illustrate how to add commercial layers to an OpenLayers map. For a full example file look at **/data/openlayers/alllayers.html** on the DVD.

<!— MapQuest Layer —>

```
OpenLayers.Layer.MapQuestOSM = OpenLayers.Class(OpenLayers.Layer.XYZ, {
    name: "MapQuestOSM",
    //attribution: "Data CG-By-SA by <a href='http://openstreetmap.org/'>
        OpenStreetMap</a>",
    sphericalMercator: true,
    url: ' http://otile1.mqcdn.com/tiles/1.0.0/osm/{z}/{x}/{y}.png',
    clone: function(obj) {
        if (obj === null) {
            obj = new OpenLayers.Layer.OSM(
                this.name, this.url, this.getOptions());
        }
        obj = OpenLayers.Layer.XYZ.prototype.clone.apply(this, [obj]);
        return obj;
    },
    CLASS_NAME: "OpenLayers.Layer.MapQuestOSM"
});
```

<!-- Bing Layers -->

```
// API key for http://openlayers.org. Please get your own at
// http://bingmapsportal.com/ and use that instead.
var BingApiKey = "
    AqTGBsziZHIjYYxgivLBf0hVdrAk9mWO5cQcb8Yux8sW5M8c8opEC2lZqKR1ZZXf";
var b_street = new OpenLayers.Layer.Bing({key: BingApiKey, type: "Road",name
    : "Bing Roads map"});
b_street.buffer = 0; map.addLayer(b_street);
var b_satellite = new OpenLayers.Layer.Bing({key: BingApiKey,type: "Aerial",
    name: "Bing Aerial"});
b_satellite.buffer = 0; map.addLayer(b_satellite);
var b_hybrid = new OpenLayers.Layer.Bing({key: BingApiKey,type: "
    AerialWithLabels",name: "Bing Hybrid"});
b_hybrid.buffer = 0; map.addLayer(b_hybrid);
```

<!-- Open Street Map Layer -->

```
var mapnik = new OpenLayers.Layer.OSM("Open Street Map");
mapnik.buffer = 0; map.addLayer(mapnik);
```

<!-- Google V3 API Layers -->

<html>

<head>

<!-- include the Google V3.2 API , no key needed but still need to
include the API-->

```
<script src="http://maps.google.com/maps/api/js?sensor=false&v=3.2"></script>
<script type="text/javascript" src="../../lib/OpenLayers-2.13.1/OpenLayers.js"></script>
<script type="text/javascript" >

var g_streets = new OpenLayers.Layer.Google(
  "Google Roads", // the default
  {numZoomLevels: 20}
  // default type, no change needed here);

var g_satellite = new OpenLayers.Layer.Google(
  "Google Satellite",
  {type: google.maps.MapTypeId.SATELLITE, numZoomLevels: 22}
  // in OL 2.9 and Google v2 this used to be {type: G_SATELLITE_MAP,
  //   numZoomLevels: 22});
g_satellite.buffer = 0; map.addLayer(g_satellite); // add to map no tile
  buffer

var g_hybrid = new OpenLayers.Layer.Google(
  "Google Hybrid",
  {type: google.maps.MapTypeId.HYBRID, numZoomLevels: 20}
  // in OL 2.9 and Google v2 this used to be {type: G_HYBRID_MAP,
  //   numZoomLevels: 20});
g_hybrid.buffer = 0; map.addLayer(g_hybrid); // add to map no tile buffer

var g_terrain = new OpenLayers.Layer.Google(
  "Google Terrain",
  {type: google.maps.MapTypeId.TERRAIN}
  // in OL 2.9 and Google v2 this used to be {type: G_PHYSICAL_MAP});
</script>
</head>
</html>
```

OpenLayers examples included on the class DVD

A long list of general OL examples can be found [here for OL 2.X](#) and [here for OL 3](#). Another interesting web page with OpenLayers 2 examples can be found [here](#). There are five examples included on the DVD in the `/data/openlayers/identify` directory:

- **map.html** Identify office location point features on the map (often multiple in one location); display attributes using a MapServer query template (compare code listing in the next section). If you like to implement this example locally you can import the *location* point data from the DVD *data/layers/officelocations.shp* into PostGIS using *shp2pgsql*. View live example [here](#).
- **map_controls.html** Example of multiple controls in OL: edit tool bar, scale bar, and navigation tool bar (pan and zoom). The navigation tool bar is placed in a `<div>` tag to change it's position (located outside the right upper corner of the map). View live example [here](#).
- **map_dynamic.html** Demo of a dynamic MapServer layer. Click on the map to highlight a County. This is a server side implementation and uses the dynamic (PostGIS based) MapServer Layer *querylayer* in *query.map*. Clicking on a County on the map triggers an Ajax call and runs a query to PostGIS (via the *getCounty.php* script) to retrieve the County name. This value is then used for the replacement variable updating the dynamic layer in OL. Sounds tricky doesn't it? But this set-up enables any functionality to be run on the retrieved feature on the server side in PostGIS, including analysis like buffers and intersections or any custom function you may want to write in [PL/pgSQL](#). View live example [here](#).
- **map_staticbuffer.html** Similar to *map_dynamic.html*, but also buffers the County feature geometry before highlighting on the map. View live example [here](#).
- **alllayers.html** Example file with many base layers at once: Google (v3.2), Bing, Open Street Map, and MapQuest. Compare the code snippets for the individual map layer on page 109. View live example [here](#).

Map files and OL style sheet used with the above examples:

- **style.css** OL standard style sheet
- **query.map** map file with two dynamic layers called *querylayer* and *bufferlayer* (including a MapServer replacement variable)
- **identify.map** map file with a point layer of office *locations* with specified MapServer query templates, e.g. *locations_template.html*
- **getCounty.php** returns the County name from the *counties* layer in PostGIS

Below is a list of the MapServer query templates used in **map.html** (for retrieving the identified feature attributes and styling of the output). When running a query, all of the template below are assembled (in the listed sequence) at runtime by MapServer. The result is a HTML table including the results.

locations_template.html
general_header_template.html
table_header_template.html
table_footer_template.html
general_footer_template.html

Code Listing for map.html: Example for identifying attributes in OL (getfeatureinfo request)

This example is included on the DVD **/data/OpenLayers/identify/map.html** along with the map file **identify.map**.

```
<html>
<head>
  <title>Template Example</title>
  <style type="text/css">
    #mapandlegend {
      width: 940;
      height: 650;
      position: relative;
    }
    #map {
      width: 700;
      height: 600;
      border: 1px solid black;
    }
    #thelegend {
      width: 220;
      height: 100;
      border: 1px solid black;
      padding: 12px;
      position: absolute;
      top: 0px;
      right: -10px;
    }
    #results {
      width: 220;
      height: 160;
      right: 0;
      position: absolute;
      top: 140px;
      font-size: 10 pt;
    }
  </style>
</head>
```



```
font: 10 pt "Lucida Grande", Verdana, Lucida, Helvetica, Arial, sans-
    serif;
}
</style>
<script src="../../OpenLayers.js"></script>
<script type="text/javascript">
    var map;
    var options = {
        maxResolution: 898,
        projection: new OpenLayers.Projection("EPSG:2285"),
        units: "m",
        tileSize: new OpenLayers.Size(256, 256),
        maxExtent: new OpenLayers.Bounds(680000,-150000,1600000,740000)
    };

    window.onload = function(){
        map = new OpenLayers.Map( 'map', options );

        var base = new OpenLayers.Layer.WMS("Base Map",
            "http://alliance.terragis.net/cgi-bin/mapserv?map=/var/www
            /mapdata/projects/alliance/alliance2.map", {layers: '
            States,Puget_Sound,Counties,Cities,Roads,Lakes,Rivers,
            Major_City_Names,Main_City_Names,City_Names', '
            transparent': true}, {isBaseLayer:true, 'opacity': 1,
            'visibility': true, singleTile:true} );

        var locations = new OpenLayers.Layer.WMS( "Office Locations",
            "http://localhost/cgi-bin/mapserv.exe?map=C:/class/data/
            mapfiles/identify.map", {layers: 'locations', '
            transparent': true, 'WEBSQL':'roads'}, {isBaseLayer:
            false, 'opacity': 0.7, 'visibility': false, singleTile
            :true} );

        var bounds = new OpenLayers.Bounds(680000,-150000,1600000,740000);
        map.addLayers([base, locations]);
        map.zoomToExtent(bounds);
        map.addControl( new OpenLayers.Control.LayerSwitcher() );
        mp=new OpenLayers.Control.MousePosition();
        mp.displayProjection = new OpenLayers.Projection("EPSG:2285");
        map.addControl(mp);

        map.events.register('click', map, function (e) {
            OpenLayers.Util.getElement('results').innerHTML = "One second...";
            var url = locations.getFullRequestString({
```

```

        REQUEST: "GetFeatureInfo",
        EXCEPTIONS: "application/vnd.ogc.se_xml",
        BBOX: locations.map.getExtent().toBBOX(),
        X: e.xy.x,
        Y: e.xy.y,
        INFO_FORMAT: 'text/html',
        MODE: 'nquery',
        FEATURE_COUNT: 2,
        QUERY_LAYERS: locations.params.LAYERS,
        WIDTH: locations.map.size.w,
        HEIGHT: locations.map.size.h});
    OpenLayers.loadURL(url, '', this, setHTML);
    OpenLayers.Event.stop(e);
  })

}

function setHTML(response) {
  OpenLayers.Util.getElement('results').innerHTML = response.responseText;
}
</script>
</head>
<body><div id="title">Template Example – Office Location Map</div>
<div id="mapandlegend">
<div id="map"></div>
<div id="thelegend"><span class="style1">Offices</span>
  <div></div>
  <div>
  </div>
  <div id="results"></div>
  </div>
</body>
</html>

</script>
</head>
<body><div id="title">Template Example – Office Location Map</div>
<div id="mapandlegend">
<div id="map"></div>
  <div id="thelegend"><span class="style1">Offices</span>

```

```
<div></div>
<div>
</div>
<div id="results"></div>
</div>
</body>
</html>
```

Example for using TileCache layers in OpenLayers 2

A description for installing TileCache can be found here: <http://tilecache.org> The following lines of code illustrate how TileCache based layers can be added to the JavaScript section in the OL web page:

```
var wms_basemap_tiles = new OpenLayers.Layer.WMS("Base tiled", "http://localhost/tilecache/tilecache.py", {layers: 'mymap_base', 'transparent': true}, {isBaseLayer:true, 'opacity': 1, 'visibility': true, singleTile: false} );
var wms_images_tiles = new OpenLayers.Layer.WMS("Images tiled", "http://localhost/tilecache/tilecache.py", {layers: 'mymap_images', 'transparent': true}, {isBaseLayer:true, 'opacity': 1, 'visibility': true, singleTile: false} );
var wms_raster_catalog = new OpenLayers.Layer.WMS("Images 2008", "http://localhost/cgi-bin/mapserv?map=/mnt/maps/mymap.map", {layers: 'Orthos2008', 'transparent': true}, {isBaseLayer:true, 'opacity': 1, 'visibility': true, singleTile:false} );
```

The snippet below shows the configuration of one of the above TileCache layers **mymap_base** in tilecache.cfg (the TC configuration file). The value for *maxresolution* can be determined as follows: Take your *maxExtent* and find the width of it in map units. This is literally the value of east minus the west in our case below:

$$\mathbf{-117.115453 - (-117.459654) = 0.344201 \text{ degrees}}$$

Now we need to divide the width by the number of pixels in which it should be rendered. Given that a tile defaults to 256x256, this means that dividing the width by 256 would fit the entire map into 1 tile. Dividing by 1024 (256*4) fits it into 4 tiles. Example:

$$\mathbf{0.344201 / 1024 = 0.000672267578125}$$

The value above is the **maxResolution** for our example. This should be used in the definition on the OpenLayers page. In tilecache.cfg this value needs to be entered along with the **maxExtent** and the **numZoomLevels** (number of zoom levels):

```
[mymap_base] type=WMS
metaTile=true
url=http://localhost/cgi-bin/mapserv?map=/maps/mymap.map
srs=EPSG:4326
layers=Ocean,Parks,Landmarks,Lakes,Rivers,Lagoons,Roads,Railway,Citynames
bbox=-117.459654,32.960546,-117.115453,33.282610
maxresolution = 0.0003361337890625
levels=7
extent_type=loose
```

5.7 PAQs: Participant Asked Questions

How can the output of the MapServer legend be styled?

The LEGEND definition in the map file gives you a fair amount of customization options such as label font, font size, key icon width and height etc. Images can be used for the legend by specifying the KEYIMAGE parameter in the CLASS object.

LEGEND

```
keysize 16 12    # width and height of generated icon
keyspacing 5 2   # spacing between icons
TRANSPARENT ON   # background set transparent
  LABEL          # Label tag
    TYPE TRUETYPE # Font category
    FONT 'verdana' # Font (needs to be installed and
                  path known to mapserver
    SIZE 8        # Font Size
    COLOR 1 1 1   # RGB color value
  END
END
```

Another option is to use MapServer HTML templates instead of the legend graphics (gif format) generated by default. A simple solution is to incorporate a HTTP GetLegendGraphic request as a HTML image source tag into our web page or application. More information at mapserver.org.

Which charting tools are available for use in my web application?

One simple option are the **MapServer Chart layers**. They provide simple Pie and Bar charts and are easily [configured](#). Pie Charts are the default chart type. This can also be set by typing

```
PROCESSING "CHART_TYPE=PIE"
```

In the following example (taken from the MapServer documentation) for each shape in the layer's data source, the STYLE SIZE is used to set the relative size (value) of each pie slice, with the angles of the slices that are automatically computed so as to form a full pie:

LAYER

```
NAME "Ages"
TYPE CHART
CONNECTIONTYPE postgis
CONNECTION "blabla"
DATA "geom from demo"
PROCESSING "CHART_TYPE=pie"
PROCESSING "CHART_SIZE=30"
STATUS ON
CLASS
```

```
NAME "Population Age 0–19"
STYLE
  SIZE [v1006]
  COLOR 255 244 237
END
END
CLASS
NAME "Population Age 20–39"
STYLE
  SIZE [v1007]
  COLOR 255 217 191
END
END
CLASS
NAME "Population Age 40–59"
STYLE
  SIZE [v1008]
  COLOR 255 186 140
END
END
END
```

In the example above, if for a given shape we have $v1006=1000$, $v1007=600$ and $v1008=400$ then the actual pie slices for each class will be respectively 50, 30 and 20 percent of the total pie size.

Here are three more options to explore:

In MS4W **OWTchart** a CGI based solution is included: <http://www.maptools.org/owtchart>. "The OWTChart Engine produces GIF images of virtually any type of chart from a set of input parameters. The program can be used as a CGI in a web server environment, in which case it will return a GIF image with the chart built from the parameters found in the QUERY_STRING part of the URL, or from the body of a POST request."

How can we connect from MapServer to MS SQL Server 2008?

Answer: via a MapServer plug-in. This example is taken from <http://mapserver.org/development/rfc/ms-rfc-38.html?highlight=sql%20servermapserver.org>

```
LAYER
NAME "Roads"
CONNECTIONTYPE PLUGIN
PLUGIN "C:\class\software\ms4w\Apache\specialplugins\
  msplugin_mssql2008.dll"
```

```
CONNECTION "server=mysqlserver2008.com; uid=dbusername;
          pwd=dbpassword; database=Roads Database; Integrated
          Security=false"
DATA "geom from roads"
TYPE LINE
STATUS ON
PROJECTION
  "init=epsg:4326"
END
CLASS
  STYLE
    COLOR 0 0 255
    WIDTH 8
  END
END
END
```

How can we join data from a DBF file to a shapefile in MapServer?

Below is an example of a shapefile based map layer including a one-to-many join of *legal service provider offices (DBF file)* to a *region polygon shapefile* (one region with multiple legal aid providers that can have multiple offices each). The following query templates are used (included on the DVD in **data/templates/join**):

```
table_header_template.html
table_footer_template.html
region_sum_template.html
region_providers_officelist_template.html
```

LAYER

```
NAME "Providers_Regions"
TYPE POLYGON
DATA "c:/class/data/layers/projects/alliance/
      layers/region_sum"
HEADER "templates/table_header_template.html"
FOOTER "templates/table_footer_template.html"
TEMPLATE "templates/region_sum_template.html"
JOIN
  NAME officelist
  TABLE "c:/class/data/layers/projects/alliance/layers
          /region_providers_officelist.dbf"
  FROM REG_ID
```

```

TO REG_ID
TYPE one-to-many
TEMPLATE "templates/
        region_providers_officelist_template.html"
END
TOLERANCE 0
STATUS OFF
CLASS
    NAME "Providers by Region (2007)"
    STYLE
    END
    STYLE
        SYMBOL "POINT"
        SIZE 4
        OUTLINECOLOR 000 000 031
    END
END
METADATA
"wms_title" "Providers_Regions"
"wms_srs" "epsg:2285 epsg:4326"
"wms_feature_info_mime_type" "text/html"
END
END

```

Below are listings of the two template files involved.

```

<!-- MapServer Template -->
<!--region_providers_officelist_template.html-->
<tr><td style="width:_50px">[officelist_FTE_MAP] FTE</td>
<td>[officelist_NAME] [officelist_OFFICENAM], [
officelist_STREET] [officelist_SUITE], [
officelist_CITY], [officelist_STATE] [officelist_ZIP
]</td></tr>

```

```

<!-- MapServer Template -->
<!--region_sum_template.html-->
<tr><th colspan="3">Provider FTE</th><th>Region [REG_ID]
"[REGION]"</th></tr>
<tr><td colspan="3">Number of Attorneys in Region "[
REGION]"</td><td align="center">[SUM_ATTORN] </td></tr>
<tr><td colspan="3">FTE sum</td><td align="center">[
PROVIDER_S] </td></tr>
<tr><td colspan="3">Northwest Justice Project</td><td
align="center">[NWJP] </td></tr>

```



```
<tr><td colspan="3">Columbia Legal Services</td><td align="center">[CLS] </td></tr>
<tr><td colspan="3">Pro Bono FTE equivalents</td><td align="center">[PROBONO] </td></tr>
<tr><td colspan="3">Mediation Services</td><td align="center"> not available</td></tr>
<tr><td colspan="3">CLEAR FTE equivalents</td><td align="center">[CLEAR_FTE] </td></tr>
<tr><td colspan="3">Specialty Legal Aid Providers </td><td align="center">[SPECIALTY]</td></tr>
[join_officelist]
```

How can we create a dynamic layer with MapServer and PostGIS?

Below is an example of a (MapServer map file) layer definition using a MapServer replacement variable (embedded in "%" strings). MapServer will replace the variable with its value when the layer is requested at run time. For example you can use a variable you retrieved via a query against a PostGIS database via a PHP POST request to your server. Note that the variable value can be any string that PostGIS can handle in a SQL query, including all PostGIS functions. Working with OpenLayers the *mergeNewParams* method can be called on the map layer (see below) to update a layer dynamically. To avoid any MapServer errors initially set the replacement variable to a SQL string that evaluates to retrieve zero features (nothing displayed for the query layer on the map initially). Make sure to use uppercase characters for the variable name in the map file and the OL page (this is necessary in order to have the variable names match exactly because OL converts the params to all uppercase).

In a map file you can define a dynamic, PostGIS based layer similar to this:

LAYER

```
NAME "querylayer"
GROUP "querylayer"
TYPE POLYGON
STATUS ON
```

METADATA

```
"wms_title" "querylayer"
"wms_srs" "epsg:2285 epsg:4326"
"wms_feature_info_mime_type" "text/html"
```

END

```
DUMP true
```

```
CONNECTIONTYPE postgis
```

```
CONNECTION "user=postgres password=postgres dbname=
osgis host=localhost"
```

```
DATA "geom from (%WEBSQL%) as subquery using unique gid
using SRID=2285"
```

```
# %WEBSQL% will be replaced at run time by MapServer.
with the string containing the subquery syntax to
PostGIS
```

```
# e.g. "select * from counties where name ilike 'King'"
```

```
# this translates at runtime to the full statement:
```

```
# DATA "geom from (select * from counties where name
ilike 'King') as subquery using unique gid using
SRID=2285"
```

VALIDATION

```
'WEBSQL' '.' # needed for security check (this is not
secure because it allows any input variable
value – but one can use regular expressions here
to validate
```

```
END
    DUMP true
    CLASS
        NAME "Query Result"
    STYLE
        OUTLINECOLOR 255 0 0
        WIDTH 3
    END
END
END
```

In the OL page include the dynamic layer, and an *update function* (and call the update function when appropriate):

```
querylayer = new OpenLayers.Layer.WMS( "Locations", "http://orca.terragis.net/
    cgi-bin/mapserv?map=/var/www/mapdata/projects/examples/identify.map&", {
    layers: 'querylayer', 'transparent': true, 'WEBSQL': 'counties where 1 =
    0'}, {isBaseLayer: false, 'opacity': 0.7, 'visibility': true, singleTile:
    true} );
```

```
// merge parameter from the PHP post: call update() function for this
update_query();
```

```
function update_query() {
    var querystring = "<?PHP echo $mapquery; ?>";
    querylayer.mergeNewParams( { 'WEBSQL': querystring } );
}
```

Using this set-up you can even write your own pgSQL functions for use with the MapServer replacement variable. There are virtually no limits to use this ...

Can I use ESRI File Geodatabases as input for MapServer?

Read access to ESRI file geodatabases is supported via OGR (the software version needs to be GDAL version ≥ 1.9). Note two main requirements: Only file geodatabases created by ArcGIS version 10.0 can be used read, and your GDAL/OGR software build needs to contains the file geodatabase *FileGDB* driver. For more information see <http://mapserver.org/input/vector/filegdb.html> Below is an example for a map file with a file geodatabase layer.

LAYER

```
NAME "fgdb_poly"
TYPE POLYGON
STATUS OFF
CONNECTIONTYPE OGR
CONNECTION "../data/filegdb/us_states.gdb"
DATA "statesp020"
LABELITEM "US STATE"
CLASS
  NAME "US States"
  STYLE
    COLOR 255 120 120
    OUTLINECOLOR 0 0 0
  END
  LABEL
    COLOR 255 255 255
    OUTLINECOLOR 0 0 0
  END
END
END
END
```

Index

- Anti Grain Geometry, 50
- Apache, 3
 - start service, 35
- CGI, 8
- Charts, 119
- Class DVD
 - examples, 112
- Client Side Frameworks, 12, 13
 - Leaflet, 13
 - OpenLayers, 12
- Commercial Layers, 28, 109
- D3, 13
- data
 - formats, 88
- Data-Driven Documents, 13
- database
 - spatial, 17
- debugging, 33, 38, 39
- Django, 17
- error logging, 33, 38, 39
- ESRI File Geodatabases, 125
- FeatureServer, 18, 23
- fonts, 51
- foss
 - licenses, 76
- FOSS4G, 4
 - Annual Conference, 84
 - articles, 79
 - books, 78
 - mailing lists, 84
 - User Groups, 84
 - web mapping, 81
 - web sites, 79
- GDAL, 67
 - formats, 88
- GeoEXT, 12
- GeoJSON, 14
- GeoMoose, 23
- GeoServer, 23
- GIS
 - data formats, 88
 - desktop, 83
 - Live DVD, 83
 - Samplers, 83
- GNU
 - project, 3
- Google
 - map projection, 30
 - map tiles, 29
 - maps v2, 28, 109
 - maps v3, 28, 109
- GRASS, 70
- gvSIG, 75
- IRC, 84
- Java Script, 13, 26
- labels, 51
- Leaflet, 13, 23
- licenses, 4, 76, 77
 - BSD, 13, 77
 - GNU, 76
 - LPGL, 76
 - MIT, 76
 - Mozilla, 77
- Linux
 - GIS install, 86
 - Server, 24
 - Ubuntu, 86

- map file, 38
 - highlighting, 48
 - tools, 49
- map tile engines, 82
- Map Viewer
 - Leaflet, 13
 - OpenLayers, 12
- Mapbender, 14
- MapCache, 7, 21
- MapFish, 15, 23
- Mapnik, 66
- MapProxy, 21
- MapScript, 7
- MapServer, 17, 23, 33
 - AGG, 50
 - book, 78
 - CGI, 33, 37
 - charts, 119
 - configuration, 38
 - data formats, 88
 - debugging, 33, 38, 39
 - dynamic layer, 123
 - error logging, 33, 37–39
 - ESRI File Geodatabases, 125
 - exercises, 40
 - fonts, 49, 51
 - for Windows, 35
 - HTML templates, 118
 - join dbf file, 120
 - labels, 51
 - legend style, 118
 - map file, 38
 - map file tools, 48
 - MapCache, 7
 - MapScript, 7
 - migration guide, 49
 - MS4W installation, 35
 - notes, 49
 - performance, 46
 - publish Web Map Services, 42
 - rendering, 49, 50
 - sample map files, 97
 - shp2img, 39
 - Suite, 7, 8, 21
 - TinyOWS, 7
 - tools, 47
 - utility programs, 8, 47
 - version 6, 49
 - version migration, 49
 - WMS, 42
- MS4W, 35
 - MS Seven, 35
 - MS Vista, 35
- Notepad++, 48
- OGC
 - standards, 1
 - WMS standard, 43
- OGR
 - formats, 94
- Open Street Map, 19
 - projection, 30
- OpenGeoSuite, 21
- OpenJUMP, 73
- OpenLayers, 6, 12, 23, 26
 - API, 26
 - Bing, 28
 - Bing layer, 109
 - book, 78
 - Commercial Layers, 28, 109
 - examples, 32, 108, 112
 - exercises, 31
 - Google layer, 109
 - Google maps, 28, 109
 - MapQuest, 28
 - MapQuest layer, 109
 - open street map, 28, 109
 - optimizing, 32
 - OSM layer, 109
 - tools, 81
 - WMS, 29
- PostGIS, 17, 23, 56
 - administration, 56
 - and MapServer, 63
 - book, 78, 79
 - data encoding, 58
 - data export, 58

- data import, 58
- dynamic layer, 123
- exercises, 62
- installation, 52
- interacting with, 54
- notes, 64
- pgAdmin, 56
- using, 52
- programming language
 - JavaScript, 13
 - Python, 17
- Proj4, 67
 - utilities, 67
- Python, 17
- QGIS, 72
- Quantum GIS, 72
- Server
 - configuration Linux, 24
 - GIS install, 86
 - hosting options, 22
- shp2pgsql
 - data encoding, 58
- software
 - licenses, 76
- spatial database, 17
- Spatialite, 66
- TileCache, 21, 23
- TileMill, 21
- TinyOWS, 7
- TopoJSON, 14
- Transactional Web Feature Service, 7, 8
- Ubuntu
 - GIS, 24
 - GIS install, 86
 - GIS repository, 86
 - Server, 24
- uDig, 72
- utility programs, 8, 47
- Web Feature Service, 7
- WEB GIS
 - frameworks, 6
 - web mapping
 - frameworks, 17, 81
 - software, 81
 - WFS, 23
 - WFS-T, 7, 8
 - WMS, 23
 - exercises, 45