Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

# Section VII
# Cookbook on Webmapping and GeoWebServices

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# Introduction to GeoWebServices
# Lecture note on the history and principles of geowebservices

*Version 2.0 - September 18, 2012*



*Key points*

This document is intended to serve as a quick introduction to the notion of geo-webservices. It was originally written for students at the International Institute for Geo-Information Science and Earth Observation (ITC) that are fairly familiar with spatial data and GIS. It is based on the teaching materials (slides, exercises) for various modules on WebGIS, Webmapping, Spatial Data Infrastructures and the like, developed by Rob Lemmens, Andreas Wytzisk and Barend Köbben.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## B.1. From monolithic to distributed GIS architectures

Interoperable geowebservices are the latest in a long line of software for handling geographic, or spatial, information. In 1962 Roger Tomlinson built the Canada Geographic Information System to determine the land capability for rural Canada by mapping information about soils, agriculture, land use, etc. He coined the term GIS for software that is used to gather, visualize and analyse geo-information. Tomlinsons GIS software was running on a large mainframe computer and its architecture was what we call monolithic, with the presentation logic, application logic, and data management layers combined in one software tier. This might still be the typical design for simple desktop applications, but nowadays larger GISs, like other software systems, have their logical layers separated. Separate logical layers, a depicted in Figure B.1 improve the interoperability between Geographic Information Systems.

One or more database layers take care of the data storage and retrieval, application layers are used to analyse the information, a mapping engine turns the information into maps, and separate client software gives the actual users access to all of this. The main advantage of such a setup is the flexibility: the different parts can be distributed over various computers and thus can easily be scaled, that is, adopted to changing conditions, such as an increasing number of users. Because the client software is separated from the application logic, the same backend can serve different client platforms (e.g. PCs, PDAs and mobile phones). Equally the data being used in the analysis part can come from various different information sources, from various providers, even from different places on the globe.

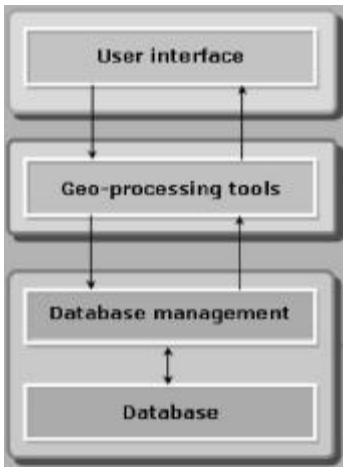


*Fig. B.1: A typical setup of layers in a GIS.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## B.2. Interoperable webservices

Probably the bestknown example of such a distributed system approach is the World Wide Web. The WWW consist of many distributed servers that are responding to individual requests of a sheer endless amount of distributed clients. This kind of system is only possible when the different components are guaranteed to work together, because they are interoperable.

The International Standards Organisation (ISO) defines interoperability as: "The capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units." Interoperability in computer systems, as shown in Figure B.2, means in the first place that the systems are able to transfer data seamlessly.

This can be achieved by having the data encoded in a standardized, platform and application independent manner. Nowadays,the popular encoding scheme used for that purpose is the eXtensible Markup Language (XML).

### *B.2.1. XML: eXtensible Markup Language*

The eXtensible Markup Language (XML) is a computer language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format and widely used for the representation of arbitrary data structures, for example in web services [wikipedia.org].

The structure of XML uses a strict separation of content from presentation. XML is "self descriptive", it provides a way to encode both structure and con-

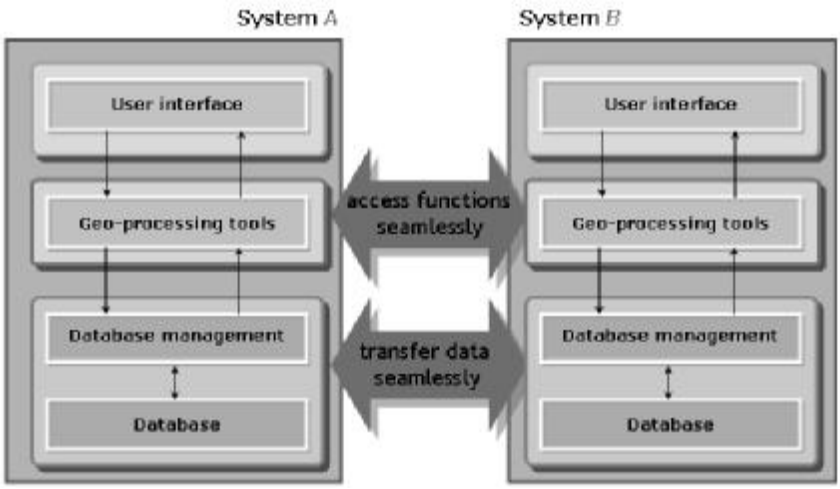


*Fig. B.2: Interoperable Geographic Information Systems.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



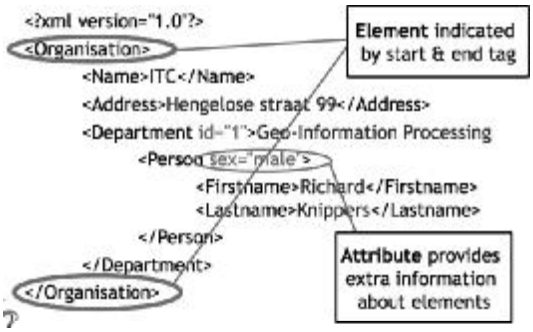


*Fig. B.3: An example of XML.*

tent of data. In any XML-language, the XML standard just sets the syntax, and an XML schema is needed to explain what the XML means. One can compare it to a set of languages that share exactly the same grammar. The big advantage is that only translation of individual words is needed to translate one language to another...

### *B.2.2. Webservices*

Interoperability also means that two information systems should be able to access distributed functionality in a seamless manner. This would for example mean that one Geographic Information System could use the geo-processing tools of another GIS. For that to work regardless of operating system, computer platform or software used, we have to specify and set up an infrastructure of interoperable software services. Service-oriented software differs from traditional software in that it fully encapsulates its own functionalities and makes it accessible only via well specified and standardised interfaces. These interfaces publicise the methods that a software component implements, and its calling conventions. In other words, you do not know, nor have to know, how the service actually works, only what input it can receive and what output you can expect back. There are many ways of setting up such Service Oriented Architectures (or SOA's), but by far the most used distribution platform is the WWW and the services implemented on that are logically called webservices.

Webservices are components that can be described, published, located and invoked over the Internet. They are defined as loosely coupled components that communicate via XML-based interfaces.

Loosely coupled means that they are independent of computer platform and can be exchanged by similar components, e.g., when one service fails or is too busy, the system can find a similar service elsewhere on the Web. The XML-based interfaces are human readable and selfdescribing, which allows for automated discovery of their functionality.

A simple example of such a Webservice is a currency convertor, for example the one at http://www.webservicex.net/. If an application (such as the Demo

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



form on the webpage) is used to send a request to this service formatted as a standardized XML message

```
<FromCurrency>USD</FromCurrency>
<ToCurrency>EUR</ToCurrency>
```

the service returns a standardized XML response:

```
<double>0.92635</double>
```

If webservices have spatial functionality, for example if they use geographic data, can output maps or find routes, we call them geowebservices.

## B.3. Geowebservices

There are many such geowebservices available, the best-know proponent of it probably is Google Maps, and other examples are Bing Maps and Yahoo Local Maps. These examples can be used by anybody, as their interfaces are publicly available, but they are still proprietary in the sense that they are defined, developed and owned by commercial companies. Alternatively, Open Standards are created in an open, participatory process, where everyone interested can influence the standard.

The resulting specifications are non-proprietary, that is, owned in common. That means free rights of distribution (no royalty or other fee) and a free, public, and open access to interface specifications that are also technology neutral. There is a set of such well defined Open Standards for geowebservices: the Open Web Services (OWS) of the Open Geospatial Consortium (OGC).

The OGC was founded in 1994 as a not-for-profit, international voluntary consensus standards organization that develops Open Standards for geospatial and location based services. Their core mission is to deliver interface specifications that are openly available for global use. The Open Web Services specifications are the basis of many high-profile projects (e.g. the European Community INSPIRE initiative).

### *B.3.1. Open Web Services specifications (OWS)*

There are OWS specifications for most parts of the spatial data storage, analysis and delivery process:

- for geographic data encoding: the complete and complicated Geographic Markup Language (GML), and the simpler, more limited Keyhole Markup Language (KML);
- for spatial data delivery: the Web Coverage Service (WCS) and Web Feature Service (WFS), for querying and retrieving raster and vector data respectively;
- for processing of spatial data: the Web Processing Service (WPS). Note that this specification does not standardize the analysis or processing methods themselves, but rather defines a standardized interface that lets you publish

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



geospatial processes, and lets client software find those processes and employ them.
- for data visualisation in the form of maps there is the Web Map Service (WMS). This is by far the most mature and widest adopted OWS specification. There are numerous open source as well as commercial solutions offering WMS functionality. Related to WMS are the Styled Layer Descriptor (SLD) specification, for map styling, and the Web Map Context Documents (WMCD) specification, for map setup and layout;
- for describing and finding spatial data, there is a set of metadata specifications in the Catalog Service Web (CSW).

We will use WMS as an example to show the working of OGC specs in a bit more detail, see the OGC website for more details on the other specifications.

The WMS specification defines four interfaces: GetCapabilities, GetMap, GetLegendGraphic and GetFeatureInfo.

The GetCapabilities interface is used by client software to ask for the capabilities of the service: what layers are available, what projection systems can the maps be delivered in, what output formats can be requested, etc.

So if you point your browser to the URL:

```
http://geoserver.itc.nl/cgi-bin/mapserv.exe?
map=d:/Inetpub/mapserver/
config.map&SERVICE= WMS&VERSION=1.1.1&REQUEST=GetCapabilities
```

the service will return an XML file describing the capabilities. It is human-readable text, but not really meant for reading: Software (GIS or mapping clients) will parse it to be able to show layers, zoom to the extent, know what projection to use, etc...

Based on the GetCapabilities result, the GetMap request is issued to ask for an actual map. Because the client knows the possibilities of the service from the GetCapabilities response, it can issue its request with specific parameters asking for example for a one or more layers of information (LAYERS=borders, forest), in a certain part of the world (BBOX=x1,y1,x2,y2). It will also request that the output is returned in a format that the client can handle (e.g., a web browser will request FORMAT=image/png), and in a specific size in pixels (e.g., WIDTH=250&HEIGHT=300). Furthermore, the map output will have to be be requested in a specific projection (SRS=EPSG:4362).

The SRS, or Spatial Reference System is usually expressed as an EPSG code. EPSG is the European Petroleum Standards Group, and their standardized descriptions of datums and projections are used worldwide. See http://spatialreference.org for an overview.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Thus, if you type in the URL:

```
http://geoserver.itc.nl/cgi-bin/mapserv.exe?
map=D:/Inetpub/mapserver/
config.map&SERVICE=WMS&VERSION=1.1.1&REQUEST=GetMap
&LAYERS=forest,railroad,airports
&STYLES=
&SRS=EPSG:4326
&BBOX=97,5,105,20
&WIDTH=400&HEIGHT=600
&FORMAT=image/png
```

in a web browser, the WMS at that site will return the graphic shown in Figure B.4.

If the user of a mapping client triggers a zoom command, the client will issue a new GetMap request, now with a smaller BBOX. This process will be repeated over and over again while the user interacts with the map.

The GetLegendGraphic request is used in a similar fashion to get a pictorial rendering of a legend item separately.

In WMS services that advertise layers of data as queryable, the GetFeatureInfo request can be used to find attribute values in the underlying data. The client software reports the location of a request in pixel coordinates within the map graphic, and requests theWMS to report on the attribute values of the location in specific layers. The WMS thus has to take into account the size in pixels and in real world coordinates of the graphic it has supplied, plus any reprojections is has done, to calculate the request location in the coordinate space of the origi-

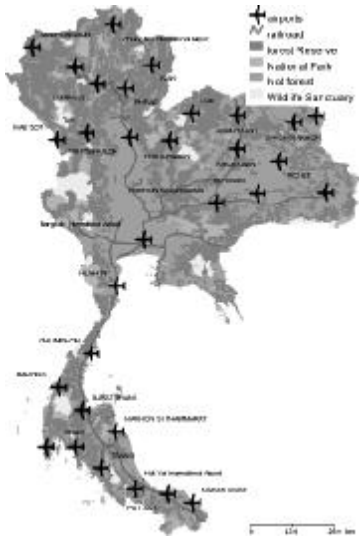

*Fig. B.4: Result of the WMS request in the URL.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



nal data. It then finds data objects at that location, and does a lookup in the original data store (a vector or raster file, a database, etc) to find appropriate data attributes. It then reports these back to the client in the format requested (plain text, HTML or XML).

You can experiment with the WMS and other OWS services at our website.

```
http://geoserver.itc.nl/mapserver/
```

An easy way to see how the different parameters of the interfaces work is the testing form at:

```
http://geoserver.itc.nl/mapserver/testURL.html
```

### *B.3.2. Mapping in a Service Oriented Architecture environment*

The principle of disseminating maps in a webservices environment is depicted in Figure B.5. This general setup is being used in many of today's webmapping efforts, with considerable variation in the choice of technology for the mapping service and the subsequent map formats. This choice to a large extent defines the possibilities of the system as a whole to achieve what we call the automatic and direct production of maps.

By "direct" we mean that the maps are generated on-the-fly from the data. This is necessary because the map generation should fit in interoperable Spatial Data Infrastructures, and it guarantees the maps are always upto-date. To achieve this directness, the visualization functionality should be loosely coupled to the other parts of the system. The Open Geospatial Consortium's (OGC) Open Web Services (OWS) and related specifications are especially useful for this. These web-services are designed to take their input from a variety of distributed sources and generate output meant for Internet dissemination.

We consider "automatic" to mean that the maps will be generated from the data by the system "working by itself with little or no direct human control". It is important to note that this automation in most current systems does not include the cartographic decisions as to what type of map to use for different datatypes and data instances. The link between data- and visualisation-type has to be made by a human (the cartographer in Figure B.5), setting up the appropriate service configuration.

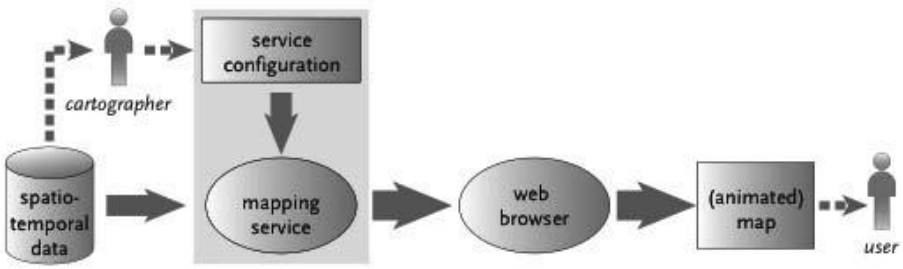


*Fig. B.5: General principle of dissemination of maps in a Service Oriented Architecture.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### *B.3.3. The FOSS4G geo-webservices stack*

We use geowebservices at ITC in teaching, research and projects. Students and staff build services and clients using a software and data stack that is employing so called Free and Open Source Solutions for Geospatial (or FOSS4G for short). This Open Source stack can be set up using a wide choice of software.

We use a set-up that we call SDI-light. The term SDI for Spatial Data Infrastructure may be usually connected with (very) large regional or national spatial data warehouses, but the principles of SDI can also be applied in more simple and cost effective ways. The down-to-earth approach of SDI-light provides researchers and students alike with a proof of concept platform for relatively simple, low cost, yet powerful ways of sharing data amongst various distributed offices and institutions as well as the general public. To achieve this, we use open standards whenever available, opensource solutions where possible and commercial software where necessary (more about SDI-light in Köbben et al (2010)).

In general, the main building blocks we use are:

- *A spatial database backend* that stores the spatial data using the Open Geospatial Consortium Simple Features specifications. As a platform, the PostGIS extension to the object relational DBMS PostgreSQL is a logical choice. First, because PostgreSQL is a solid DBMS that has a reasonably gentle learning curve, yet is wonderfully appropriate for advanced database applications, and its documentation is very transparent. PostGIS in addition, is the leading open standards implementation of spatial vector management, and enjoys a lively and supportive user/developer community.
- A set of *interoperable middleware web applications* that interface with the database backend and with each other, and fulfill tasks such as delivering maps for visualisation purposes or providing data and processing services. We use existing open source solutions, mainly MapServer and GeoServer.

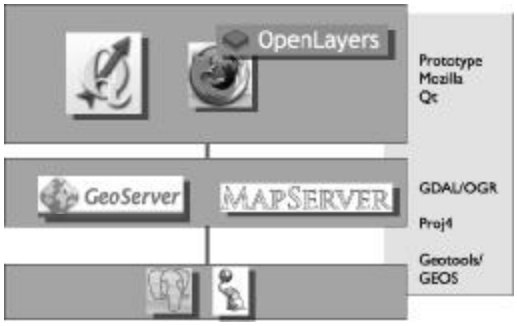


*Fig. B.6: The Open Source stack used in ITC's SDI-light.*

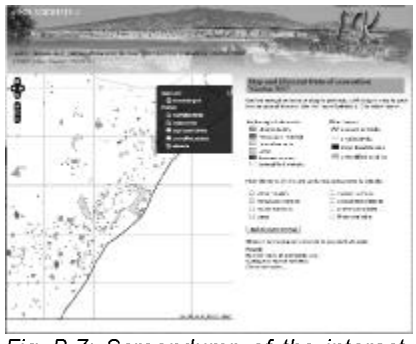

*Fig. B.7: Screendump of the interactive map part of the Melka Kunture Virtual Museum.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



- Simple (thin) *browser-based clients* enabling access to the maps and data. We employ the OpenLayers API and Scalable Vector Graphics (SVG). There are also more established standalone (thick) clients: We use QuantumGIS for map viewing, data analysis and editing.

### *B.3.4. An application example: The Melka Kunture Virtual Museum*

One website that uses this stack is the Melka Kunture Virtual Museum at:

```
http://geoserver.itc.nl/melkakunture/
```

Melka Kunture lies in the upper Awash valley, 50 km south of Addis Abeba (Ethiopia). It is the site of very important archeological finds and 70 archaeological levels have been discovered so far, the oldest dating back to about 1.7 million years. In 2003 the Museum of Melka Kunture was built, and in 2007 a Virtual Museum was put on the WWW. It represents not only the four buildings and the Open Air Museum, but also gives access (in the For Scientists section) to the vast amount of spatial data of the excavations and various finds. All elements have been stored in a spatial database. We use MapServer to provide WMSes for the excavation data and the OpenLayers script offers these as interactive map views in the web pages. Custom Javascript on client-and server-side provides further links from the spatial excavation data to the content of the museum exhibition panels, and the other way around. In this way, users can click on finds in the map, find more data (and sometimes drawings and photos) of that element, as well as links to the appropriate museum sections.

## Further reading

This further reading explains the ITC FOSS4G software stack and its use in education, research and projects.

OGC website. http://www.opengeospatial.org/. This is the offcial home page of the Open Geospatial Consortium, where you can find all relevant standards and information about them.

Köbben Barend; de By, Rolf; Foerster Theodor (2010) Using the SDIlight approach in teaching a Geoinformatics Master. Transactions in GIS, 14 : s1, pp. 25-37.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# Introduction to the OSGeo Live System

*Version 2.0 - September 18, 2012*

*Key points*

In this section the starting of OSGEO LIVE system is explained. For the following hands-on examlpes we will be using the free and open source GIS and webmapping applications from the OSGEO LIVE system…

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



The Open Source Geospatial Foundation, or OSGEO, is a not-for-profit organization whose mission is to support and promote the collaborative development of open geospatial technologies and data. The foundation provides financial, organizational and legal support to the broader open source geospatial community. OSGeo also serves as an outreach and advocacy organization for the open source geospatial community, and provides a common forum and shared infrastructure for improving cross-project collaboration.

The foundation's projects are all freely available and useable under an OSI-certified open source license. More information at the website:

```
http://www.osgeo.org/
```

The OSGEO LiveDVD (the current version is OSGeo-Live 6.0) is a self-contained bootable DVD, USB drive or Virtual Machine based on the Xubuntu operating system, that allows you to try a wide variety of open source geospatial software without installing anything. It is composed entirely of free software, allowing it to be freely distributed, duplicated and passed around.

It provides pre-configured applications for a range of geospatial use cases, including storage, publishing, viewing, analysis and manipulation of data. It also contains sample datasets and documentation.


Many applications are also provided with installers for Apple OSX and Microsoft Windows. More information, including alternative disk images and installations, can be found on the website:

```
http://live.osgeo.org/
```

Task 1 : To use the OSGEO LiveDVD, simply:

1. Insert the DVD or the USB stick in a computer.
2. Reboot the computer. In Microsoft Windows, you might have to verify or set the boot device order, to allow startup from a CD (mostly by pressing a key during start-up, typically the F12 key). On Apple OSX, to boot from CD, you hold the C-key while restarting.
3. For CD booting: Wait for the "boot:" prompt, then press "Enter" to startup and login.
4. For USB booting, you should see a menu where you choose the first option (start normally).

The Xubuntu OS should start up, and now you should have a desktop similar to the figure below (depending on the version of the LiveDVD you are using).

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# A Simple Map Client in a Webpage
# OpenStreetMap in an OpenLayers Website

*Version 3.9 - September 17, 2012*



*Key points*

This is a reference for the use of OpenLayers, a JavaScript Library based on AJAX-principles (Asynchronous JavaScript And XML). OpenLayers can be used to build general WebMapping clients, among others to connect to OGC Web Map Services. In this exercise we will us it to create a web page that gives you access to OpenStreetMap, an Open Data map service. You will learn how to:

- learn the principles of client-side Javascript;
- Use the OpenLayers library in a web page;
- Make a simple map using the OpenStreetMap background.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### D.1. Short introduction to JavaScript

**!** → You are advised to read is page in your web browser, because you can then see how the actual examples work. You can find the web page either in the appropriate BlackBoard module as a link to Dynamic HTML and scripting or directly at: http://kartoweb.itc.nl/courses/JavaScriptExamples/

Javascript allows us to create and change webpages programmatically, through the use of Dynamic HTML. DHTML is built on top of HTML. That is, DHTML includes all the elements that make up a traditional web page. However, with DHTML all of those elements are now programmable objects. You can assign each element an ID and then use scripting to alter them after the page has been downloaded.

Dynamic HTML allows you to dynamically control the look and content of a Web page. For example, the web page has an line of text that dynamically changes colour if you move the mouse over it. To achieve this, we have added the id attribute to the HTML element <H4>, and assigned the id the value "myHeader", thus creating a programmable object:

```
<H4 id="myHeader">THIS WILL CHANGE IF THE MOUSE MOVES OVER IT...</H4>
```

Then we've added calls to two script functions to the object, that will be triggered by events (i.e. actions by the user or the web browser) in this case the onmouseover and the onmouseout events:

```
<H4 id="myHeader" onmouseover="ChangeColor()" onmouseout="ChangeBack()">
THIS WILL CHANGE IF THE MOUSE MOVES OVER IT...</H4>
```

Finally we've added the actual script functions, using the JavaScript language, that will retrieve the programmable object by its name and change the colour attribute of its style:

```
<SCRIPT LANGUAGE="JavaScript">
function ChangeColor()
{ document.getElementById("myHeader").style.color = "blue"; }
function ChangeBack()
{ document.getElementById("myHeader").style.color = "black"; }
</SCRIPT>
```

On the web page you will find links to other examples of what you can achieve with scripting DHTML.

From the name alone, you might think JavaScript is like the programming language Java. However, the JavaScript language resembles Java in some ways, but does not have its static typing and strong type checking. JavaScript does support most Java expression syntax and basic control-flow constructs.

In contrast to Java's compile-time system of classes built by declarations, JavaScript supports a runtime system based on a small number of data types

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



representing numeric, Boolean, and string values. This results in so called loose typing, which means variable data types do not have to be declared specifically: if you assign a string value to a variable, it automatically will become a string type variable! JavaScript is interpreted (not compiled) by the client software (usually the browser). The code is integrated with, and embedded in, HTML.

### *D.1.1. Embedding Javascript in a web page*

You can embed JavaScript in an HTML document in the following ways:

1. By specifying a JavaScript expression as the value of an HTML attribute, using an event handler, as demonstrated in the previous section.
2. As statements and functions within a <SCRIPT> tag.

Task 1: In listing 1 you find a simple script.

Create a new, empty, WWW-page in coding mode (e.g., use NotePad++, TextWrangeler, Medit (or a similar smart editor). Type the JavaScript and HTML code as shown in listing 1 into the HTML-code editor window.

Save the page and open it in a web browser: It should display the following in the browser:

```
This is scripted content! This is plain HTML...
```

Notice that there is no difference in appearance between the first line, generated with JavaScript, and the second line, generated with plain HTML.

The <SCRIPT> tag is an extension to HTML that can enclose any number of (Java)Script statements. A document can have multiple script tags, and each can enclose any number of JavaScript statements.

*Listing 1: A simple scripted page*

```
<HTML>
  <HEAD>
  <SCRIPT LANGUAGE="JavaScript">
    document.write("This is scripted content!");
  </SCRIPT>
  </HEAD>
  <BODY>
  <P>This is plain HTML... </P>
  </BODY>
</HTML>
```

Generally, you should define the JavaScript for a page in the HEAD portion of a document. That way, all functions are defined before any content is displayed. Otherwise, the user might perform an action while the page is still loading that triggers an event handler and calls an undefined function, leading to an error.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Rather than embedding the JavaScript in the HTML, the SRC attribute of the <SCRIPT> tag lets you specify a file as the JavaScript source. For example:

```
<SCRIPT LANGUAGE="JavaScript" SRC="common.js"></SCRIPT>
```

This is especially useful for sharing the same JavaScript code among many different pages. Note the ending </SCRIPT> tag is necessary to create a valid HTML script object! The SRC attribute can specify any URL, relative or absolute. For example:

```
SRC="http://www.mysite.com/functions/jsfuncs.js"
```

External JavaScript files cannot contain any HTML tags: they must contain only JavaScript statements and function definitions. External JavaScript files should have the file name suffix .js.

Task 2: Open a new empty file.

Put the JavaScript you used in Task 1 into this file (you can use copy and paste) and save this file. Change the Web Page you made in Task 1, so that it calls the JavaScript from a code file, instead of having the code embedded.

### *D.1.2. JavaScript Objects and the Document Object Model*

JavaScript is an object oriented language, like Python, C++, Java and many others. There are several types of objects you can use:

*JavaScript core language objects:* these are objects that are part of the JavaScript language. Examples are the Date object that can be used to manipulate dates and times, the String object for manipulating text strings and the Math object for manipulating numbers.

*Browser or Document Object Model (DOM) objects:* the parts of the browser and web page that are made accessible to the scripting language are structured in the so-called Document Object Model (DOM); this is a structured hierarchical object tree that provides a way to control elements on the page using scripting, provides multimedia controls for animations and other effects, and provides a way to bind data to an HTML page. Unfortunately, the browser manufacturers have not yet agreed on one common DOM, therefore there are small differences in the DOM for different browsers. The scheme in figure D.1 shows you the structure of the lowest common–denominator DOM. The DOM objects can be used to manipulate the HTML webpage, and turn it into DHTML, as shown in the examples in further sections.

#### *D.1.2.1. The document.write method*

As you saw in the previous example, the write() method of the DOM object document displays output in the browser, and its syntax follows JavaScripts standard object notation: `objectName.methodName(arguments)`, where `object-`

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



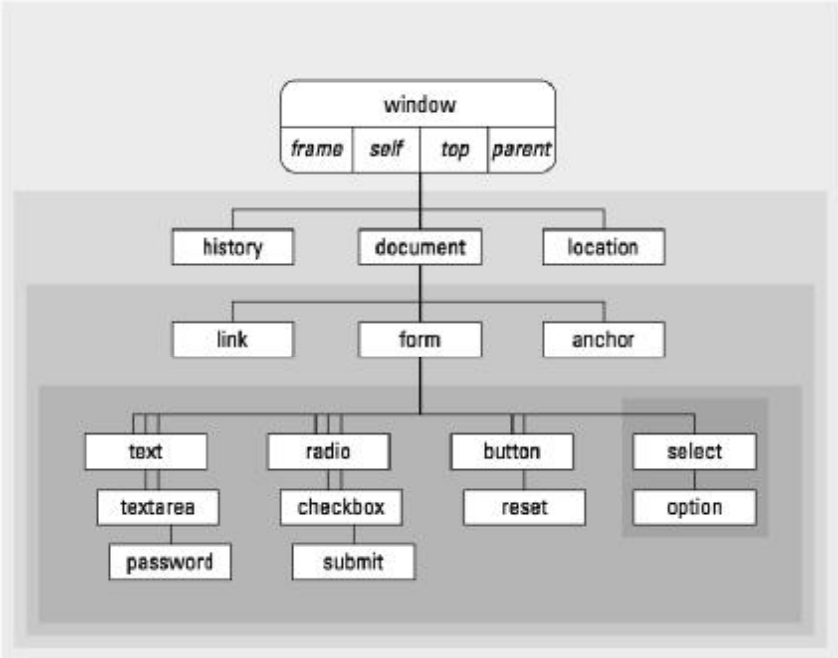


*Fig. D.1: The lowest common–denominator DOM.*

`Name` is the name of the object, `methodName` is the name of the method, and `arguments` is a list of arguments to the method, separated by commas.

"Big deal," you say, "HTML already does allow me to write text in a web page." But in a script you can do all kinds of things you can't do with ordinary HTML. For example, you can display text conditionally or based on variable arguments. For these reasons, write is one of the most often–used methods. The write method takes any number of arguments, that can be string literals (a text between quotes that will be show literally), or variables, that is any variable that evaluates to a string value. You can also use the string concatenation operator (+) to create one long string from several separate ones, thus after running the script fragment:

```
myNameString = "Barend Kobben";
myStringVar = "My name is " + myNameString;
```

myStringVar would have the value My name is Barend Kobben.

| Task 3: Write a script that shows the line "Hello, it is now:..." , where the ... part is showing the current date and time. |
|---|

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



To achieve this, you should use:

- the method write() of the DOM object document;
- the JavaScript built–in method `Date()` that returns the current date and time.

#### *D.1.2.2. The document.getElementById method*

Another much used DOM method was already used in the first section: The `getElementById` method of the document object. This finds an HTML object by its unique ID, thus creating a programmable object. You can assign an ID to virtually all HTML objects, by simply adding an attribute `id="someID"` to it. You have to take care yourself the ID is unique!

### *D.1.3. Using functions*

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure — a set of statements that performs a specific task. A function definition has these basic parts:

- The function keyword;
- A function name;
- A comma-separated list of arguments to the function (in parentheses);
- The statements in the function {in curly braces}.

It's important to understand the difference between defining and calling a function. Defining the function simply names the function and specifies what to do when the function is called. Calling the function actually performs the specified actions with the indicated parameters.

Task 4: Open a new empty WWW-page.

Type the code shown in listing 2 into the HTML-editor window. Save the page and open it in a web browser.

*Listing 2: A simple function*

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
function square(number) {
return number * number;
}
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
  document.write("The function returns: ", square(5), ".");
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
</SCRIPT>
<P>All done.</P>
</BODY>
</HTML>
```

This script defines a simple function in the HEAD of a document and then calls it in the BODY of the document: The function `square` takes one argument, called number. The function consists of one statement: `return number * number;`. The return statement specifies which value will be returned by the function, so this means the function will return as result the argument of the function multiplied by itself. In the BODY of the document, the statement `square(5)` calls the function with an argument of 5. The function executes its statements and returns the value 25.

We now have some basic working knowledge of Javascript. Now let's put it to use in an actual web-mapping application!

### D.2. Open Data: using OpenStreetMap

First we'll show you a prime source of free maps and data on the web: Open-StreetMap.

Note: The OpenStreetMap Project, based at openstreetmap. org, is the world-wide mapping effort that includes more than 400,000 volunteers around the globe. OpenStreetMap is an initiative to create and provide free geographic data, such as street maps, to anyone.

There are many ways in which you can access the OpenStreetMap: as a simple webmapping service (not unlike Google and Bing Maps, but based on truly free non-proprietary data on a non-commercial website), as a webservice in various GIS viewers and as a database service, providing the actual vector data in raw form.

| Task 5: Visit http://www.openstreetmap.org/ using a web browser.<br>Try to find the location you are currently in, by zooming and panning. |
|---|

The OpenStreetMap site itself uses the OpenLayers Javascript API, just as we will do later ourselves. The icons you see in the map are the default Graphics User Interface (GUI) of OpenLayers. They offer the following interactivity:

- You can pan using the arrow icons, or by dragging the map;
- You can zoom in using the + icon, or shift–drag a zoom box in the map;
- You can zoom out using the – icon;

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Using the OpenStreetMap site as described above is fine for casual map browsing, but if you want to disseminate spatial data yourself, you might want to include the OpenStreetMap in your own websites. And of course that is possible. We will use a popular browser–based mapping library for creating such a site: OpenLayers.

### D.3. The OpenLayers API

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles and markers loaded from any source.

Note: OpenLayers is a pure JavaScript library for displaying map data in most modern web browsers, with no server-side dependencies. OpenLayers implements a JavaScript API (Application Programming Interface) for building rich web-based geographic applications, similar to the Google Maps, with one important difference: OpenLayers is Free Software, developed for and by the Open Source software community based at http://openlayers.org/. OpenLayers is written in object-oriented JavaScript, using components from Prototype.js and the Rico library. In this exercise, we will only show the basic building blocks, and how to employ them. Those wanting to go further, should check out the development pages and the examples at the website.

The latest version of the OpenLayers script library is always available on the OpenLayers website. You can "install" the API by including a link to the Javascript files in your own HTML web-pages and then call the methods and properties of the library using simple JavaScript functions. Using the Openlayers API is done by creating webpages (using HTML) that include Javascript script; this code makes calls to the API methods to create the necessary map object and connect that to an HTML placeholder. Mostly we use an HTML  element as a placeholder.


The OpenLayers API has two concepts which are important to understand in order to build your first map: *Map*, and *Layer*. An OpenLayers *Map* stores information about the default projection, extents, units, and so on of the map. Inside the map, data is displayed via *Layer*s. A *Layer* is a data source – information about how OpenLayers should request data and display it. We then uses the methods and properties of the API to change the content and behaviour of the map. In practice, all this means typing (and/or copying) HTML and JavaScript code.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### D.4. Making an OpenLayers viewer for OpenStreetMap

In listing 3 you see the most basic example of using OpenLayers with the OpenStreetMap service.

> Task 6: Create an HTML page with the content of listing 1 and save it as a os-m.html.
>
> You can type the code, but it is easier to copy it from the file we stored in the filefragments folder. Do not copy from this PDF file! Make sure you save this file as a new file with the extension .html, not .html.txt!).
>
> View the result in a web browser.

*Listing 3: osm.html*

```
<!DOCTYPE html><html>                                    html document starts
<head>                                                   html header starts
   <script
src="http://localhost/openlayers/OpenLayers.js"></script>     load
                                                                the API
   <script type="text/javascript">                       script for our map
      var myMap, myOSMLayer;                    define map and layer object
      var myCenter = new OpenLayers.LonLat(                  define center
         254031,6254016                                       XY of Paris
      );
   function init() {                              function triggered on load
   myMap = new OpenLayers.Map("mapDiv");                  create map object
   myOSMLayer = new OpenLayers.Layer.OSM("OSM Map");      create OSM
                                                                layer
   myMap.addLayers([myOSMLayer]);                        add layer to map
   myMap.setCenter(myCenter,16);                          zoom to center
   }
   </script>
</head>
<body onload="init()">                                        run init script
   <div id="mapDiv"                                        map placeholder
   style="width:400px; height:400px;"></div>             placeholder style
</body></html>
```

The result should look like figure D.2, showing the OpenStreetMap for the Porte Maillot area in Paris (France).

You can set up the OpenStreetMap to start at any place on the globe, by changing the coordinates that were used in the myCenter variable:

```
var myCenter = new OpenLayers.LonLat(254031,6254016);
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



But in order to find which coordinates to use to zoom to, it would be nice to have a knowledge of where (in coordinates) you are in the map. For that we will include a coordinate readout line and a scale bar:

Task 7: Add the following line in the script just before the line with the myMap.SetCenter command:

```
myMap.addControl(new    OpenLayers.Control.MousePosition());
myMap.addControl(new OpenLayers.Control.ScaleLine());
```

Save the results as osmPlusCoordinates.html. Try out the result in the browser.

The coordinates you see are X– and Y–coordinates in a Mercator projection on the spherical WGS84 datum. This is used nowadays by most popular public webmapping services (such as Google Maps, Bing Maps and OpenStreetMap). The projection is officially standardized as EPSG code 3857, and named "WGS 84 / Pseudo-Mercator". Unfortunately, lots of software uses instead the un-official EPSG code 900913 (chosen because it sort of spells "google"), that was introduced and has become popular before the official standard was set.

Now you can change the line myMap.setCenter(myCenter,16) to set an alternative starting point (change myCenter variable) and zoom (from 0–18) for the map.

Task 8: Try setting up the map in such a way, that it starts zoomed in on your current location.


*Fig. D.2: result of loading listing 1.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# Creating data in a desktop GIS using Quantum GIS

*Version 2.0 - September 18, 2012*



*Key points*

The web page we made in the lesson about using Open-StreetMap in Open-Layers is nice, but what if we want to add our own data to that map? In order to do that, we shall first have to create such data. We will do that using the same OpenStreetMap as a reference and using QGIS, a free Open Source GIS.

Note that this exercise assumes you are using the OSGEO LIVE system, and that you have previously done the exercise “Using OpenStreetMap in an OpenLayers website”.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## E.1. Introduction to QGIS

QGIS (officially QuantumGIS, http://qgis.org) is an Open Source, stand-alone GIS client, programmed in C++ using the multi-platform Qt framework. You can use it to work with vector- and raster files, databases or any open standard WMS or WFS compliant server. A strong point of QGIS is its extensibility: you can add plugins that are written in either C++ or Python, and you can connect it to GRASS, a powerful GIS analysis tool.

Task 1: ITC users can just double click (or copy) the shortcut that they find at P:\QuantumGIS\Quantum GIS to their own computer.

If you are using the OSGEO LIVE system, you can start QGIS by going to the menu Geospatial > Desktop GIS > Quantum GIS.

Other users go to the URL http://qgis.org/ and download and install the latest (stable) version for their operating system.

QGIS can load maps and data from a huge array of possible sources:

- online maps served as an OGC-compliant Web Map Service (WMS);
- online spatial data served as an OGC-compliant Web Feature Service (WFS) and Web Coverage Services (WCS);
- various other map services, such as OpenStreetMap, Google Maps, Bing Maps, etcetera;
- most vector formats supported by the OGR library, including ESRI shapefiles, MapInfo, KML, GPX and GML;
- raster formats supported by the GDAL library, such as digital elevation models, aerial photography or satellite imagery;
- spatially-enabled PostgreSQL tables using PostGIS and SpatiaLite, by means of a “live” connection to such databases;
- locations and mapsets from GRASS (an open source GIS);

The list is in principle endless, because the functionality of QGIS can be extended by plugins. Plugins add functionality to QGIS, and they are usually made by others than the main QGIS developers. Because QGIS is an Open Source software, anyone can add plugins, they can be programmed using C++ or Python.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## E.2. Connecting QGIS to Web Map Services

Task 2: Choose the `Layer > Add WMS Layer..` menu or click the WMS layer icon. You can choose your `Server Connection` from the list.

This list can be initially filled with some well-known services by clicking "`Add default servers`". Try to find the service provided by LizardTech server. Once you have selected it and clicked the Connect button, the software sends a so–called GetCapabilities request to the WMS. The resulting XML description of the capabilities of the service will be parsed by QGIS and it will display the layers that were advertised as being available.

Try to find and load a layer that displays a satellite image of the whole earth. This will usually be the so called BMNG, the "Blue Marble Next Generation". This is a dataset created from MODIS satellite images by NASA JPL.

You can also add your own WMS connections to this list, e.g., to the ITC WMS services:

Task 3: Choose again the `Layer > Add WMS Layer...` menu (or click the WMS layer icon). Now add your own server connection by clicking "`New`". In the Name field you can give any name for the connection you want. In the URL field you put the so called root URL, where the software can find the GetCapabilities interface of the service.

Use this URL for a general world map provided by ITC WMS services:

```
http://geoserver.itc.nl/cgi-bin/mapserv.exe?
map=D:/Inetpub/geoserver/mapserver/ config_world.map&
```

Now that connection is added, you can use it like the previous one to retrieve capabilities and load WMS layers.


*Fig. E.1: Add layers from a server.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Using these techniques, you can use any WMS available on the internet or your local intranet. The only thing you need to know is the the root URL, where the GetCapabilities interface of the service can be found. E.g., if have learned how to use ArcGIS Server to publish a OGC-compliant WMS from an ArcMap (.mxd) file, or if you have learned how to use Geoserver: To access these services in QGIS, again you would use the "base URL" to the service.

### E.3. Using OpenStreetMap as a layer in QGIS

The extensibility of QGIS can serve us well: we can use several existing geo-webservices as background layers by means of the "Open-Layers Plugin":

Task 4:

Start QGIS. To enable the use of OpenStreetMap, we will enable the plug–in that offers that functionality: Open the menu `Plugins > Manage Plugins...` From the list, find the one called "OpenLayers plugin" and enable it by making sure it is selected. Press OK.

Now open the menu `Plugins` again. A new item called `OpenLayers Plugin` should have been added. Choose `Add OpenStreetMap layer` from this sub-menu, and the OpenStreetMap will be opened, zoomed out on the whole world.

Navigate to the location you saved for webpage you made earlier (your own place or the Porte Maillot).

QGIS is not limited to OpenStreetMap, it can load maps and data from a huge array of possible sources, such as Google, Bing and Yahoo Maps.

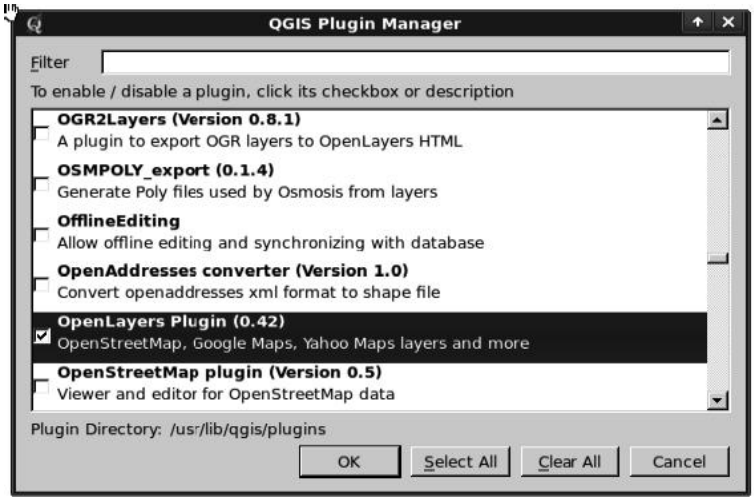


*Fig. E.2: QGIS PlugIn Manager.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



It also allows you to create data in many of these formats, and that is was we are going to do next...

### E.4. Using QGIS to create your own data

We will create a very simple vector line dataset, that depicts a walking or running route. You could for example try to digitize the road you took this morning from your hotel to the conference location...

When creating new data, QGIS will adopt the projection of the current project. Therefore we must make sure that the data is saved correctly projected, otherwise our new layer won't fit the OpenStreetMap base later on:

Task 5: Choose the menu `Layer > New > New Shapefile Layer...`

The New Layer dialog opens (see below). Make the following settings:

1. For Type choose Line;
2. Click Specify CRS and make sure you choose the Google Mercator (EPSG:3857 or EPSG:900913). It can be found under `Projected Coordinate Systems > Mercator`;
3. In the New attribute section, create one attribute, named routeName, of type Text. Click the Add to attribute list button to actually add it;
4. Click OK to create the new file. Save it on your Hard Disk or the USB stick and name it "myRoutes.shp" [if this is not possible with your configuration, you will be have to later use the prepared file at the website].

Now you can start adding lines for your route:

Click first the `Toggle Editing` button in the QGIS menubar, then the `Capture Line` button. Create a nice walking route, e.g. one near the conference centre. You can add points to the line by clicking in the map, undo them by using the CTRL-Z key or `Edit > Undo` menu.

If you have finished a route, right-click, fill in the route name and press OK. Use the `Toggle Editing` button again to stop editing. You can change the visualisation of the line by right-clicking the layer name in the layers list, or choosing the `Layer > Properties` menu.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



You now have created your own data. But only you can look at it, locally using your GIS viewer. In order to publish this in your web site, you'll have to turn the data into a webservice. We will do that by creating an Open Standard WMS service, and we will use the MapServer and/or GeoServer software to publish it.

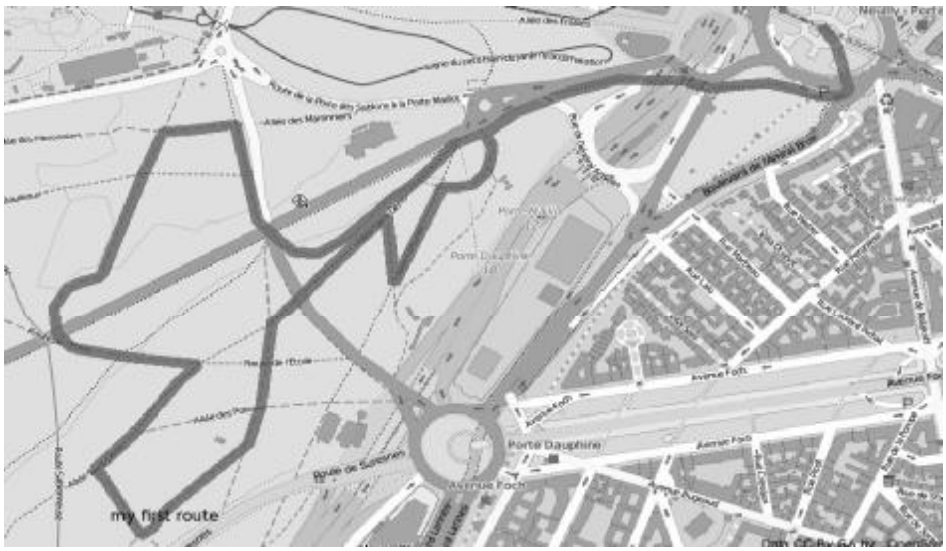


*Fig. E.3: Example of own data input.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# F. Serving Data as an OGC Web Map Service Using WMS with UMN Mapserver

*Version 1.0 - September 18, 2012*



*Key points*

In this exercise you wil learn how to set up the software MapServer as a Web Map Service using the Open Geospatial Consortium's OWS standards. This exercise is geared towards using it in the OSGEO LIVE DVD environment.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## F.1. Setup of MapServer for the OSGEO LIVE environment

### *F.1.1. Setup parameters*

The MapServer software comes pre-installed on the OSGEO LIVE environment. You will place all the MapServer configuration files, templates etc., inside the special exercise directory "thailand" that is also pre-instaled or given to you by the workshop teachers. You can use only this location!

These 3 parameters will be needed repeatedly in the following exercises:

```
<CGIPATH> = http://localhost/cgi-bin/mapserv
<NETPATH> = /home/user/thailand
<URLPATH> = http://localhost/thailand
```

The <CGIPATH> points tothe location where the webserver can reach the MapServer CGI software.

The <NETPATH> points to your map configuration files as they should be read by the MapServer.

The <URLPATH> points to the web location where a browser can find the output of the MapServer, e.g., HTML files, temporary image files or HTML templates.

### *F.1.2. Installing the exercise data*

You will have to make sure the data for the Thailand OWS exercises is installed. This data might be already in a folder "thailand" on the desktop. Otherwise, it can be found on the CD/DVD/USB stick you were given, and the workshop teachers will tell you how to install it.

Task 1: To test if the Apache web server is running properly, open your Web browser and find your local host Web service by entering the following URL:

```
http://localhost/.
```

Localhost is a shortcut telling your browser to connect to the local machine as a server. You should now see the main OSGEO LIVE page in your Web browser. This gives you general information about your installation along with configuration information for the OSGEO Live applications.

Now point a browser to the URL: `<URLPATH>/index.html`

A web page with the text "This is the home page for the Thailand OWS exercise site..." should appear.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



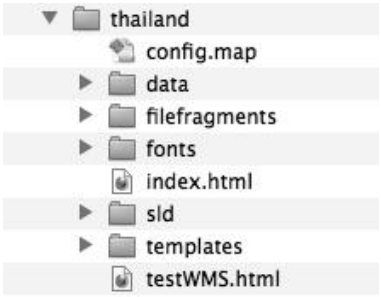


*Fig. F.1: Structure of exercise data folder "thailand".*

## F.2. Introduction of MapServer WMS

A Web Map Service produces maps of georeferenced data. A "map" is just a visual representation of geodata; a map is not the data itself. These maps are generally rendered in a pictorial format such as PNG, GIF or JPEG, or occasionally as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

In these exercises we'll use the software called MapServer for implementing the WMS. MapServer is an Open Source (OS) application for constructing spatially enabled web sites. It builds upon several existing popular OS libraries ((e.g., Proj4, GDAL/OGR) and runs on *nix systems (Linux/Unix/MacOSX) as well as Windows. It was originally developed in 1994 at the University of Minnesota (USA), funded by NASA. It can be connected to numerous spatial data stores: Vector support is built–in for Shapefiles, ArcSDE , Oracle Spatial and PostGIS, with the OGR module supporting many other formats. Raster support goes through the GDAL module supporting more than 30 formats (e.g. TIFF/GeoTIFF, EPPL7, ECW, Erdas). It can be set up to deliver data according to the following OGC specifications: WMS, WFS (not WFS–T), WCS and SLD and WMCD.

MapServer is a so called Common Gateway Interface (CGI) application, a standard for interfacing external applications with web servers. Executed on basis of an URL Request sent to the web server, it is able to provide dynamic/real time information.

### *F.2.1. 2.1 Basic parameters of a map file*

One configures the MapServer CGI service by providing it a map file, a text file with the .map suffix that defines an object tree in a hierarchical structure. You will learn how to set up such map files in the next sections. To get an overall view about the MapServer .map 'language', please take a look at the "MapFile Reference" on the ITC Blackboard system or directly at the mapserver website in http://mapserver.org/documentation.html

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



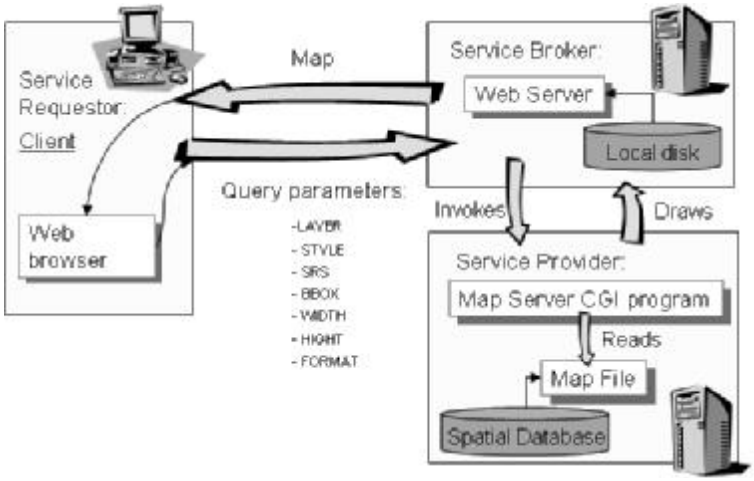


*Fig. F.2: Setup of MapServer CGI*

You can now start to create your Map File which will control what the MapServer does with your GIS dataset:

Task 2: Open the (empty) text file called `config.map` inside the directory `thailand`, using a text editor like WordPad or TextPad (do NOT use MS Word or another wordprocessor!).

Put the code in listing 1 into this file and save. See the note on page 1 about entering and copying code!

Make sure you've saved the file as config.map, NOT as config.txt or config.map.txt .

*Listing 1: filefragments/config.map.txt*

```
MAP
NAME "Thailand"
IMAGECOLOR 255 255 255
SIZE 600 800
IMAGETYPE PNG24
PROJECTION
   "init=epsg:4326" #latlon on WGS84
END
EXTENT 97.35 5.61 105.65 20.47 # lon/lat extents of Thailand
WEB
   IMAGEPATH "c:/tmp/ms_tmp/"
   IMAGEURL "/ms_tmp/"
   METADATA
      "ows_enable_request" "*"
      "map" "<NETPATH>/config.map"
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
     "ows_schemas_location" "http://schemas.opengeospatial.net"
     "ows_title" "Thailand WMS"
     "ows_onlineresource" "<CGIPATH>?map=<NETPATH>/config.map&"
     "ows_srs" "EPSG:4326 " #latlon
     "wms_feature_info_mime_type" "text/plain"
     "wms_feature_info_mime_type" "text/html"
     "wms_server_version" "1.1.1"
     "wms_formatlist" "image/png,image/gif,image/jpeg"
     "wms_format" "image/png"
   END #metadata
END #web
LAYER
   NAME "forest"
   TYPE POLYGON
   STATUS ON
   DATA data/forest
   METADATA
      "ows_title" "forest"
   END #metadata
   PROJECTION
      "init=epsg:4326"
   END
   CLASS
      NAME "forest"
      OUTLINECOLOR 255 255 255
      COLOR 137 205 102
   END #class forest
END #layer forest
END #map
```

Map Files are the basic configuration mechanism for MapServer. Anything associated with a particular application is defined here. The various parts are:

Layer Name and ows_title metadata: Every individual layer needs its own unique name and title. Layer names are also used in GetMap and GetFeatureInfo requests to refer to layers that should be included in the map output and in the query. Layer names must start with a letter when setting up a WMS server (layer names should not start with a digit or have spaces in them).

PROJECTION and ows_srs: WMS servers have to advertise the projection in which they are able to serve data using EPSG projection codes. In the mapfile, you define the default projection like this:

```
PROJECTION
  "init=epsg:4326"
END
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Then later in the WEB objects of the map and its individual layers, you can advertise which projections should be offered by the OWS interfaces, next to the default one, by specifying one or more EPSG code in the ows_srs parameter. Note that when asked for a SRS that's not the default one, Mapserver will have to re–project the data before turning it into a map. A good place to find which EPSG code to use for your projection is http://spatialreference.org/.

By default layers inherit the SRS of their parent layer (the map's PROJECTION in the MapServer case). For this reason it is not necessary (but still strongly recommended) to provide PROJECTION and ows_srs parameters for every layer. If a layer PROJECTION is not provided then the top-level map PROJECTION will be assumed. Layer PROJECTION and ows_srs metadata are defined exactly the same way as the map's PROJECTION and ows_srs metadata.

The WEB object: The WEB object defines how a web interface (such as OGC WMS) will operate. The METADATA keyword allows arbitrary data to be stored as value pairs. This is used with OGC WMS to define such things as ows_title, ows_onlineresource and ows_srs. If you want to combine layers with different Spatial Reference Systems, you can list several EPSG codes as shown above.

Per MAP, one or more LAYER objects: The LAYER object describes layers used to make up a map. Layers are drawn in their order of appearance in the Map File (first layer is on the bottom, last is on top). The TYPE specifies shows how the data should be drawn. It normally is the same as the data type. The DATA parameter specifies the path from your "config.map" file to the data file of your layer (e.g., a shapefile). Later we will learn how we can add other data sources (such as PostGIS database connections or raster files).

Per LAYER, zero or more CLASS objects: Embedded in the LAYER object are CLASS objects which defines the thematic classes for a given layer. How classes work and what you can do with them you will see in the next sections. You can make layers without classes, if you do not need styling (e.g. if you get styles from external styling such as SLD, or if you are making a WFS or WCS).

Per CLASS, zero or more STYLE objects: If you need classes with more than the simplest styling, you can embed STYLE objects to more freely define class styles.

Below you will find the list of parameters and metadata items that are optional for MapServer in general, but are required for a WMS configuration.

At the map level:

- Map NAME
- Map PROJECTION
- Map Metadata (in the WEB Object):
    - ows_title

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



- ows_onlineresource. The ows_onlineresource specifies the URL that should be used to access your server. This is required for the GetCapabilities output.
- ows_enable_request. This configures which of the OWS service interfaces will be exposed. "*" will allow them all.
- ows_srs

And for each layer:

- Layer NAME
- Layer PROJECTION
- Layer Metadata
  - ows_title
  - ows_srs (optional if you want the layers to inherit the map's SRS value)

### *F.2.2. Testing your Map File*

To test if your "config.map" file is actually working, we will now request the WMS services from the MapServer application, through OGC GetCapabilities and GetMap requests. To display the map server settings using the GetCapabilities request:

Task 3: Type the following URL in a web browser:

```
<CGIPATH>?map=<NETPATH>/config.map&SERVICE=WMS&VERSION=1.1.1&
→REQUEST=GetCapabilities
```

Note again the <CGIPATH> and <NETPATH> parts that have to be substituted with the proper paths for your installation! In this URL, you REQUEST the service operation you want to perform, in this case the OGC WMS standard request GetCapabilities. The VERSION parameter specifies the protocol version number. This request will return an XML document.

Look at the XML reply of MapServer. Depending on your web browser setup, this XML might already be automatically displayed inside your browser after the GetCapabilities request. In other setups, the browser might not recognise the XML file as readable output, and there might have been a prompt to open or save the file. In that case, save the file and open it in a Text Editor or an XML editor.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



To display an actual map using the GetMap request:

Task 4: Type (or copy) the following URL in a web browser:

```
<CGIPATH>?map=<NETPATH>/config.map&SERVICE=WMS&VERSION=1.1.1&
→REQUEST=GetMap&LAYERS=forest&STYLES=&SRS=EPSG:4326&
→BBOX=97,5,106,21&WIDTH=600&HEIGHT=800&FORMAT=image/png
```

- With REQUEST you define the operation you want to perform
- In LAYERS you list all the layers you want to be displayed
- the STYLES parameter is used to associate styles with the layers. If the styles list is empty, Mapserver will use the default styles defined in the map file
- In the SRS parameter you define which Spatial Reference System you want used
- With the BBOX you set a particular Bounding Box to be requested
- The WIDTH and HEIGHT parameters specify the size in pixels of the map image to be produced.
- With the FORMAT parameter you state the desired format of the map image

### F.3. Making class maps

If you have attributes of your layers you would like use to display different classes within your layer (e.g. using different colors, a so called chorochromatic map), you should add more CLASSes and separate them as follows:

Task 5: In the config.map file, change the part from CLASS until END to the text in listing 2. Check if you now have four CLASS objects (not five, as the original should have been removed!)

Test the new map.

*Listing 2: filefragments/forest.txt*

```
CLASSITEM "FOR_TYPE" #use this attribute for classifying
CLASS
   NAME "Forest Reserve"
   EXPRESSION "FR" #if for_type=FR
   COLOR 56 168 0
END
CLASS
   NAME "National Park"
   EXPRESSION "NP" #if for_type=NP
   COLOR 171 205 102
END
CLASS
   NAME "Wildlife Sanctuary"
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
   EXPRESSION "WS" #if for_type=WS
   COLOR 245 245 122
END
CLASS
   NAME "Not forest"
   EXPRESSION "NF" #if for_type=NF
   COLOR 200 200 200
END
```

What you see in this example is the layer class of "Forest" was split up into different types. The name of the column in the attribute table of this shapefile to be used for the classification is FOR_TYPE. With the parameter CLASSITEM you tell MapServer which item in the attribute table it has to use in separating the class object. And with EXPRESSION in each CLASS object you define which attribute values will become part of this class: if testing the EXPRESSION returns TRUE, the feature will be mapped using this CLASS.

With COLOR you define the fill–colour of your object, defined in this case as three 8-bit values (from 0–255) for Red, Green and Blue respectively. 0 0 0 is black, 255 255 255 is white, 255 0 0 is full red, etc.

For the stroke–colour you can use OUTLINECOLOR R G B, but here we have not done that, to have polygons without a stroke.

### F.4. Hiding and showing layers at predefined scales

If you would like some layers not to appear until a special scale you can give them a MAXSCALE. Let's add a layer for waterbodies, that should only show if the scale is larger than 1 : 1,000,000:

> Task 6: Add the code in listing 3 after the forest LAYER (but before the last END, which ends the MAP object).
>
> Use the same GetMap request you did earlier, except for changing the LAYERS=forest part to LAYERS=forest,waterbody

*Listing 3: filefragments/waterbody.txt*

```
LAYER
   NAME waterbody
   TYPE POLYGON
   STATUS ON
   MAXSCALE 1000000
   DATA data/waterbody
   METADATA
      "ows_title" "Waterbody"
   END
   PROJECTION
      "init=epsg:4326"
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
  END
  CLASS
    NAME "Waterbody"
    COLOR 0 0 255
  END
END # layer waterbody
```

Note that the waterbody is not displayed initially. So you'll have to zoom in until you reach that scale:

Task 7: Use the same GetMap request you used earlier, expect for changing the `BBOX=97,5,106,21` to `BBOX=100,15,101,17`.

This means you'll zoom into the area between the longitudes 100 – 101 and between the latitudes 15 – 17. Of course this is not a very user–friendly way of zooming in, but in later exercises we will see how the `BBOX` can be set by a client–application instead of by typing in the URL.

### F.5. Creating symbols

Below we explain some of the basic ways of creating symbols. Symbols in MapServer are very powerful tools, but can also become quite complex. Basically, you add one or more STYLE objects to a CLASS to define the symbolisation. STYLEs can refer to separate SYMBOL objects that define individual symbols to use in one or more styles. For an extensive explanation of creating cartographic symbology in MapServer, refer to the document "Constructing Cartographic Symbols" on the MapServer documentation pages at http://mapserver.org/

#### *F.5.1. 5.1 Line Symbols*

A simple way to define line symbols in MapServer is to use only default lines and specify line width and colour:

Task 8: Add the code in listing 4 after the waterbody LAYER.

Use the same GetMap request you did earlier, expect for changing the `BBOX` back to the original value and changing the `LAYERS` part to `LAYERS=forest,waterbody,railroad`

*Listing 4: filefragments/railroad.txt*

```
LAYER
  NAME railroad
  TYPE LINE
  STATUS ON
  DATA data/railroad
  METADATA
    "ows_title" "railroad"
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
   END
   PROJECTION
      "init=epsg:4326"
   END
   CLASS
      NAME "railroad"
      STYLE
         COLOR 0 0 0
         WIDTH 2
      END #style
   END #class
END #layer railroad
```

### *F.5.2. Simple Point Symbols*

In order to draw point symbols which are more then a single pixel you will have to include a SYMBOL object. SYMBOL objects are defined on the hierarchical level directly below the MAP object. This way, several layers can use the same symbols. In your LAYER object you have to include a STYLE object that addresses the SYMBOL you created. The next example creates a simple filled circle. Using non-equal values for the point will actually render an ellipse.

Task 9: Add the code in listing 5 after the railroad LAYER. Note that the SYMBOL object could be moved to anywhere within the MAP file as long as it remains at the same hierarchical level (i.e., the level of the MAP object).

Use the same GetMap request you did earlier, now with

```
LAYERS=forest,waterbody,railroad,airports
```

*Listing 5: filefragments/airports.txt*

```
SYMBOL
   NAME "circle_sym"
   TYPE ellipse
   FILLED true
   POINTS
      1 1
   END
END #symbol

LAYER
   NAME airports
   TYPE POINT
   STATUS ON
   DATA data/airports
   METADATA
      "ows_title" "airports"
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
  END
  PROJECTION
     "init=epsg:4326"
  END
  CLASS
     NAME "airports"
     STYLE
        SYMBOL "circle_sym"
        SIZE 22
        COLOR 0 0 0
     END #style
  END #class
END #layer airports
```

The SYMBOL “name” in the STYLE refers to the NAME keyword in the symbol definition object. The default (when you do not specify any symbol) results in a single pixel, single width line, or solid polygon fill, depending on layer type.

### *F.5.3. Font-based Point Symbols*

To create symbols that are more than simple circles or squares, you can also specify a gif or png file by name, to use the graphic in that file as the symbol. The path of that file should be stated relative to the location of the mapfile. The disadvantage is that these symbols are raster elements and thus can not be scaled well.

A more convenient and better scaleable method for creating point symbols is the use of TrueType font marker symbols, just as software like ArcGIS does. To achieve that in MapServer, you have to create a font directory with the actual fonts and a fonts.list file which lists each of the available fonts and the names you want to address them by. As the ESRI symbol fonts are good to use for this purpose, we have copied one of them into the font directory and have included a font list for it (in the directory thailand/fonts).

Task 10: In your Map File you have to specify the FONTSET parameter which specifies the path to the font list file, just at the start of the MAP object:

```
MAP
 FONTSET "fonts/fonts.list"
… etc …
```

This points MapServer to a fontlist which maps fontnames to fonts in the system. Now use the fonts in a symbol definition: Add another SYMBOL into your MAP object as shown in listing 6.
Then change the SYMBOL name in the STYLE object of the airport LAYER to `"airport_sym"`. •

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



*Listing 6: filefragments/airportsymbol.txt*

```
SYMBOL
   NAME "airport_sym"
   TYPE TRUETYPE
   FONT "ESRI_Default_Marker"
   FILLED true
   ANTIALIAS true
   CHARACTER "o"
END #symbol
```

The CHARACTER is the glyph in the font's character encoding. It's the equivalent in a 'normal' text font of the character you are looking for in the symbol font. You'll need to figure out the mapping from the keyboard character to font character. When using Windows, you can find what glyphs to use for characters by using the "Character Map" application (usually found under Programs>Accessoires, or run charmap.exe in a DOS cmd-window).

### F.6. Labelling your map

Of course it is also possible to label objects in WMS. Next we will show how to add name labels to the airports:

> Task 11: Put a LABELITEM object before the existing CLASS definition of airports.
>
> Then put the LABEL code inside the existing CLASS object.
>
> It should end up looking like listing 7.

*Listing 7: filefragments/airportlabels.txt*

```
LABELITEM "NameEnglis" #attribute to use
CLASS
   NAME "airports"
   STYLE
      SYMBOL "airport_sym"
      SIZE 22
      COLOR 0 0 0
   END #style
   LABEL
      COLOR 0 0 0
      TYPE TRUETYPE
      FONT "arial"
      MINSIZE 6
      MAXSIZE 12
      POSITION AUTO
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
    PARTIALS FALSE
    BUFFER 4
  END #label
END #class
```

With LABELITEM you define which column in the attribute table of your shapefile has to be used for labelling. In the CLASS object you insert one (or more) LABEL objects to define how the text labels should be rendered:

- COLOR to draw the text in.
- TYPE of the font to use.
- FONT alias as defined in the FONTSET to use for labelling.
- MINSIZE tells which minimum font size to use when scaling the text.
- MAXSIZE tells which maximum font size to use when scaling the text.
- POSITION: Where to position the label text in relation to the label points. The value is a combination of vertical and horizontal positions. You have the following choices for vertical alignment: C for centre, U for upper, and L for lower. For horizontal alignment you have the following choices: C for centre, L for left, and R for right. So, to align the label text to the centre you'd use the value "CC" (centre–centre). Or if you'd like it to be on the lower left, you'd use "LL". Another way is to let MapServer decide the best position for your labels. For this you use the value AUTO.
- PARTIALS: Tells MapServer whether to generate incomplete label texts or not, e.g., when the label would cross the edge of the map, or if it overlaps a label already placed. The default is not to generate fragments of a label text (PARTIALS FALSE).
- BUFFER of 4 (pixels) means that no label will be drawn if another label already is within four pixels distance.

### F.7. Creating a legend

For a better understanding of your map it is necessary to add a legend in your map where the symbols you used are listed and named:

> Task 12: Create MapServer code for the legend as shown in listing 8.
>
> Put it in the MAP object, directly under it (in the hierarchy, so on the same level as the LAYERs).

*Listing 8: filefragments/legend.txt*

```
LEGEND
  KEYSIZE 16 12
  LABEL
    COLOR 0 0 0
    TYPE TRUETYPE
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
     FONT "arial"
     SIZE 12
END #label
STATUS EMBED
   POSITION UR
END #legend
```

The LEGEND object defines the look and placement of the legend drawing:

- KEYSIZE sets the size (in pixels) of the symbol key boxes
- LABEL set the type, the size and the color of the font you want to use for labeling the symbol keys (see description of LABEL above).
- STATUS EMBED means that the legend is embedded into the map image
- POSITION sets the place within the map image. In our case on the top–right side (U=upper, R=right, see POSITION described above).

### F.8. Creating a scale bar

A scale bar shows the user the scale of the current map:

> Task 13: Create MapServer code for the scale bar as shown in listing 9.
>
> Put it in the MAP object, directly under it in the hierarchy (so on the same level as the LEGEND).

*Listing 9: filefragments/scalebar.txt*

```
SCALEBAR
   IMAGECOLOR 255 255 255
   LABEL
      COLOR 0 0 0
      TYPE TRUETYPE
      FONT "arial"
      SIZE 8
   END
   STYLE 1
   SIZE 150 2
   COLOR 0 0 0
   UNITS KILOMETERS
   INTERVALS 2
   TRANSPARENT FALSE
   STATUS EMBED
   POSITION LR
END #scalebar
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



The SCALEBAR object parameters are:

- IMAGECOLOR defines the colour of the scalebar background
- LABEL you set the label options of the scale bar
- STYLE chooses the scale bar style. Valid styles are 0 and 1.
- SIZE is width and height in pixels
- UNITS define the output units of your scale bar (meters, kilometers, miles)
- The INTERVALS parameter defines in what number of intervals to break the scale bar into.
- TRANSPARENT sets the background transparency
- STATUS EMBED means that the scale bar is embedded in the map image
- POSITION is in this case is the lower left side (see above)

***Here ends the exercise...***

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# G. Serving data as an OGC Web Map Service
# Establishing WMS with Geoserver

*Version 1.0 - September 18, 2012*

G.1. Open Standards data dissemination: using GeoServer WMS

*Key points*

In this exercise you will learn how to set up the software GeoServer as a Web Map Service using the Open Geospatial Consortium's OWS standards.

Note that this exercise assumes you are using the OSGEO LIVE system, and that you have previously done the exercise "Creating data in a desktop GIS using QGIS".

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### G.1. Open Standards data dissemination: using GeoServer WMS

GeoServer is a service: That means that it acts as a background application, listening for requests on the web. You configure it using a series of web pages. It is installed on the OSGEO Live DVD, but the service has to be started up first in order to work:


Task 1: In the Xubuntu menu, choose Geospatial > Web Services > Start GeoServer.

The server starts up and a browser window is opened to show the connection to the service, at the URL `http://localhost:8082/geoserver/web`

This is the "public" interface, in order to set up the services you will have to login. Fill in the username ("admin") and password ("geoserver") and click the login button. The administrator pages are loaded.

GeoServer (http://openserver.org) is an open source software server written in Java. Designed for interoperability, it publishes data from any major spatial data source using open standards. Being a community-driven project, GeoServer is developed, tested, and supported by a diverse group of individuals and organizations from around the world. GeoServer is the reference implementation of the Open Geospatial Consortium (OGC) WebFeature Service (WFS) and WebCoverage Service (WCS) standards, as well as a high performance certified compliant Web Map Service (WMS).

GeoServer uses a fairly elaborate setup: There are several Workspaces, that each can hold one or more DataStores. These connect the service to various datastores, either simple ones like vector- and raster files, or more complicated ones like spatialdatabases or other OGC services. Each datastore can in turn contain one or more Layers.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 2: We will use an existing Workspace, the default one called "it.geosolutions". In the Data menu, on the left side of the screen, click the Stores icon or name.

Now click the `Add New Store` icon. From the list of datastores, choose the on called `"Shapefile - ESRI(tm) Shapefiles (*.shp)"`.

Now fill in the dialog: For the Shapefile location, you can use the Browse... button to navigate to the shapefile you created earlier in QGIS [alternatively, you can use the pre-made one at our website]. Click the Save button.

Now click in the data menu (left side of screen) on the Layers item. In the next screen, choose the `Add new Resource` item.

From the list choose `it.geosolutions:myRoutes`, and in the list of layers that is shown next, click on the Publish link next to the myRoutes layers (should be the only one to choose from).

Now you can edit the Layer properties. Most fields should have been filled in correctly by default. Check if the projection is set correctly (to EPSG:3857 or EPSG:900913). Now fill in both the Native Bounding Box and Lat/Lon Bounding Box by clicking the `Compute from data` links under them. Then click the Save button.

Now you can test the Layer publishing by choosing Layer Preview in the Data menu (left side of screen). There are many tests here, the most easy being to click on the OpenLayers link.

Now your data is available for anyone on the internet that connects to one of the webservices that the GeoServer offers.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.




*Fig. G.1: Geoserver´s basic store info*

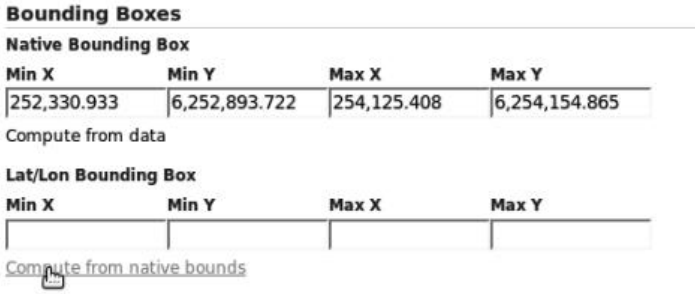


*Fig. G.2: Input of the Bounding Box*

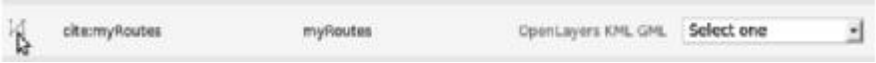


*Fig. G.3: Layer selection*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# H. Web Map Services in a web client: Mapserver WMS in OpenLayers

*Version 3.9 - September 17, 2012*



*Key points*

This is a reference for the use of OpenLayers, a JavaScript Library for web mapping. In this exercise we will us it to create a web page that gives youa ccess to Web Map Services, including the Thailand WMS you built in earlier exercises. You will learn how to:

1. Use the OpenLayers library in a web page;
2. Make a simple map using a WMS;
3. Add some layout and controls to it;
4. Add your own Thailand WMS data to it.
5. Query the WMS layers for attributes

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## H.1. The OpenLayers API

OpenLayers makes it easy to put a dynamic map in any web page. It can display map tiles and markers loaded from any source.

Note: OpenLayers is a pure JavaScript library for displaying map data in most modern web browsers, with no server-side dependencies. OpenLayers implements a JavaScript API (Application Programming Interface) for building rich web-based geographic applications, similar to the Google Maps, with one important difference: OpenLayers is Free Software, developed for and by the Open Source software community based at http://openlayers.org/. OpenLayers is written in object-oriented JavaScript, using components from Prototype.js and the Rico library. In this exercise, we will only show the basic building blocks, and how to employ them. Those wanting to go further, should check out the development pages and the examples at the website.


The latest version of the OpenLayers script library is always available on the OpenLayers website. You can “install” the API by including a link to the Javascript files in your own HTML web-pages and then call the methods and properties of the library using simple JavaScript functions. Using the Openlayers API is done by creating webpages (using HTML) that include Javascript script; this code makes calls to the API methods to create the necessary map object and connect that to an HTML placeholder. Mostly we use an HTML  element as a placeholder.

The OpenLayers API has two concepts which are important to understand in order to build your first map: Map, and Layer. An OpenLayers Map stores information about the default projection, extents, units, and so on of the map. Inside the map, data is displayed via Layers. A Layer is a data source – information about how OpenLayers should request data and display it. We then uses the methods and properties of the API to change the content and behaviour of the map. In practice, all this means typing (and/or copying) HTML and JavaScript code.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### H.2. Making an OpenLayers viewer for a WMS

In listing 1 you see the most basic example of using OpenLayers with a sample WMS.

Task 1: Create an HTML page with the content of listing 1 and save it as a wms.html. You can type the code, but it is easier to copy it from the file we stored in the filefragments folder. Make sure you save this file as a new file with the extension .html, not .html.txt!). View the result in the web browser.

*Listing 1: wms.html.txt*

```
<!DOCTYPE html><html>                                    html document starts
<head>                                                   html header starts
  <script
src="http://localhost/openlayers/OpenLayers.js"></script>      load
                                                                the API
  <script type="text/javascript">                        script for our map
    var myMap, myWMSBaseLayer;                  define map and layer object
    var myCenter = new OpenLayers.LonLat(0,0);  define center long, lat
                                                                function
    init(){ function triggered on load
      myMap = new OpenLayers.Map("mapDiv");              create map object
      myWMSBaseLayer = new OpenLayers.Layer.WMS(         create WMS
                                                                layer
        "world WMS", title of OL layer
        "http://geoserver.itc.nl/cgi-bin/mapserv.exe?
          map=D:/Inetpub/→
         geoserver/mapserver/config_world.map&",            baseURL
        {layers: "world"}                          WMS layer(s) to load
        );
      myMap.addLayers([myWMSBaseLayer]);              add layers to map
      myMap.setCenter(myCenter,2);             map at lon-lat(0,0) zoom 2
      }                                                 end of init script
    </script>
  </head>
<body onload="init()">                                    run init script
<div id="mapDiv"                                          map placeholder
  style="width:800px; height:400px;"></div>               placeholder style
</body></html>
```

The result should look like figure H.1, showing the simple WMS service of world countries, zoomed out at the world's center (longitude = 0, latitude = 0). You can set up the WMS map to start at any place on the globe, by changing the coordinates that were used in the myCenter variable:

```
var myCenter = new OpenLayers.LonLat(0,0);
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



*Fig. H.1: Result of loading listing 1.*

Note that the order is longitude -latitude! This is done to mimic the order X-Y when using projected (cartesian) systems. In order to find which coordinates to use to zoom to, it would be nice to have a knowledge of where (in coordinates) you are in the map. For that we will include a coordinate readout line and a scale bar:

Task 2: Add the following line in the script just before the line with the `myMap.SetCenter` command:

```
myMap.addControl(new OpenLayers.Control.MousePosition());
myMap.addControl(new OpenLayers.Control.ScaleLine());
```

Save the file and try out the result in the browser.

The coordinates you see are longitude and latitude in a degrees on the spherical WGS84 datum. The is officially standardized as EPSG code 4326. You can change the line `myMap.setCenter(myCenter,1)` to set an alternative starting point (change myCenter variable) and zoom (from 0–18) for the map.

Task 3: Try setting up the map in such a way, that it starts zoomed in on Thailand.

### H.3. OpenLayers: Adding layers

Open Layers allows the use of many layers in the same client, coming from different sources:

- OGC Web Map Services (WMS) & Web Feature Services (WFS);
- Other open geo-webservices (e.g., OpenStreetMap, GeoRSS, NASA WorldWind);
- Commercial geo-webservices like Google Maps, Yahoo, Microsoft Bing Maps, MultiMap;
- Client-side data such as GML, KML, marker-layers, drawing layers, text layers and others.

You define layers using the OpenLayers.Layer object:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
myNewLayer = OpenLayers.Layer.layerType( nameOfLayer, {listOfOp-
tions} );
```

The `layerType` defines what server- or client side layer is to be used. `{listOfOptions}` is an array of name:value pairs that are used to add a multitude of special settings. Depending on the `layerType`, zero or more options are required. We saw for example in the WMS `layerType` we used in section H.2 that a few additional options were needed. E.g. the `baseURL`, that determines where to find the WMS, and `layers`, a string that will be used to ask one or more layers from the WMS.

There is an important difference between what OpenLayers calls Base Layers and Overlays. There is always at least one base layer needed. It's the one that determines the projection, extent and units of the map. You can define several base layers if they share the same parameters (e.g., several layers of one WMS, or the various types of Google maps), but the user can always have only one turned on, the choice will be made using a radio-button list. Overlays are layers that can be fitted on top of the base layer. Because they can be transparent, many overlays can be used, and they each can individually be turned on or off (using a checkbox list). By default, new raster layers will become base layers, unless you have made sure they are transparent, in that case they can be overlays. Most vector layers are by default Overlays.

### *H.3.1. Adding Thailand WMS layers to the map*

Now let us add a layer using the Thailand WMS we made earlier in the MapServer exercises:

Task 4: First find the line that reads:

```
var myMap, myWMSBaseLayer;
```

and change it to:

```
var myMap, myWMSBaseLayer, railroadLayer;
```

This adds a necessary variable to hold the data for the WMS layer. Add the code below just after the layer definition you made earlier (for the WMS world map):

```
railroadLayer = new OpenLayers.Layer.WMS("Thailand RailRoad",
"<CGIPATH>?map=<NETPATH>/config.map&",  {layers: "railroad",
transparent: "true", format: "image/png"} );
```

Refer to the earlier MapServer exercises, to know how to substitute the <CGI-PATH> and the <NETPATH> for the proper values for your set-up!
Now look for the line in the code that reads:

```
myMap.addLayer([myWMSBaseLayer]);
```

To include the new layer in the map object change it to:

```
myMap.addLayers([myWMSBaseLayer, railroadLayer]);
```

Try it out in the browser...!

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



There is still one annoying problem: The legend and scale bar you defined in the earlier WMS exercise will be shown several times within the map. This is because OpenLayers uses a tiling mechanism to subdivide the WMS output in many small tiles. Each of these tiles will have a legend and scale bar, because you did set the LEGEND and SCALEBAR objects in the configuration file to STATUS EMBED.

Task 5: Open the config.map file and set the LEGEND and SCALEBAR objects to STATUS ON.

Try viewing wms.html in the browser again.

The legend (and scale bar) will now disappear, but we will make the legend appear again in a different way further on.

Task 6: Now add two more layers of the Thailand WMS: forest and airports. You should be able to figure out how to do that based on the previous tasks...

Now we have more than one layer, we want to control which of those are shown in our map. For that, add the line :

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

after the other addControl statements. Check the result.

A new control should have been added to the map: next to the pan and zoom tools, there now should be a little + icon in the upper-right corner. Clicking it will reveal a Layer Control, which can be used to switch layers on and off.

### *H.3.2. Enhancing the layout*

In order to have a place to put the legend in, we will make a more sensible layout than the one we had earlier. You can change the layout by setting the properties of the separate place-holders in HTML. Many HTML elements can be used as place-holders, one of the most simple to use is the  element we used earlier.

Task 7: Change the line:

```
style="width: 800px; height: 400px; "></div>
```

to:

```
style="position: absolute; left: 5px; top: 5px; width: 400px;
height: 600px; → overflow: hidden; border: 1px solid blue;
"></div>
```

The style properties are expressed using CSS, Cascading Style Sheets, a styling language standardised by the W3C, the World Wide Web Consortium. The meaning of the properties used is:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



- position: this lets you define positioning absolute or relative to other divs.
- left & top: the (absolute) location with respect to the upper left corner of the window
- width & height: the size of the element (in pixels)
- overflow: if the content is larger than fits the div, it will not be shown if this is set to hidden. Other settings are visible (will overflow), scroll (will make scrollbars) and auto (let browser decide).
- border: the border look (width, type and colour). You can also set the fill in a similar way.

### *H.3.3. Adding the legend*

Now add another section to the layout that will hold the legend for the Thailand WMS layers:

Task 8: Add a second  element for the legend:

```
<div id="legend" style="position:absolute; width:250px;
height:200px; left: →
420px; top: 5px; overflow: auto; border: 1px solid red; ">
<img src="<CGIPATH>?
map=<NETPATH>/config.map&SERVICE=WMS&REQUEST= →
GetLegendGraphic&VERSION=1.1.1&FORMAT=image/png&LAYER=forest">
<br/>
</div>
```

This should result in the layer legend being shown in the legend div. Now add the legends for the other layers in a similar fashion within the same div.

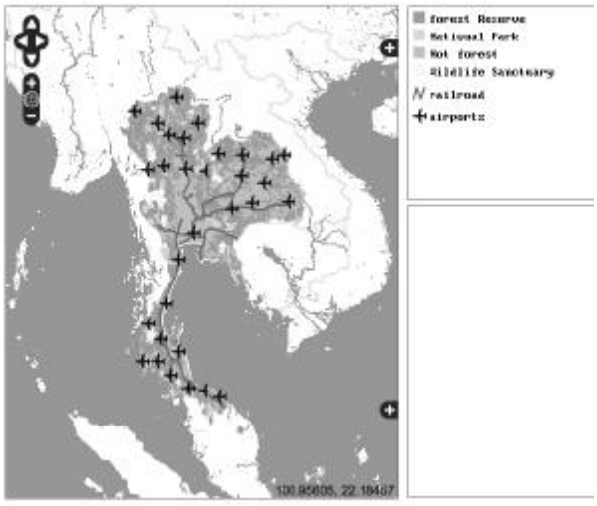


*Fig. H.2: After addition of the legend and query divs.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



The src parameter of the img tag issues a new OGC WMS request called GetLegendGraphic. This retrieves the legend of the LAYER requested as an image in the FORMAT requested from the WMS.

### H.4. Querying attributes from WMS layers

Above we issued an additional WMS GetLegendGraphic request to get the legend. We can use a similar mechanism to retrieve attribute data of the maps by issuing the OGC GetFeatureInfo request. This involves some more work, because the result will be some arbitrary content that will arrive after an unknown amount of time. For this we will use something called Asynchronous JavaScript.

First we have to prepare MapServer to actually be setup to respond to the request in a way the OpenLayers script can handle.

Task 9: Edit the config.map file. In each LAYER object you want to query, put the lines:

```
TOLERANCE 4
TOLERANCEUNITS pixels
```

In each LAYER object's METADATA put the extra line:

```
"wms_include_items" "all"
```

In each CLASS object put the extra line:

```
TEMPLATE "empty"
```

This "TOLERANCE" and "TOLERANCEUNITS" lines will make sure that an object is considered clicked upon, even if you are up to 4 pixels off. You will probably want to change these values later, to refine the results per layer.

The "wms_include_items" tells MapServer to show all attributes it has available. You can change it to a comma-delimited list of attribute names if you want to limit the output.

The "TEMPLATE" line is needed for GetFeatureInfo to work, but it does not have to point to an actual template document (that is why we put a "dummy" value in there). If you want to use a real template, to nicely format the GetFeatureInfo output, you let it point to an actual template HTML file (see the Mapserver documentation for more information).

Task 10: Now edit your html file: Put the code in listing 2 just at the end of the init() script (after the myMap.setCenter() line). •

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



*Listing 2: filefragments/getFeatureInfo.txt*

```
getFeatureInfoCntrl = new OpenLayers.Control.WMSGetFeatureInfo→
(  { infoFormat:"text/plain" } );
getFeatureInfoCntrl.events.register→
("getfeatureinfo", this, showQueryResult);
myMap.addControl(getFeatureInfoCntrl);
getFeatureInfoCntrl.activate();
```

The first statement creates a new Control of the "WMSGetFeature-Info" type. This will fire a GetFeatureInfo request to the WMS services associated with the map. You can override the default request parameters by providing them in the `{options}` object. Here we leave all of them at default except the `infoFormat: "text/plain"`. This request the output to be in plain text, instead of the default of GML.

The next line registers a so-called event-listener, that will be triggered every time anyone makes a "click" in the map object. This will send the GetFeature-Info request URL to the server and point to a callback function named `showQueryResult`. This function will be triggered later, if and when something comes back. We call this asynchronous processing. The last two lines add the control to the map and activate the event listener.

Now you will have to create the call-back function:

Task 11 : Put the code below at the end of the script (after the ending `}` of the init function, but before the ending tag `</script>`).

```
function showQueryResult(evt) { alert(evt.text); }
```

Test the result in a web browser. Do not open the file from your file–system (c:\ or m:\), but always from the actual web site, e.g. http://itcnt07.itc.nl/~ or http://localhost/. This is the only way to properly test your web site!

The reason is that some browsers (notably FireFox) are strict on so–called JavaScript cross–domain security, meaning it won't run JavaScript that is not coming from the same server as the html. And from the browser viewpoint, a file coming from the file system is from another domain than one from http://localhost/, even if both addresses point to the same file...!

Showing the GetFeatureInfo results in alertboxes, like we do here, is a simple, but not a very good solution. The way it works and looks will be different on various web browsers and operating systems. Let's make the results appear in our map interface instead:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 12 : Edit your html file: Add a third element to the HTML layout as a placeholder for the query results:

```
<div id="queryresultsDiv" style="position: absolute; width:
250px; height: → 395px; left: 420px; top: 210px; overflow:
auto; border: 1px solid red;">
</div>
```

Now change the showQueryResults function to:

```
function showQueryResult(evt) →
{ queryresultsDiv.innerHTML = evt.text; }
```

Now the results should be shown nicely in the results box. You will notice that especially at small scales, many elements will be found. You can manipulate this by editing the TOLERANCE values in the config.map files. Try to get sensible values for the various LAYERs, and keep in mind their different types (polygon, line and point data)…!

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# I. WMS in a webmap page: Combining WMS and OpenStreetMap

*Version 3.9 - September 17, 2012*

I.1. OpenLayers: adding layers

I.1.1. Combining Thailand WMS layers with an OpenStreetMap background

*Key points*

In this exercise we will us it to create a web page that gives you access to OpenStreetMap, an Open Data map service, and overlay that with your own Web Map Services. You will learn how to:

1. Use the OpenLayers library in a web page;
2. Make a simple map using the OpenStreetMap background;
3. Add your own WMS data to it.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### I.1. OpenLayers: Adding layers

Open Layers allows the use of many layers in the same client, coming from different sources:

- OGC Web Map Services (WMS) & Web Feature Services (WFS);
- Other open geo-webservices (e.g., OpenStreetMap, GeoRSS, NASA World-Wind);
- Commercial geo-webservices like Google Maps, Yahoo, Microsoft Bing Maps, MultiMap;
- Client-side data such as GML, KML, marker-layers, drawing layers, text layers and others.

You define layers using the OpenLayers.Layer object:

```
myNewLayer = OpenLayers.Layer.layerType( nameOfLayer, →
{listOfOptions} );
```

The `layerType` defines what server– or client–side layer is to be used. `{listOfOptions}` is an array of name:value pairs that are used to add a multitude of special settings. Depending on the `layerType`, zero or more options are required. We saw for example in the WMS `layerType` we used in section 2 that a few additional options were needed. E.g. the `baseURL`, that determines where to find the WMS, and `layers`, a string that will be usedt o ask one or more layers from the WMS.

There is an important difference between what OpenLayers calls Base Layers and Overlays. There is always at least one base layer needed. It's the one that determines the projection, extent and units of the map. You can define several base layers if they share the same parameters (e.g., several layers of one WMS, or the various types of Google maps), but the user can always have only one turned on, the choice will be made using a radio-button list. Overlays are layers that can be fitted on top of the base layer. Because they can be transparent, many overlays can be used, and they each can individually be turned on or off (using a checkbox list). By default, new raster layers will become base layers, unless you have made sure they are transparent, in that case they can be overlays. Most vector layers are by default Overlays.

#### *I.1.1. Combining Thailand WMS layers with an OpenStreetMap background*

Now let us try to use the OpenStreetMap as a background layer for the Thailand WMS we made earlier in the MapServer exercises:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 1: First find the line that reads:

```
var myMap, myWMSBaseLayer, forestLayer, railroadLayer,
airportsLayer ;
```

and change it to:

```
var myMap, myOSMLayer, forestLayer, railroadLayer,
airportsLayer ;
```

Next, find the lines that read:

```
myMap.addLayers([myWMSBaseLayer, forestLayer, railroadLayer,
airportsLayer]);
```

To change to the OSM as a base layer, change it to:

```
myMap.addLayers([myOSMLayer, forestLayer, railroadLayer,
airportsLayer ]);
```

Now look for the line in the code that reads:

```
myWMSBaseLayer = new OpenLayers.Layer.WMS
( <...more code here...> );
```

and change it to:

```
myOSMLayer = new OpenLayers.Layer.OSM("OpenStreetMap");
```

Try it in the browser.

Note that the result is not what you might have expected: The map is no longer zoomed in to the correct location! But even if you zoom and pan back to Thailand, no Thailand WMS layers are present. Instead you see so-called “pink tiles”, which are OpenLayer’s way of indicating it has trouble with one or more layers...!

Your initial try to combine OSM and WMS failed, because it is not so straightforward as it may seem. The reason for that is the difference in map projection of the layers. The basic OSM layer is using the Google Mercator projection (EPSG code 3857 or 900913), as explained in the exercise where you loaded the OSM separately. But the Thailand WMS data is in degrees longitude–latitude on the WGS84 datum (EPSG code 4326). We can not influence the OSM projection, but a proper WMS should be able to serve its data in any projection that the software supports. However, we do have to set up the WMS to allow that, which is very simple actually:

Task 2: In the WMS configuration file (config.map) look for the WEB/METADATA object and find the line

```
"ows_srs" "EPSG:4326 " #latlon
```

and add the Google Mercator to the list of allowed projections, thus:

```
"ows_srs" "EPSG:4326 EPSG:3857" #latlon,google
```

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Next we have to make sure OpenLayers makes the proper request for the re–projection to happen:

Task 3: In the html file change the simple map object definition:

```
myMap = new OpenLayers.Map("mapDiv");
```

to the more detailed:

```
myMap = new OpenLayers.Map("mapDiv", {projection:"EPSG:3857"});
```

What we do here is tell OpenLayers explicitly to use the Google Mercator projection. The OSM layer "knew" about this already, but now also the WMS layer will be told to behave!

Now try loading the map once again in the browser. You now should have more success.

The only small problem left is that the map initially starts zoomed into the wrong place. This is because the coordinates we provided for the center of Thailand were in degrees longitude-latitude. And now that we use the Google-Mercator projection, we need to provide it in meter X-Y coordinates:

Task 4: Change the `myMap.getCenter` coordinates and the zoom factor to have the map initially zoomed in on Thailand. Note that the function used to set the coordinates should remain the same, although its name is now a bit confusing and contra-dictionary.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Markus Jobst

# J. Introduction to SQL, the Structured Query Language

*Version 1.0 – October 10th, 2012*



*Key points*

This document is intended to serve as a quick introduction to the Structured Query Language (SQL). This language is the central tool to access database content and extract information that is stored in tables and relations most efficiently. Databases are one main component in spatial data infrastructures and feed material for map production (beside the direct access to data via geo web services).

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## J.1. A SQL explanation

SQL, which is an abbreviation for Structured Query Language, is a language to request data from a database, to add, update, or remove data within a database, or to manipulate the metadata of the database.

Instructions are given in the form of statements, consisting of a specific SQL statement and additional parameters and operands that apply to that statement. SQL statements and their modifiers are based upon official SQL standards and certain extensions to that each database provider implements. Commonly used statements are grouped into the following categories:

Data Query Language (DQL)

- SELECT - Used to retrieve certain records from one or more tables.

Data Manipulation Language (DML)

- INSERT - Used to create a record.
- UPDATE - Used to change certain records.
- DELETE - Used to delete certain records.

Data Definition Language (DDL)

- CREATE - Used to create a new table, a view of a table, or other object in database.
- ALTER - Used to modify an existing database object, such as a table.
- DROP - Used to delete an entire table, a view of a table or other object in the database.

Data Control Language (DCL)

- GRANT - Used to give a privilege to someone.
- REVOKE - Used to take back privileges granted to someone.

## J.2. Dealing with database tables

Before learning SQL, relational databases have several concepts that are important to learn first. Databases store the data of an information system. We regroup data by groups of comparable data (all the employees, all the projects, all the offices...). For each group of comparable data, we create a table. This table is specially designed to suit this type of data (its attributes). For instance, a table named “employee” which stores all the employees would be designed like the following:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



| employee (the table) | |
|---|---|
| id_employee (the primary key) | an integer |
| firstname (a column) | a string of characters (a column type) |
| lastname | a string of characters |
| phone | 10 numbers |
| mail | a string of characters |

The storage will look like the following figure.

| employee | | | | |
|---|---|---|---|---|
| id_employee (a column value) | firstname | lastname | phone | mail |
| 1 | Big | BOSS | 936854270 | big.boss@company.com |
| 2 | John | DOE | 936854271 | john.doe@company.com |
| 3 | Linus | TORVALDS | 936854272 | linus.torvalds@company.com |
| 4 | Jimmy | WALES | 936854273 | jimmy.wales@company.com |
| 5 | Larry | PAGE | 936854274 | larry.page@company.com |

The data stored in a table is called entities. As a table is usually represented as an array, the data attributes (first name, last name...) are called columns and the records (the employees) are called rows. `id_employee` is a database specific technical identifier called a primary key. It is used to link the entities from a table to another. To do so, it must be unique for each row. A primary key is usually underlined. Any unique attribute (for instance, the mail) or group of attributes (for instance, the first name and last name) can be the table primary key but it is recommended to use an additional technical id (id_employee) for primary key.

| employee | | | | |
|---|---|---|---|---|
| id_employee | firstname | lastname | phone | mail |
| 1 | Big | BOSS | 936854270 | big.boss@company.com |
| 2 | John | DOE | 936854271 | john.doe@company.com |
| 3 | Linus | TORVALDS | 936854272 | linus.torvalds@company.com |
| 4 | Jimmy | WALES | 936854273 | jimmy.wales@company.com |
| 5 | Larry | PAGE | 936854274 | larry.page@company.com |

| project | | | | |
|---|---|---|---|---|
| id_project | name | created_on | ended_on | # manager |
| 1 | Google | 1998-09-08 | NULL | 5 |
| 2 | Linux | 1991-01-01 | NULL | 3 |
| 3 | Wikipedia | 2001-01-01 | NULL | 4 |

If we add another table called “project”, we can explain the use of a foreign key. `id_project` is the primary key of the project table and `manager` is a foreign key. A foreign key is a technical id which is equal to one of the primary keys stored in another table (here, the employee table). Doing this, the Google project is linked to the employee Larry PAGE. This link is called a relationship. A foreign key is usually preceded by a sharp (#). Note that several projects can point to a same manager so an employee can be the manager of several projects.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



If we want to create, not a single link, but multiple links, we have to create a junction table. A junction table is a table that isn't used to store data but links the entities of other tables. The following figure shows a table called `members` which links employees to project.

| employee | |
|---|---|
| id_employee | an integer |
| firstname | a string of characters |
| lastname | a string of characters |
| phone | 10 numbers |
| mail | a string of characters |

| members | |
|---|---|
| # id_employee | an integer |
| # id_project | an integer |

| project | |
|---|---|
| id_project | an integer |
| name | a string of characters |
| created_on | a date |
| ended_on | a date |
| # manager | an integer |

An employee can be associated to several projects (John DOE with Google and Wikipedia) and a project can be associated to several employees (Wikipedia with Jimmy, John and Jenny), which is impossible with just a foreign key. A junction table hasn't its own primary key. Its primary key is the couple of foreign keys, as this couple is unique. A junction table can link more than two entity tables by containing more columns.

### J.3. Relationships

Several various relationships can be defined within a relational database. So let's list the different types of relationships:

- One to one,
- One to many (for instance, the manager of a project),
- Many to many (for instance, the members of the projects).

For each type of relationships, there is a way to link the entities:

- One to many relationship: create a foreign key from an entity table to the other,
- Many to many relationship: create a junction table,
- One to one relationship: just merge the two tables.

Now you know how to design a database schema and to put the data of your information system into it. The Data Query Language is used to extract data from the database. It doesn't modify any data in the database. It describes only one query: SELECT.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## J.4. The SELECT statement

The following figure shows an exhaustive overview on the syntax of the SELECT query.

```
SELECT [ ALL | DISTINCT ] <COLUMN name> [ AS <alias> ] [, [ ALL | DISTINCT ] <COLUMN name> [ AS <alias> ] ]*
FROM <table> [ AS <alias> ] [ [ FULL | LEFT | RIGHT ] [ OUTER ] | [ INNER ] JOIN <table> ON <expression> ]
  [, <table> [ AS <alias> ] [ [ FULL | LEFT | RIGHT ] [ OUTER ] | [ INNER ] JOIN <table> ON <expression> ] ]*

[ WHERE <predicate> [ [ AND | OR ] <predicate> ]* ]
[ GROUP BY <COLUMN name> [, <COLUMN name> ]* ]
 [ HAVING <predicate> [ [ AND | OR ] <predicate> ] ]*
[ ORDER BY <COLUMN name> [ ASC | DESC ] [, <COLUMN name> [ ASC | DESC ] ]* ]
 [ FETCH FIRST <count> ROWS ONLY ] ;
```

In order to explain the commands, following figures show the given table “reunion”. The left figure shows the table structure, the right figure shows the table with some test data.


### *J.4.1. A first query: a simple SELECT statement*

The following command in SQL will derive all content from the table “reunion”:

```
SELECT *
FROM reunion;
```

The result looks like this:

```
|-----------|------------|-----------------------------------------|--------|-------|----------|--------|--------|----------|----------------|
|id_reunion |name        |description                              |priority|planned|date      |hour    |duration|id_office |pdf_report      |
|-----------|------------|-----------------------------------------|--------|-------|----------|--------|--------|----------|----------------|
|1          |Planning    |We need to plan the project.             |A       |1      |2008-03-24|10:30:00|60      |35        |48644...846348  |
|2          |Progress    |What we have done.                       |C       |1      |2008-05-12|14:00:00|30      |13        |9862...15676    |
|3          |Change      |What we need to change in the project.   |B       |1      |2008-06-03|9:30:00 |90      |41        |34876...4846548 |
|4          |Presentation|Presentation of the project.             |D       |0      |2008-09-11|15:30:00|120     |27        |NULL            |
|5          |Reporting   |Explanation to the new beginner.         |B       |1      |2009-03-15|14:00:00|60      |7         |19739...37718   |
|6          |Learning    |A new software version has been installed.|B       |1      |2009-09-21|16:00:00|120     |11        |785278...37528  |
|-----------|------------|-----------------------------------------|--------|-------|----------|--------|--------|----------|----------------|
```

The graphical form of the result depends on the client application. It can be returned as a text output (backend), a HTML page (thin client) etc... The statements, queries, clauses (SELECT, FROM...), instructions and operators are not case sensitive but they are commonly written in uppercase for readability.

The SELECT and FROM clauses are the two required clauses of a SELECT query:

- FROM : list the tables the query uses to return the data.
- SELECT : list the data to return.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### *J.4.2. WHERE clause*

The WHERE clause doesn't influence the columns that are returned by the query. But it influences the rows. It filters the rows applying specific “predicates” on it. A “predicate” specifies conditions that can be true or false. SQL can handle conditions whose result is unknown.

The following query returns the reunions which have a B priority level:

```
SELECT *
FROM reunion
WHERE reunion.priority = 'B';
```

| id_reunion | name | description | priority | planned | date | hour | duration | id_office | pdf_report |
|---|---|---|---|---|---|---|---|---|---|
| 3 | Change | What we need to change in the project. | B | 1 | 2008-06-03 | 9:30:00 | 90 | 41 | 34876...4846548 |
| 5 | Reporting | Explanation to the new beginner. | B | 1 | 2009-03-15 | 14:00:00 | 60 | 7 | 19739...37718 |
| 6 | Learning | A new software version has been installed. | B | 1 | 2009-09-21 | 16:00:00 | 120 | 11 | 785278...37528 |

### *J.4.3. Predicates*

Compared to the second operand, the first operand can be :

- equal : =
- different : <>
- lesser : <
- lesser or equal : <=
- greater : >
- greater or equal : >=

The following query returns the reunions which have another priority level than B:

```
SELECT *
FROM reunion
WHERE priority <> 'B';
```

| id_reunion | name | description | priority | planned | date | hour | duration | id_office | pdf_report |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Planning | We need to plan the project. | A | 1 | 2008-03-24 | 10:30:00 | 60 | 35 | 48644...846348 |
| 2 | Progress | What we have done. | C | 1 | 2008-05-12 | 14:00:00 | 30 | 13 | 9862...15676 |
| 4 | Presentation | Presentation of the project. | D | 0 | 2008-09-11 | 15:30:00 | 120 | 27 | NULL |

### *J.4.4. Operators*

The WHERE clause can have several conditions using the operators AND (all the conditions must be true) and OR (only one condition needs to be true). The operator OR is inclusive (several conditions can be true). The order of evaluation can be indicated with brackets. NOT inverts a condition. The following query returns the reunions which have a B priority level and last more than an hour or which take place on 2008/05/12:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



```
SELECT *
FROM reunion
WHERE(priority = 'B' AND NOT duration <= 60) OR DATE = '2008-05-12';
```

| id_reunion | name | description | priority | planned | date | hour | duration | id_office | pdf_report |
|---|---|---|---|---|---|---|---|---|---|
| 2 | Progress | What we have done. | C | 1 | 2008-05-12 | 14:00:00 | 30 | 13 | 9862...15676 |
| 3 | Change | What we need to change in the project. | B | 1 | 2008-06-03 | 9:30:00 | 90 | 41 | 34876...4846548 |
| 6 | Learning | A new software version has been installed. | B | 1 | 2009-09-21 | 16:00:00 | 120 | 11 | 785278...37528 |

### *J.4.5. The FROM clause*

The FROM clause defines the tables that are used for the query but it can also join tables. A JOIN builds a *super* table with the columns of two tables to be used for the query. To explain what a join is, we consider two archaic tables without primary keys nor foreign keys.

We want to associate values from columns of different tables matching values on a given column in each table. Then we use in most cases “JOIN”.

### *J.4.6. The FULL OUTER JOIN*

A JOIN is made matching a column on a table to a column on the other table. After a FULL OUTER JOIN, for a given value (red), for a given row with this value on one table ([ red | 9999 ]), one row is created for each row that matches on the other table ([ red | OOOOOO ] and [ red | LLLLLL ]). If a value exists in only one table, then a row is created and is completed with NULL columns.

```
FROM table_1 FULL OUTER JOIN table_2 ON table_1.common_value =
table_2.common_value
```

| common_value | specific_value_1 | specific_value_2 |
|---|---|---|
| red | 9999 | OOOOOO |
| red | 9999 | LLLLLL |
| grey | 6666 | **NULL** |
| white | 0000 | **NULL** |
| purple | 7777 | **NULL** |
| purple | 2222 | **NULL** |
| black | 8888 | FFFFFF |
| green | **NULL** | HHHHHH |
| yellow | **NULL** | PPPPPP |
| blue | **NULL** | RRRRRR |

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## J.5. The INSERT statement

The following figure shows an exhaustive overview on the syntax of the INSERT statement.


The INSERT statement is used to add new records (rows) in a table. For instance, we want to add a new reunion:

- Its primary key is 7,
- Its name is "Job interview",
- Its description is "Meeting with Mr. SPENCER",
- Its priority is B,
- Its planned,
- Its date is on October 28, 2009,
- Its hour is 18:30:00,
- Its duration is 30,
- Its office technical id is 23,
- There is no pdf report.

The statement uses the following statement:

```
INSERT INTO reunion (id_reunion, name, description, priority,
planned, DATE, HOUR, duration, id_office, pdf_report)
VALUES (7, 'Job interview', 'Meeting with Mr. SPENCER', B, 1,
2009-10-28, 18:30:00, 30, 23, NULL);
```

reunion

| id_reunion | name | description | priority | planned | date | hour | duration | id_office | pdf_report |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Planning | We need to plan the project. | A | 1 | 2008-03-24 10:30:00 | 60 | 35 | | 48644...846348 |
| 2 | Progress | What we have done. | C | 1 | 2008-05-12 14:00:00 | 30 | 13 | | 9662...15676 |
| 3 | Change | What we need to change in the project. | B | 1 | 2008-06-03 9:30:00 | 90 | 41 | | 34976...4846548 |
| 4 | Presentation | Presentation of the project. | D | 0 | 2008-09-11 15:30:00 | 120 | 27 | | |
| 5 | Reporting | Explanations to the new beginner. | B | 1 | 2009-03-15 14:00:00 | 60 | 7 | | 19739...37718 |
| 6 | Learning | A new software version has been installed. | B | 1 | 2009-09-21 16:00:00 | 120 | 11 | | 785278...37526 |
| 7 | Job interview | Meeting with Mr. SPENCER | B | 1 | 2009-10-28 18:30:00 | 30 | 23 | | |

The INTO clause contains the name of the table where the record needs to be inserted. It can be followed by a list of columns in brackets. The VALUES

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



clause contains the values to insert in brackets. If the column names are omitted, the VALUES clause must contains as many values as the number of columns of the table. The values are inserted in the table columns in the same order that the order in which the columns have been declared in the table. If the column names are mentioned, there must be as many column names as values. The values are respectively inserted into the named columns. If a column in the table is omitted, a NULL value is inserted instead.

### J.6. The DELETE statement

The following figure shows an exhaustive overview on the syntax of the DELETE statement.

```
DELETE FROM <table name>
[WHERE <predicate> [{AND | OR} <predicate>]*];
```

The DELETE statement is used to remove specific rows in a table with conditions. The FROM clause is followed by the table name in which the rows need to be removed. The WHERE clause contains predicates. If the predicates are true for an row, this row will be removed. If the predicates are false for all the rows, the statement do nothing. A DELETE statement without WHERE clause empties the table.

For example, we want to remove all the reunions that last two hours:

```
DELETE FROM reunion
WHERE duration = 120;
```

### Further reading

This lecture bases on the wikibook SQL (http://en.wikibooks.org/wiki/Structured_Query_Language, [accessed in October 10th 2012]), which gives a comprehensive and easy readable overview on this topic.

Melton, Jim; Alan R Simon (1993). "1.2. What is SQL?". Understanding the New SQL: A Complete Guide. Morgan Kaufmann. p. 536. ISBN 1-55860-245-3.

Wagner, Michael (2010). SQL/XML:2006 - Evaluierung der Standardkonformität ausgewählter Datenbanksysteme. Diplomica Verlag. p. 100. ISBN 3-8366-9609-6.

ISO/IEC 9075-11:2008: Information and Definition Schemas (SQL/Schemata). 2008. p. 1.

DateC. J. with Hugh Darwen: A Guide to the SQL standard : a users guide to the standard database language SQL, 4th ed., Addison Wesley, USA 1997, ISBN 978-0-201-96426-4

SQL in Wikipedia, http://en.wikipedia.org/wiki/SQL [accessed in October 10th 2012]

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 1: Use SQL statements on a geographical database. Spatiallite offers a graphical user-interface that allows to create and access a database easily.

The tool “spatialite GUI” can be found on the OSGeo Live system in “databases”.

Following steps give you a first SQL experience:

- Open “spatialite GUI”.
- Create a new database.
- Use the operation “load SHP” to import a table from a *.shp file. (In our case, we have downloaded a dataset called “world_factbk”.)
- Start your first SQL queries in the query window (e.g. like the following).

Request the whole content of the table “world_factbk”:

```
SELECT * FROM world_factbk
```

Request content that is from continent “Europe”:

```
SELECT * FROM world_factbk
WHERE CONTINENT = 'Europe'
```

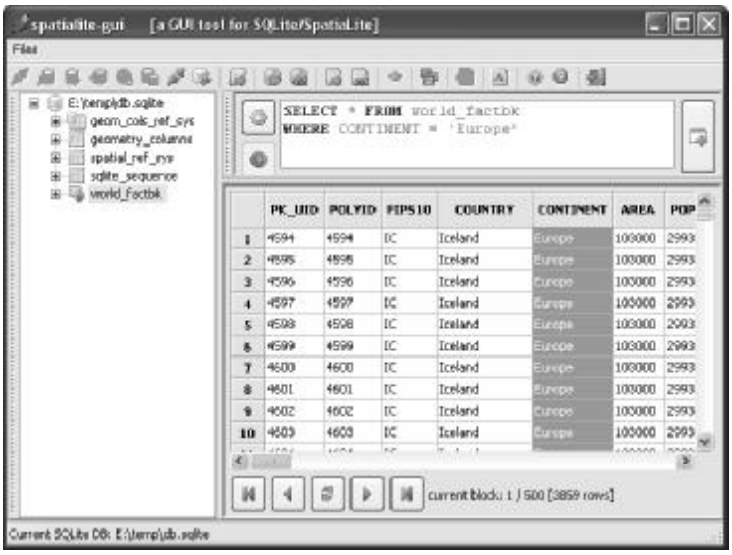


*Fig. J.1: Result of task1. A SELECT statement with WHERE clause.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# K. PostGreSQL/PostGIS Satial Databases: Using PostGIS with pgAdmin and QGIS

*Version 2.0 - September 18, 2012*



*Key points*

This is a short introduction to the database system Post-GreSQL and its spatial extension PostGIS. We will show you the basics of using pgAdmin to set up and administer PostGIS data, and how to use the desktop client QGIS to show spatial data in the form of maps as well as how to import data into those databases. This exercise is NOT an introduction to SQL or database theory.

This exercise also assumes you are working with the OSGEO LIVE system.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



## K.1. Introduction to PostGreSQL and PostGIS

In many information systems that use more then a trivial amount of data, a spatial database backend is required, to handle large amounts of data having to be served to large amounts of users. If the data we are using is spatial in nature, the PostGIS extension to the object–relational DBMS PostgreSQL is a good choice.

First, because PostgreSQL is a solid DBMS that has a reasonably gentle learning curve, yet is wonderfully appropriate for advanced database applications, and its documentation is very transparent. PostGIS in addition, is the leading open standards implementation of spatial vector management (and since this year also for raster data). It enjoys a lively and supportive user/developer community. The combination of the two is enjoying an ever-growing user base, particularly in the domain of industrial applications.

PostgreSQL(http://postgresql.org) is an object-relational database management system (ORDBMS) that is open-source and supports a large part of SQL. Among other things it enables complex queries, foreign keys, triggers, views, transactional integrity and has robust multiversion concurrency control. And because of the liberal license, PostgreSQL can be used, modified, and distributed by everyone free of charge for any purpose, be it private, commercial, or academic.

Due to a plug-in architecture it can be extended in many ways: by adding new datatypes, functions, operators, aggregate functions, index methods and procedural languages. This has been used by developers to create PostGIS: an extension that allows OGC Simple features objects to be stored in the database. PostGIS (http://postgis.refractions.net) includes support for GiST-based R-Tree spatial indexes, functions for analysis of OGC geometries and functions for processing of OGC geometries.

## K.2. Using pgAdmin to administer PostGreSQL/PostGIS

For communication with the PostgreSQL/PostGIS system, you could use any universal SQL client, but it is easiest to use the specific PostgreSQL client called pgAdmin III:

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 1: Start the application "pgAdmin III" : there should be a shortcut to it in the menu `Geospatial > Databases > pgAdminIII`.

In the list of `Servers`, there should be one (localhost) already defined to connect to. You can double-click that to open a connection to the PostgreSQL server.

[If not present yet: Choose File > Add Server and fill in the dialog: the address should be localhost, the port 5432 and you should specify the user name "user" and password "user".]

Now you can browse the databases and look at the tables, functions and other objects they hold. There are already some databases that you can look at. But what makes a database into a spatial database? For that the spatial function and types of the PostGIS extension should be loaded.

This is best demonstrated when we create a spatial database:

Task 2: Create a new database by right-clicking the Databases icon and choosing `New database...`

Name the DB `test` with `UTF-8` as encoding and `template_postgis` as template: Using the template_postgis database ensures that the PostGIS spatial datatypes and functions are loaded.

Open the database you just created, it should have one schema (public), with two tables: 'spatial_ref_sys' and 'geometry_columns'. You can look at the contents of any table by right-clicking them and selecting

`View Data` or `View data (top 100 rows)`.

Table spatial_ref_sys: This is the special PostGIS table that holds the EPSG definitions of the spatial reference systems. Users can only read this table, and the internal PostGIS functions use it for (re)projections and geometric calculations.

Table Geometry_columns: This is the special PostGIS table that will store the metadata about the spatial data in the database: it holds information about which tables hold geometries, in which column the geometry is stored, what type that geometries is (polygon, point, line, etc.) and which spatial reference systems they use. At the moment it is still empty, because we have no spatial data in the database yet.

We can populate a database using the SQL commands in pgAdmin or running command-line programs such as `shp2pgsql`, but we will use the QGIS desktop GIS we used earlier to do that in the next section.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### *K.2.1. Using SQL for communication with the database*

You can execute queries on the DB using the SQL window:

Task 3: Open the database "natural_earth". Now start the SQL query tool by clicking the SQL icon in the toolbar or choosing the menu `Tools > Query`. In the SQL editor pane, type the query:

```
SELECT * FROM "10m_admin_0_countries"
```

Click the run (little triangle) icon to execute the SQL, or choose the menu `Query> Execute`.

You'll see in the output that the column `the_geom` of the table world is shown as a long row of (hexadecimal) characters. This is because these are the geometric data, stored in an internal binary format, which cannot be read by humans straight-away. But you can use the PostGIS SQL function `AsText(the_geom)` to show it in OGC WKT, which stands for "Well Know Text", a human-readable form of the OGC geometry:

Task 4: Try to find out how to get the geometry shown as OGC Well Know Text in the output window. You should use the PostGIS function called

```
AsText(the_geom)
```

## K.3. Using PostGIS data in QGIS

In the earlier lesson where QuantumGIS, or QGIS was introduced, we already explained that this Open Source desktop GIS can connect to spatially-enabled PostgreSQL/PostGIS tables, by means of a "live" connection to the database server.

### *K.3.1. Showing PostGIS data in QGIS layers*

PostGIS data can be loaded into map layers. Below we will show you how to load vector data from one of the available sets of test data in the OSGEO Live system: Natural Earth.

Natural Earth is a dataset that provides cartographers an off-the shelf solution for creating small-scale world, regional, and country maps at 1:10-, 1:50-, and 1:110millionscales. Both political (administrative) and physical (natural) features are included and vector features align perfectly with included raster data.

Natural Earth solves the common problem that many cartographers face: finding vector data for publication-quality small-scale maps at the appropriate level of detail for the maps they are making. Its core features are that vector features include name and other attributes, with built-in scale attributes to direct features to be shown at different zoom levels; Large polygons are split for more efficient

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



data handling; Lines contain enough data points for smooth bending in conic projections, but not so many that computer processing speed suffers; Raster data includes grayscale-shaded relief and cross-blended hypsometric tints.

Natural earth data is public domain, unlicensed and free to use for everyone. Its available on the website http://naturalearthdata.com

Task 5: Start QGIS. Click the "Add PostGIS Layers" icon or use the `Layer > Add PostGIS Layers...` menu. From the pull-down menu `Connection`, choose the "NaturalEarth" connection.

A list with available layers will appear. Click the `Edit` button to see details of the connection. Notice that QGIS will try to connect to a database running on the `localhost` host. Cancel the connection editor.

The PostGIS connection can be to any PostGreSQL server running on any host, providing that QGIS can connect to this server over the internet or an intranet, through the PostGreSQL database port (normally 5432). In this case the server is running on your own computer within the OSGEO Live environment.

QGIS will only show tables that have spatial content, i.e. that have at least one column of spatial data. We can now load any of these available tables:

Task 6: Select the first layer "10m_admin_0_countries" and click the add button. The map layer will be drawn. •

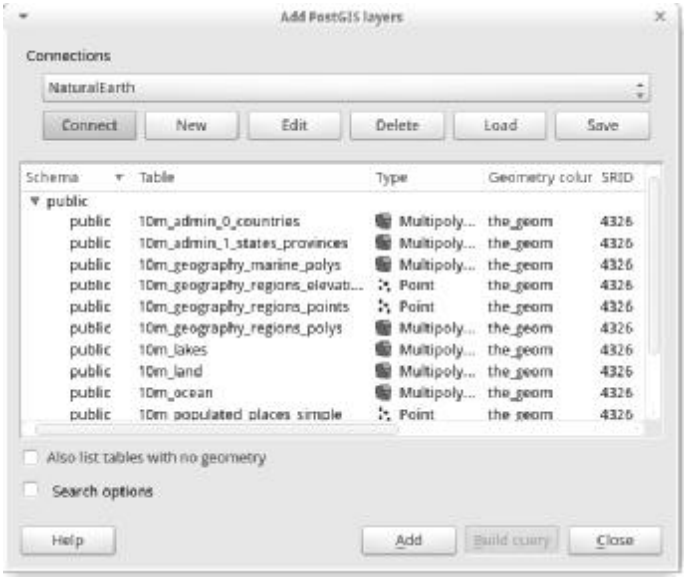


*Fig. K.1: PostGIS connections in QGIS.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Initially all data from the table will be loaded and drawn. Essentially the QGIS connection is used to do a `SELECT * FROM "10m_admin_0_countries"` SQL query. We can however change this behaviour:

Task 7: Select the layer in the Layers Panel and choose the menu `Layer > Query...` (or right-click the layer name and choose `Query`). A Query Builder window will appear.

Try to make the layer show only those countries of which the name starts with the letter "N". Note that in SQL, text comparisons are done using the "LIKE" keyword, and that the SQL wildcard character is the percentage sign %.

### *K.3.2. Loading data into PostGIS using QGIS*

We can also use QGIS to load data into PostGIS tables. Any vector data we have available as a QGIS layer can be saved as a Shapefile, and there is a Shapefile to PostGIS Import Tool (SPIT) plugin. You will first have to activate this plugin:

Task 8: Choose the `Plugins > Manage Plugins...` menu. Select the plugin named "SPIT" and press `OK`. A new menu item `Database > Spit` will be available.

Now you can use the plugin to load one or more shape files into a database.


*Fig. K.2: QGIS Extension Manager.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



Task 9: Choose the `Database > Spit` menu. The SPIT dialog will appear.

First you have to specify the connection to the database. In most cases you might want to first create a new database yourself (using a DB administration software such as pgAdmin), but for now we will use the existing `NaturalEarth` connection again: Select it and click `Connect`.

Now use the `Add` button to browse for a shape file. You can use the data you created in the earlier QGIS exercise (where you digitized a route on a OpenStreetMap background), or try to find another useful shape-file in the OSGEO Live system or on your own hard disk.

Now click the `OK` button, and the plugin will load the data into the appropriate database tables. You can test this by trying to load the new table as a map layer.

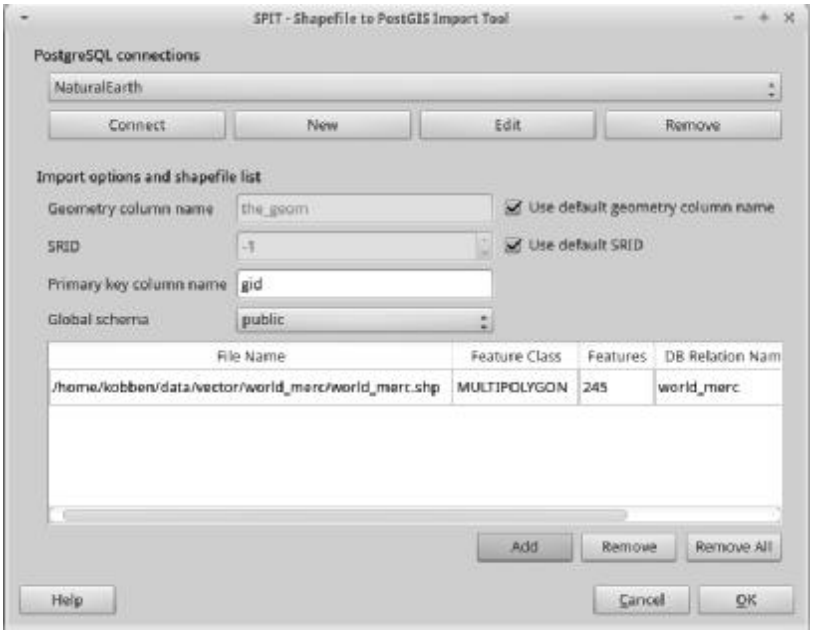


*Fig. K.3: SPIT, Shape file to PostGIS import tool.*

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.

Barend Köbben

# L. PostGreSQL/PostGIS Spatial Databases: Using PostGIS Data in a MapServer WMS

*Version 2.3 - September 18, 2012*



*Key points*

In this small exercise you will use PostGIS data in Mapserver. It is not a step–by–step exercise, but only a quick overview of some of the necessary setups. This exercise assumes you have already done the exercise “Serving data as a Web Map Services using MapServer”, as well as PostgreSQL/PostGIS basics!

This exercise also assumes you are working with the OSGEO LIVE system.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



### L.1. Principles of using postGIS in MapServer

When using shape–files, connecting a LAYER to a datasource was a simple matter of specifying the DATA parameter as a path to a file. For connections to a PostGIS datasource, you have to use the more elaborate CONNECTION parameter with a CONNECTIONTYPE postgis. The general form of the connection parameters is:

```
CONNECTIONTYPE postgis
CONNECTION "user=<username> password=<password> dbname=<database>
→ host=<dbhost> port=<port> options='-c client_encoding=
→<encoding>'"
DATA "<geometry column> FROM <table> USING unique <PK>
→USING srid=<srid>"
```

In the CONNECTION parameter, you fill in the correct username, password, etcetera. The <encoding> should match the encoding of the PostGIS database, normally this would be UTF8. In the DATA parameter, you specify a special type of SQL statement that will be sent to the PostGIS database. The <PK> should be the name of a column in your table that is unique for all rows, usually the Primary Key column.

For example: You want to get the countries of the world geometry out of the database of Natural Earth data pre–installed in PostGIS on the OSGEO LIVE system (in the table 10m_admin_0_countries of the database natural_earth on the localhost database server). Then you should type:

```
CONNECTIONTYPE postgis
CONNECTION "user=user password=user dbname=natural_earth
→ host=localhost port=5432 options='-c client_encoding=UTF8'"
DATA "the_geom FROM 10m_admin_0_countries
→ USING UNIQUE gid USING srid=4326"
```

The nice thing is, that the DATA string is basically an SQL select string; therefore, you can change it to any valid SQL statement that makes a more complex query. An example of retrieving only countries with a name starting with A would be:

```
DATA "the_geom FROM (SELECT * FROM 10m_admin_0_countries WHERE
→ cntry_name LIKE 'A%') AS foo USING UNIQUE gid USING srid=4326"
```

Note that the MapServer SQL parser is a bit peculiar: It needs its query always to result in only a geometry column: this is achieved by making a sub–query with an alias (hence the AS foo statement).

### L.2. Further Layer definitions

For correct use of PostGIS layers in many of the GIS clients, MapServer needs a full specification of the LAYERs extent in the metadata of the layer. In order to do that, in the METADATA object in the LAYER, add the line `"ows_extent"`

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.



`"xmin ymin xmax ymax"` You can usually just copy the extent string from the general MAP extent. When using the world database countries data, for example, the relevant part of your LAYER would look like:

```
CONNECTIONTYPE postgis
CONNECTION "user=user password=user dbname=natural_earth
→ host=localhost port=5432 options='-c client_encoding=UTF8'"
DATA "the_geom FROM 10m_admin_0_countries
→ USING UNIQUE gid USING srid=4326"
METADATA
   "ows_title" "world"
   "ows_extent" "-180 -90 180 90"
END
```

Task 1: Now set up MapServer to show a map of the countries in the world from the world PostGIS database described above).

You'll have to set up an appropriate LAYER in a *.map file first. You can then show the layer using a GetMap request typed in a browser URL, or even nicer: request it in an OpenLayers html page...!

### L.3. A challenge: Trying a more complicated query

If you have time and at least a bit of SQL knowledge, you should try this more challenging query:

Task 2 : The challenge is to set up Mapserver to show a map of the countries in the world that only shows the countries where the distance to ITC's location (measured on the WGS84 ellipsoid) is less then 2000 km. •

The pieces you can use for this puzzle are:

- ITC's location in lon/lat is: `6.8862 , 52.2237` ;
- There is a PostGIS function `geometryFromText(OGC_WKT, srid)` to make a geometry from an OGC Well-Known Text string (WKTs are explained in the PostGIS manual and on the OGC standards webpages);
- there is another function `ST_Centroid(geometry)` that returns the centroid of any geometry, as a point;
- and yet another function `ST_Distance_sphere(point,point)` that calculates the distance in meters between two points, as a great circle (projected on the WGS84 datum in lon/lat).

If you need additional information, use the PostGIS manual.

Published in: M. Jobst (ed), Service-Oriented Mapping 2012, Section VII, chapters B-I & K-L, pages 531–592 & 603-613.
JOBSTMedia Verlag, Vienna (Austria), 2012. ISBN 978-3-9502039-2-3.