MapServer

An OpenSource Web-Map Applications

Tryggvi Már Ingvarsson
t.m.ingvarsson@student.tudelft.nl

LIST OF FIGURES

LIST OF TABLES

LIST OF TEXT BOXES

LIST OF ACRONYMS AND SPECIFICATIONS

- *Apache Server*: An OpenSource HTTP server, developed and maintained by the Apache Software Foundation. More information can be obtained at www.apache.org.
- *CGI*: The Common Gateway Interface is a standard for interfacing external applications with information servers, such as HTTP or Web servers.
- *GIS*: Geo Information Systems.
- *GML*: Geography Mark-up Language is a XML implemented according to standards set by OGC. The OGC specification can be found at http://www.opengis.org/docs/02-023r4.pdf.
- *GNU/LGPL*: General Public Licenses / Lesser General Public License. This is similar to OpenSource licence and used by software vendors like Geo-Looks.
- *HTTP*: HyperText Transfer Protocol, the underlying protocol used by the World Wide Web.
- *MapServer*: An OpenSource software to use in interactive web mapping applications.
- *MySQL*: An OpenSource a relational database server developed by MySQL AB. More information can be gained at www.mysql.org.
- *OGC*: Open GIS Consortium is a non-profit organisation that develops computing standards for geo-processing interoperability. Examples of this are e.g. the OpenGis specification for implementation of WMS and WFS.
- *OpenSource*: An OpenSource software is a software where the source code is open for modifications and distributions should be free of charge. More can be read at www.opensource.org.
- *SQL*: Standard Query Language for data base system.
- *TextPad*: Is a simple but powerful text editor that can be adjusted to MapServer editing (creating colours and tabs to make the editing more easier). The editor can be found at www.textpad.com.
- *UML*: Unified Model Language.
- *WFS*: Web Feature Service. An implementation specification made by OGC can be found on http://www.opengis.org/docs/02-058.pdf. The newest version is 1.0.0.
- *Wiki*: Wiki is a piece of server software that allows users to freely create and edit Web page content using any Web browser. Wiki supports hyperlinks and has a simple text syntax for creating new pages and crosslink's between internal pages on the fly.
- *WMS*: Web Map Service. An implementation specification made by OGC can be found on http://www.opengis.org/docs/01-068r2.pdf. The newest version is 1.1.1.
- *WWW*: World Wide Web.
- *XML*: eXtensible Mark-up Language is a mark-up language for documents containing structured information and the foundation for the development of GML.

INTRODUCTION AND CHRONICLE OF WORK

> On the internet there is more and more free software available to create web servers (Apache), store spatial data (MySQL) and to server spatial data on the web (MapServer). All this software is freely downloadable. Although it should be easy to build a web mapping server from this software's, there is not much experience with such a project. There is a lot of software's on the internet, but the status of the software is sometimes unclear and how easy it is to integrate the different packages is also unknown. (Description of assignment by Wilko Quak)

This assignment started from an idea that Wilko Quak presented to me in the autumn 2003. The aim was to make a research on availability of open source software's available on the internet and to make an interactive web map application to display data from different sources, like SHP, DXF and DGN files. Furthermore to check if this software's could also been used to create web based geo-information system (GIS) where it would be possible to make complex queries, by using structured query language (SQL).

The research work started in September 2003. It was little bit obstacle for the research that it was implemented prior to studies in related subjects at TU Delft. I therefore recommend that students should not implement "Individual Assignment" until the second year of master education. This obstacles where evident when I was handling subjects like OpenGis Consortium, Computer Protocols and how to make an independent research and proper scientific report. The last mentioned was probably the biggest obstacle because I had to create my own methodology as I have not made concrete engineer oriented (computer) research report before. The chronicle of my work looked like this:

September-October-December: I installed Apache, MapServer and MySQL on my computer at Jaffalaan 9 at TU Delft. The first two months were spent in studying manuals that I found on the internet about the software's and in collecting some spatial data to use in experiments. The data that I collected was at first some SHP files from Suriname that I used to explore the capabilities of MapServer. Later on I got DGN, SHP files and GeoTIFF pictures from Iceland, to make a little MapServer project of my own. This project was to check if I could use various data formats in one MapServer applications (hence to view DGN, SHP etc. file formats simultaneously in a web map explorer). This experiment was success except that I could not connect MySQL database effectively to my DGN files. My conclusion was that I would have to make several converts of data to make that possible and therefore I concluded that it was not practical to explore it further (as the advance of a specific software like MySQL is remote if one has to process time consuming data conversion to use it). I also found in references that current version of MySQL does not work well with MapServer as it will slow down the web-application and it even appears to be buggy, at least in previous releases of MapServer (MapServerWiki, 2004). However, under way is version MySQL 4.1 that enables the use of spatial feature according to the OpenGIS Consortium (OGC) specification (MySQL, 2004).

The reason why I explored MySQL capabilities with MapServer is that I thought it would enable GIS applications and complex queries. My finding was that MapServer is not suitable to work with complex SQL queries where it is necessary to create new composite data layer from existing data. It can handle simple queries that just concern one data layer but it is too complicated to make complex on the fly queries on diverse data resulting in new "query-result" layer. On the other hand there are movements in those directions on the MapServer mailing board. The only examples I found of

MapServer web applications that employed MySQL had no basic SQL query possibilities (Interesting site using MySQL with MapServer: http://www.nrm.qld.gov.au/lris/webgis/burnett/index.html).

December-January-February: I spent the latter stage of my project in exploring WMS and WFS capabilities of MapServer 4.0. To do that, I downloaded data of Canada that can be accessed free of charge through GeoConnections (see homepage in references) and more governmental departments of Canada. At this stage I also needed to set MapServer again up as I switched to my computer at home. This was little bit tricky where as I accidentally downloaded different version of MapServer 4.0 than I had been using earlier and I also had XP instead of Win 2000 on my computer.

When I finally had MapServer running on my XP by using Apache server, I started to test the compatibility of the software to WMS servers that can widely be found on the internet. To be able to connect to different WMS server I had to get knowledge of using XML, at least to be able to extract information like WMS parameters and sources to use in my MapServer application. A good documentation on the structure of a XML template for WMS as specified by OGC in version 1.1.0 can be found at http://www.digitalearth.gov/wmt/xml/capabilities_1_1_0.dtd. Likewise, I explored the capabilities of WFS but unfortunately I found no WFS service that fitted my experiment. Finally I did an experiment by making a web map application by using MapServer and other OpenSource software. The original idea was to make fully equipped web-GIS application by using OpenSource instruments but that aim developed into making a web applications that were based on diverse spatial data, both concerning to format (SHP, DGN etc.) and origin (local, WMS and WFS).

Next chapter defines the main objectives of the assignment that I carried out on OpenSource web-mapping applications. Followed is a chapter that describes the processes of installing essential software on the computer. Thereafter is the main chapter of this report, “MapServer” as it describes the architecture and functionality of the software in more detail, plus clarifying the WMS and WFS capabilities of MapServer. Finally, I answer the main objectives given below in a chapter called “Conclusions of Objectives”. In the end there are two appendixes. Appendix A is a “Checklist” that I made and lists most used objects and parameters employed in MapServer map file. Appendix B contains a figure that describes by the structure and relationships of MapServer objects.

OBJECTIVES

The objectivities of this assignment can be expressed into five segments:

1. To research the availability of OpenSource software on the internet to use in interactive web-mapping applications and that are compatibility with standard GIS applications like SQL.
2. To research the widely spread OpenSource software; MapServer and check its compatibility with OGC specifications and protocols (WMS and WFS), Apache server, MySQL, GML and applicability for practical uses in GIS applications.
3. To check the capability of MapServer to use different file formats, both for vector and raster. Emphasise were on vector file formats used in software from ESRI, AutoCad, Intergraph and Bentley.
4. To check the possibility to use OpenSource software for commercial uses.
5. To make a little web server project by utilising OpenSource software like MapServer, Apache and MySQL.

PREPARATIONS

Setting up an Apache HTTP Server

Apache HTTP server can be downloaded from "www.apache.org" and works on most platforms. I both installed it on Windows 2000 and Windows XP and it worked fine in both cases. It is most convenient if employing MapServer on Windows platform, to download the newest version of the server available (that was in my instance Apache 2.0.48) that is compatible with MSI Installer. The name of the file is "Win32 Binary (MSI Installer)" and can be found at: "http://httpd.apache.org/download.cgi".[1] As the file is compatible with MSI Installer the installation is more and less automatic as Windows guides you through setup wizard when the MSI file is executed.

Next step is to configure the server to your computer and to the WWW as the wizard asks for server and client names/URL. The variables that have to be completed are "Network Domain", "Server Name" and "Administrator's Email Address". As I use ADSL net service from Wanadoo I chose my IP address as the "Network Domain" and Wanadoo server IP as the "Server Name". This IP addresses can be found in "Startup > Control Panel > Network Connections > Your connection > Details" where "server IP address" is the server name and "client IP address" is the IP address of your computer (figure 1). By doing this you have made a homepage on the internet with your client IP address as URL (e.g. in my example http://81.69.54.149).

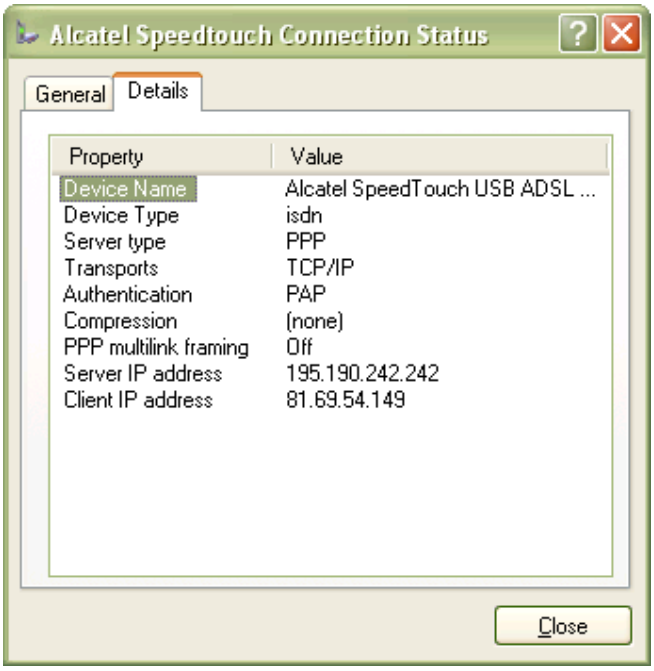

Figure 1 "Your Connection" menu.

Finally I chose "all users", "port 80", "typical setup" and the path of the server as "C:/Program Files/Apache Group/".

[1] Exact position of the file is: http://apache.proserve.nl/httpd/binaries/ win32/apache_2.0.48-win32-x86-no_ssl.msi

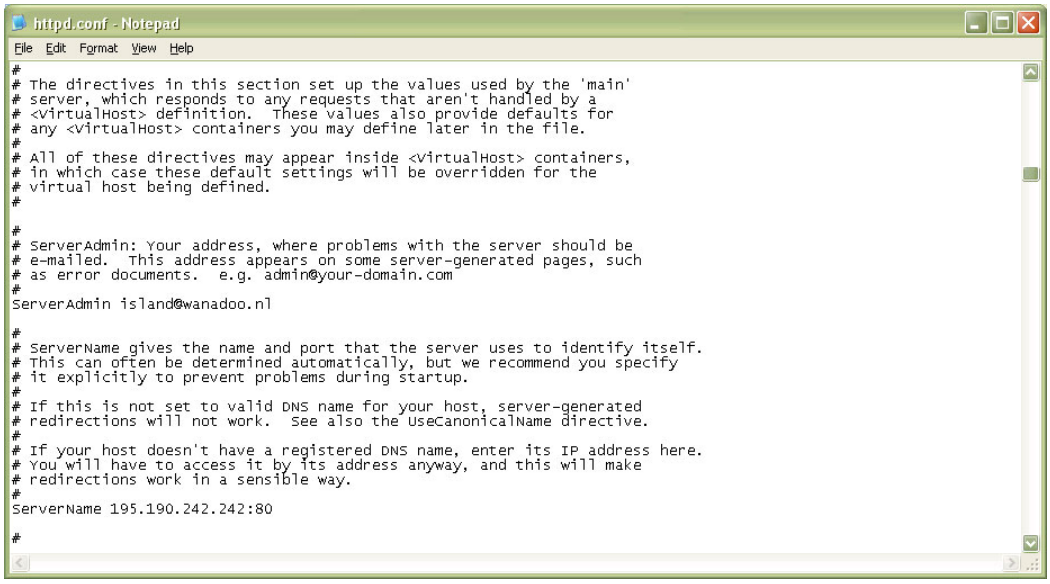

Figure 2 Configuration file for Apache Server

After the installation process is completed, the server is up and running. The sharing directory is then by default "C:/Program Files/Apache Group/Apache2/htdocs/" but that can be changed in the "configuration file" of the Apache Server. Later on it is important to monitor if your IP address changes somewhat as many internet service providers (e.g. Wanadoo, Tiscali, Zon, Cistron etc.) assign floating IP addresses to their client's computers to prohibit them to run commercial servers without paying commercial tariffs. This is fixed by changing the entry in the "configuration file" accordingly (figure 2).

Next step is to check if the server is running properly. After the installation "a feather with a play/stop-button upon" should appear at the bottom-right corner of your Windows interface. To test if it works right, i.e. is sharing and acting as a server, most simple is to execute your IP address as an URL in the web browser. You web browser should return the "index.html "file that is positioned in the "/htdocs/" folder.

It is also possible to run the server in a "so called loop" by not defining the server variables in the configuration file. That is practical when one wants to work on a server project without taking the risk of running an active web–server. The server is then running without sharing anything over the internet. To access the server through a web browser is then only possible on the server computer, by typing "localhost" as an URL in the web browser.

Further information on how to install Apache server on Windows platforms can be found at "http://httpd.apache.org/docs-2.0/platform/windows.html".

Installing MapServer

The installation file for Mapserver 4.0 for Windows platforms can be obtained at MapServer homepage "http://mapserver.gis.umn.edu/win32binaries.html". My choice was to download "ms40_png_pg.zip":

> Output support includes PDF, SWF, and all output formats supported by GDAL and GD (no GIF support); input support includes many OGR supported vector formats, GDAL raster files, PostGIS geospatial databases; also includes truetype fonts and on-the-fly projection support.

Another option is to download "ms40_gif_pg.zip" that is described as more comprehensive as it has GIF support. However I found out that MapServer developers are in favour to turn their attention to PNG and that future version of MapServer will potentially not support GIF. The reason for this is that the patent holder for LZW image compression method that GIF uses, Unisys, demands that GIF image creators pay special licence fee to the company. As this is against the fundamental thinking of OpenSource software like MapServer, where everything is supposed to be free of charge, they have also developed MapServer no-GIF supporting version that does not fall under these patent regulations. Although the patent licence for GIF is not valid longer in United States (ended 20 of June 2003), the patent is still valid in many other countries, like Canada, France, Italy, Germany, the United Kingdom, and Japan (MapServer, 2004). Therefore it is essential for the developers of MapServer to offer alternatives to the GIF package (see little bit coloured information about the GIF patent debate at http://burnallgifs.org/).

Both MapServer packages (PNG/GIF) are however: "...compiled with GDAL and support PostGIS/OracleSpatial connectivity, OGC WMS and WFS standards, and Flash output." It is also possible to download MapServer with Oracle support instead of PostGIS but it was not my objective to do more research in that field of area.

To install MapServer it is recommended to extract the MapServer ZIP file in the folder "C:\mapserver\". There are few things that are important to do so that MapServer works properly. First, is to copy the file "C:\mapserver\mapserver.exe" to "C:\Program Files\Apache Group\Apache2\cgi-bin\". Secondly, you need to create the folder "C:\proj\" by copying "C:\mapserver\proj\" directory to the root. Finally, it is necessary to copy all the files in "C:\mapserver\lib\" to "C:\Winnt\" (in Windows 2000) or "C:\Windows\" (in Windows XP) so they can be accessed everywhere on the computer. Here is also possible to define "C:\mapserver\lib\" as a system variable in "Start > Control Panel > System > Advanced > Environmental Variables".

To check if MapServer works properly, one has to type "http://[localhost]/cgi-bin/mapserv.exe" into the web browser (Windows Explorer V6) and the following message should appear: "No query information to decode. QUERY_STRING is set, but empty." If that is the case the installation is completed. If not, more detailed information are to be found in the "readme.txt" file in the MapServer directory "C:\mapserver\".

MapServer Demo for Windows platforms

A DEMO can be obtained on MapServer homepage to test how things work. As the DEMO is made for UNIX platforms it is necessary to adjust it to a Windows platform.

I started with copying the DEMO files to "C:\Program files\Apache Group\Apache2\htdocs\Itasca\" and made following changes to the "demo.map" and "demo_init.html" files to adjust it to Windows 2000:

demo_init.html
Change:
<form method=GET action="/cgi-bin/mapserv">
To:
<form method=GET action="/cgi-bin/mapserv.exe">

Change:
<input type="hidden" name="map" value="/usr/local/apache
/htdocs/mapserver_demos/itasca/demo.map">
To:
<input type="hidden" name="map" value="C:\Program Files\Apache
Group\Apache2\htdocs\itasca\demo.map">

Change:
<input type="hidden" name="program" value="/cgi-bin/mapserv">
To:
<input type="hidden" name="program" value="/cgi-bin/mapserv.exe">

Change:
<input type="hidden" name="map_web_imagepath" value="/usr/local/apache/htdocs/tmp/">
To:
<input type="hidden" name="map_web_imagepath" value="C:\Program Files\Apache
Group\Apache2\htdocs\tmp\">

demo.map
Change:
WMS_ONLINERESOURCE "http://localhost.localdomain/mapserver_ demos/itasca/demo_init.html"
To:
WMS_ONLINERESOURCE "http://[localhost] /mapserver_demos/Itasca/demo_init.html"

Text Box 1 How to adjust MapServer DEMO from Linux to Windows platform

With these changes the DEMO should work on Win 2000 and one can explore the capabilities of MapServer that are demonstrated in the DEMO.

However, I had problem with running the DEMO on Win XP and I really do not why. The reason is probably that I downloaded the PNG version of MapServer 4.0 instead of the GIF version. Or, the reason could be somehow related to different platforms that I used on my two computers. My solution to this problem was not only to convert all GIF pictures to PNG but furthermore I defined symbols that are used in the map file in separate syntaxes and referred to these syntaxes inside my layer objects (see Text Box 2). The DEMO file, however, uses as default straight references to the symbol files as they appear and skips this intermediate.

One thing that is important to mention is that if you refer to folder it is important to refer "/folder/" instead of just "/folder". The reason is that otherwise does the computer think that the folder is file and it takes her few "seconds" to realise that it is just a folder. Every second is important in computing!!!

About my study on the function of MapServer I refer to next chapter that explores it further.

Instead of using straight referring like in the DEMO map file (marked with red below):

```
…
LAYER
  NAME 'regional airports'
  DATA airpt.shp
  STATUS default
  TYPE point
  CLASSITEM 'Class'
  CLASS
    EXPRESSION 'Regional'
    COLOR 0 0 0
    SYMBOL 'C:/Program Files/Apache Group/Apache2/htdocs/canada/icons/airport.png'
  END
END
…
```

I use following definition when using a symbol to represent my spatial data:

```
…
SYMBOL
  NAME 'airport'
  TYPE pixmap
  IMAGE 'C:/Program Files/Apache Group/Apache2/htdocs/canada/icons/airport.png'
END
…
LAYER
  NAME 'regional airports'
  DATA airpt.shp
  STATUS default
  TYPE point
  CLASSITEM 'Class'
  CLASS
    EXPRESSION 'Regional'
    COLOR 0 0 0
    SYMBOL 'airport'
  END
END
…
```

Text Box 2 Using symbol objects in MapServer.

MAPSERVER

In this section I am going to explore the fundamentals of MapServer, how it functions and what it is capable of. At the end I hope to have answered objectives 2-3 introduced earlier in this report.

MapServer is software as defined by OpenSource development environment, to make interactive web map applications. It is primarily developed by The University of Minnesota with a financial support from e.g. NASA. As it is OpenSource software many other institutions and individuals participate also in the development and the programming process. Some of its “out of the box” capabilities can be seen in text box 3..

The version I downloaded is MapServer 4.0 with PNG support, which can be downloaded for Windows at “http://mapserver.gis.umn.edu/win32bin/ms40_ png_pg.zip”.

- Vector formats supported: ESRI shapefiles, simple embedded features, ESRI ArcSDE (alpha release)
- Raster formats supported (8-bit only): TIFF/GeoTIFF, GIF, PNG, ERDAS, JPEG and EPPL7
- Quadtree spatial indexing for shapefiles
- Fully customizable, template driven output
- Feature selection by item/value, point, area or another feature
- TrueType font support
- Support for tiled raster and vector data (display only)
- Automatic legend and scalebar building
- Scale dependent feature drawing and application execution
- Thematic map building using logical or regular expression based classes
- Feature labeling including label collision mediation
- On-the-fly configuration via URLs
- On-the-fly projection

(Source: MapServer, 2004)

Text Box 3 Examples of the "out-of-the-box" features provided by the MapServer CGI application.

Architecture

MapServer is a server side program compiled with ANSI C/C++ and implemented as a CGI program. This can be explained in the way that the web server processes clients commands and returns finished results in the form of complete images and html made documents. This sometimes also called “thin client” and is opposite to “fat/thick client” where the client himself processes more or less his commands on the data, see also figures 7 & 8 (Lime, 2003; Santitamnont, 2001). The definition of CGI or in more formal “The Common Gateway Interface” is as followed (2004):

> The Common Gateway Interface (CGI) is a standard for interfacing external applications with information servers, such as HTTP or Web servers. A plain HTML document that the Web daemon retrieves is static, which means it exists in a constant state: a text file that doesn't change. A CGI program, on the other hand, is executed in real-time, so that it can output dynamic information.

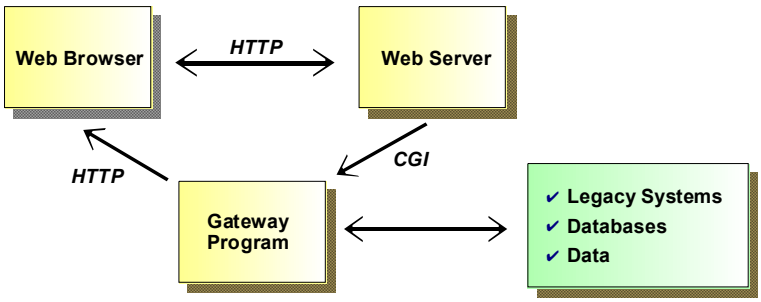

Figure 3 The architecture of CGI. The “Gateway” program is of course MapServer in our example. This architecture is sometimes also referred as “thin client” (Lime 2003).

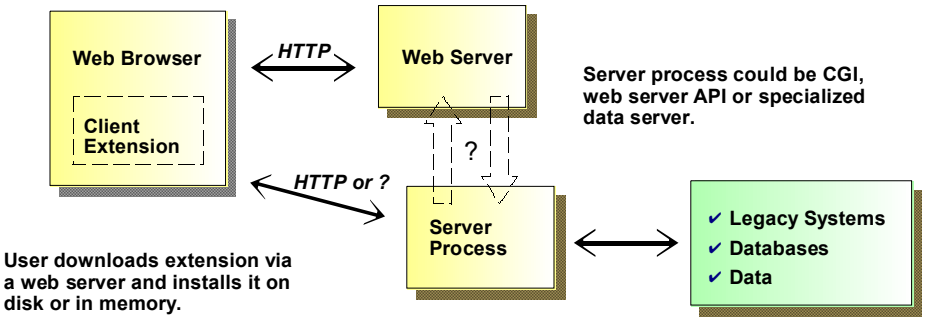

Figure 4 Structure of a client sided, or “fat client” GIS web application (Lime, 2003).

The CGI exchangeable variables that MapServer uses to enable "dynamic output information" are listed in detail in table 3 later in the report. The CGI variables are defined in the HTML document that is by default "static". When user makes a command, MapServer exchanges this variable with exact information that fits the operation that is to be preformed.

More materially, MapServer consist of several functions and files. First of all to mention are the two basic files; a map file and a HTML template file, that the editor himself creates and edits. Other files are included in the MapServer program, like e.g. "mapserv.exe" that invokes the map file and the CGI segment of MapServer. To start MapServer, one has to initialise MapServer.exe and connect the application to the map file that is to be used. That is done for example by using following expression:

<a href="[localhost]/cgi-bin/mapserv.exe?**map**=C:/Program Files/Apache Group/Apache2/htdocs/ canada/ canada.map">

This example starts the MapServer program, points to the map file (that includes the HTML template mentioned before) and returns map image as it is defined in the map file. Later in the process, MapServer adds to this URL information that enable browsing, turn on/off layers, querying etc. In the URL below I have done following changes to the default settings of the web map, zoomed in by the scale of 2 and turned on 2 layers that where not turned on by default:

http://[localhost]/cgi-bin/mapserv.exe? **zoom**=2 & **mode**=browse & **layer**=cities & **layer**=roads & **map**=C:\Program Files\Apache Group\Apache2\htdocs\canada\canada.map & **imgext**= -143.000000+31.605000+-50.000000+101.395000 & **imgxy**= 199.5 + 149.5 & **save query**=true&**img.x**=176 & **img.y**=189

Standard vector formats for MapServer are SHP files like used by ESRI's software ArcView. However, MapServer also supports various other vector formats. List of vector and raster formats supported by MapServer are showed in table 1. More detailed description on the map file, the HTML templates and CGI variables will be coverage in next chapters while figure 5 explains the author's view of MapServer architecture and interactions between the web client, the web server and MapServer.

Vector	**Raster**	
Formats	*Formats*	*Georeferencing*
Arc/Info Binary Coverage	TIFF	worldfile
ESRI Shapefile	GeoTIFF	inbuilt
DODS/OPeNDAP	GIF	worldfile
FMEObjects Gateway	PNG	worldfile
GML	JPEG	worldfile
IHO S-57 (ENC)	Erdas .LAN/.GIS	inbuilt
Mapinfo File		
Microstation DGN		
OGDI Vectors		
ODBC		
Oracle Spatial		
PostgreSQL		
SDTS / UK .NTF		
U.S. Census TIGER/Line		
VRT - Virtual Datasource		

Table 1 Vector and raster formats supported by MapServer 4.0.

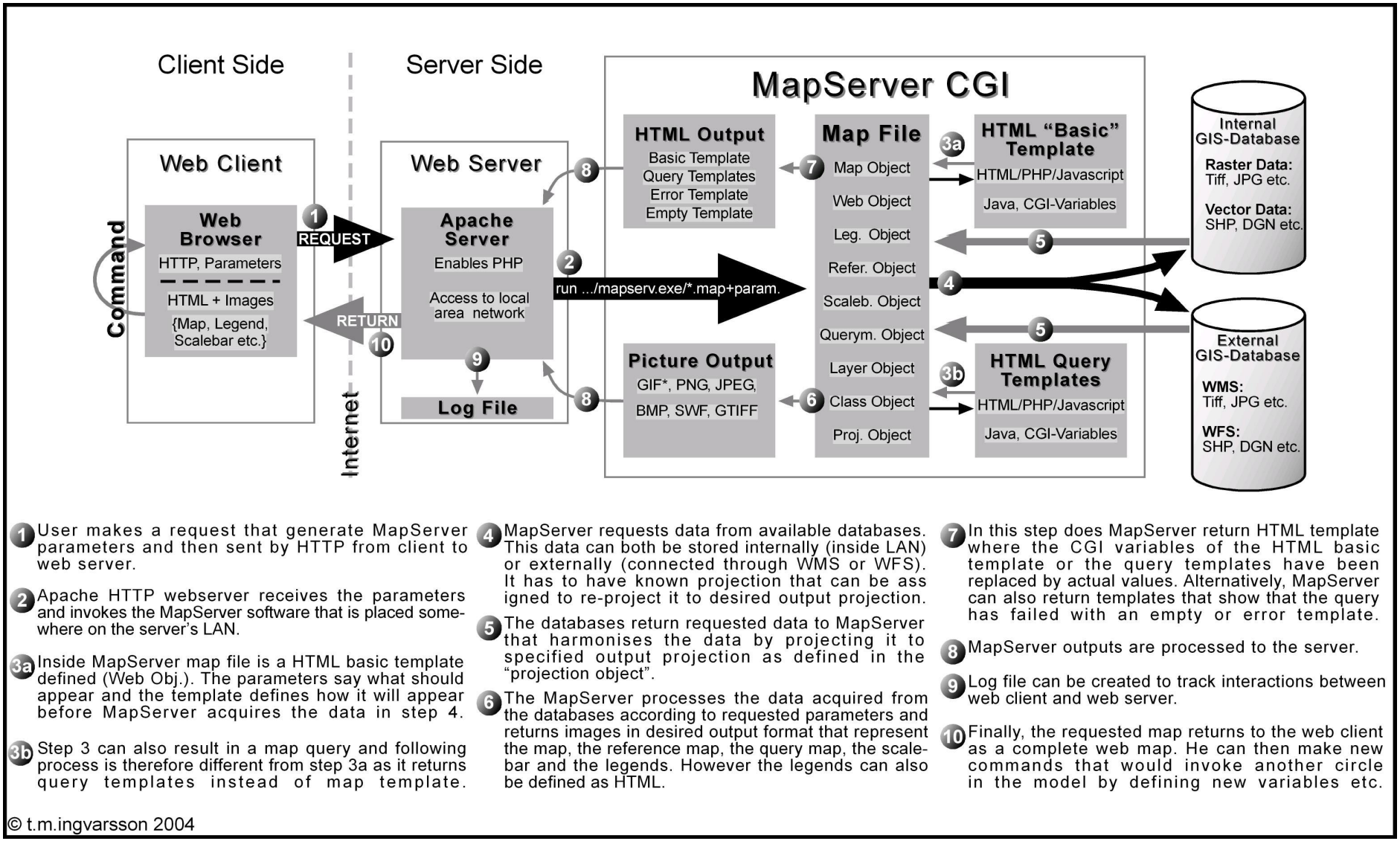

Figure 5 The interactions between web client, web server and MapServer in a MapServer CGI application.

The Map File

The map file is the foundation of MapServer applications. In the map file everything is defined that should appear or could possible appear in the web map application. The map file consist of several objects that all have specific functions. The "map object" is the master object of the map file as it both marks the start and the end of the map file. All other objects are appertaining to the map object. A list of the most important objects used by MapServer can be viewed in table 2.

Objects in Mapserver	
Objects	*Function*
Map Object	Is the master object of the map file and defines the application wide parameters.
Class Object	Sub-object of the Layer Object as each Layer Object must have at least one class.
Label Object	This Object is used to define labels and annotations. Used as sub-object to other objects.
Layer Object	Is the most used object in the map file as it defines the layers (both vector and raster) that make up the map. The order of the layers defines hierarchy in appearance (the first layer is on the bottom and the last layer on the top).
Legend Object	Used to define appearance of dynamic legends. Legend components are built automatically from class objects from individual layers
OutputFormat Object	Defines output formats that are used if the map file operates as a WMS.
Projection Object	Defines the projection of the map, and individual vector and raster files. MapServer is capable to use data in diverse projection on the fly.
QueryMap Object	Defines a mechanism to map the results of a query.
Reference Map Object	Used to make dynamic reference maps to guide users when exploring the web map.
Scalebar Object	Defines the appearance of a scalebar.
Style Object	Sub-object of the class object. The object is new in MapServer 4.0. and is intended to create standard symbolisation similar to the symbol object, that can be referred to inside the map file.
Symbol Object	Defines symbols to use in other objects. Can be defined as a sub-object of the map object or as a sub-object of the class object. If the former method is selected it is possible to refer to the symbol created anywhere in the map file.
Web Object	This object defines the interface of the web map and sources. It includes the HTML template, error template, empty query template and can be used to restrict max/min scale of the web map. It is also the home for information about Metadata.
References: http://ogr.maptools.org/ and http://mapserver.gis.umn.edu	

Table 2 The most important objects used by MapServer. How these objects are related to each other can be seen in figure 1.

It would be too long recitation to describe the function in detail of every object in MapServer or its parameters. Table 2 describes roughly the most important objects but not the hierarchy as they appear in the map file (Appendix A has more detailed description of objects and parameters in a map file while appendix B is attempt to interpret graphically the structure and potential relationships in a map file).

Basic map application could be described as having map object, web object and layer object (that have sub objects like e.g. the class object). Text box 4 below demonstrates the structure of a basic map file, but of course we could make the map interface nicer by adding scalebar, legends list and reference map, by defining equivalent objects in the map file.

```
MAP
  NAME [the name of the map]
  IMAGETYPE [the format of the displayed image]
  EXTENT [the extend of the start map in coordinates]
  SIZE [the extend of the map appearance in pixels]
  SHAPEPATH [the path to your data folder on your computer]
  UNITS [what units are you using]

  WEB
    TEMPLATE [the path to your HTML template on your computer]]
    IMAGEPATH [the path to a temp-folder where you keep images created]
    IMAGEURL [the URL of the temp-folder]
  END

  #SCALEBAR
  #REFERENCE
  #QUERYMAP
  #SYMBOL

  LAYER
    NAME [name of this layer]
    DATA [the name of the SHP file in the data archive]
    STATUS [should the map be turned on/off or by default]
    TYPE [is the data represented be point, line, polygon or is e.g. raster]
    CLASS
      COLOR [what is the colour of the element]
    END
  END

  LAYER
    …  [this layer would appear on the top of the layer described above]
  END
END
```

Text Box 4 Example of a simple map file.

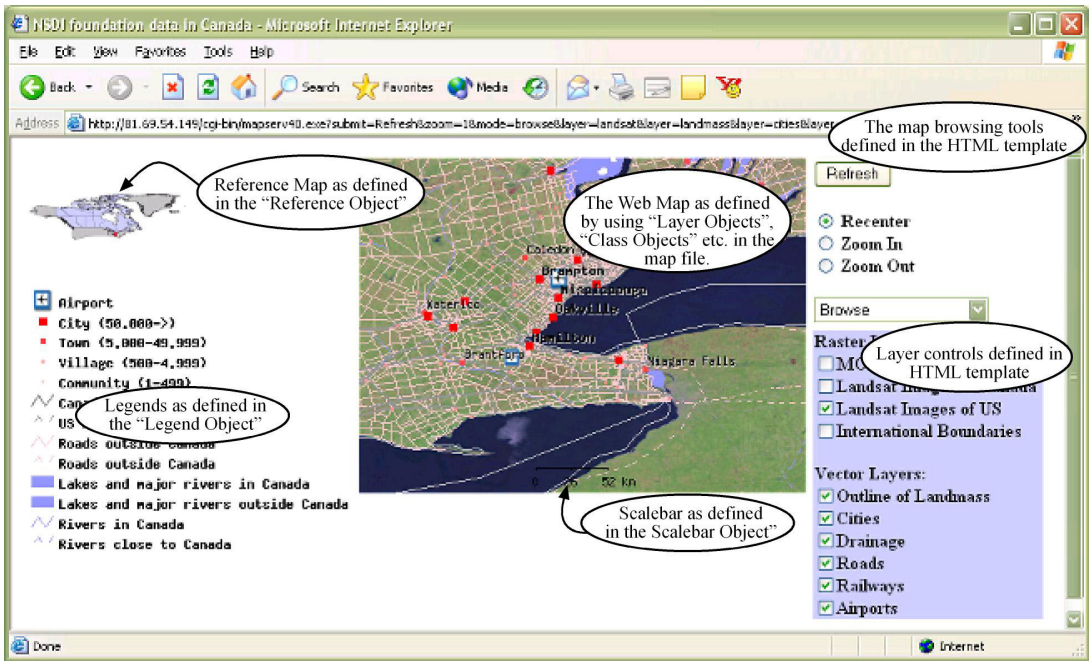

Figure 6 Demonstration of the link between the map file and the HTML template.

There exist editors found on the web that are extremely helpful in writing the map file, both properly and nicely. First to mention are the text editors TextPad and UltraEdit that can be enabled to highlight syntaxes (see figure 7). Moreover, some editors can be found that do not require that the user have extensive knowledge of MapServer scripting and have inbuilt data browsers and viewers. Examples of these are e.g. MapServer Workbench (figure 8) and MapLab. The latter one contributed by DM solutions, the largest single contributor to the development of MapServer supporting software. All these programs can function on Windows platform. There exist also utility programs that are only compiled and maintained for Linux or UNIX platforms. Example of this is e.g. MapDesk, a MapServer editor and viewer. Examples of most common utility programs that have been contributed to MapServer 4.0 can be explored at http://mapserver.gis.umn.edu/contributed.html.

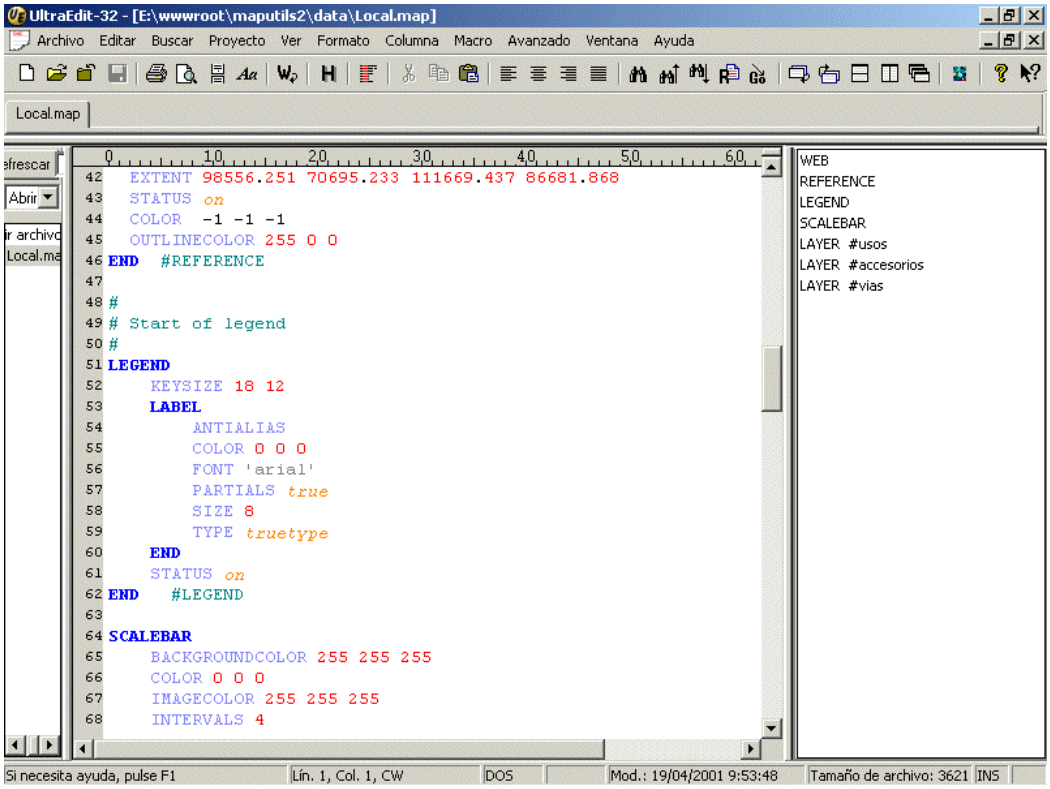

Figure 7 A screenshot of UltraEdit (v1.01) for MapServer.

- The map file is NOT case-sensitive.
- Strings containing non-alphanumeric characters or a MapServer keyword must be quoted. It is recommended to put all strings in double-quotes.
- There is a maximum of 50 layers per map file. This can be changed by editing the configurations in the source script.
- File paths may be given as absolute paths, or as paths relative to the location of the map file. In addition, data files may be specified relative to the shapepath.
- The map file has a hierarchical structure, with the Map object being the "root". All other objects fall under this one.
- Comments are designated with #.

(Source: MapServer, 2004)

Text Box 5 Facts that are important to note when handling MapServer map file.

- Build overview levels for large raster images. Overviews can be implemented as a group of raster layers at different resolutions, using MINSCALE and MAXSCALE.
- Pre-process RGB images to eight bit with a colour map to reduce the amount of data that has to be read, and the amount of computation to do on the fly.
- For large images use tiling to reduce the overhead for loading a view of a small area.
- Ensure that the image is kept on disk in the most commonly requested projection to avoid on-the-fly image warping.

(Source: MapServer, 2004)

Text Box 6 How to get more effective display of raster data in MapServer.

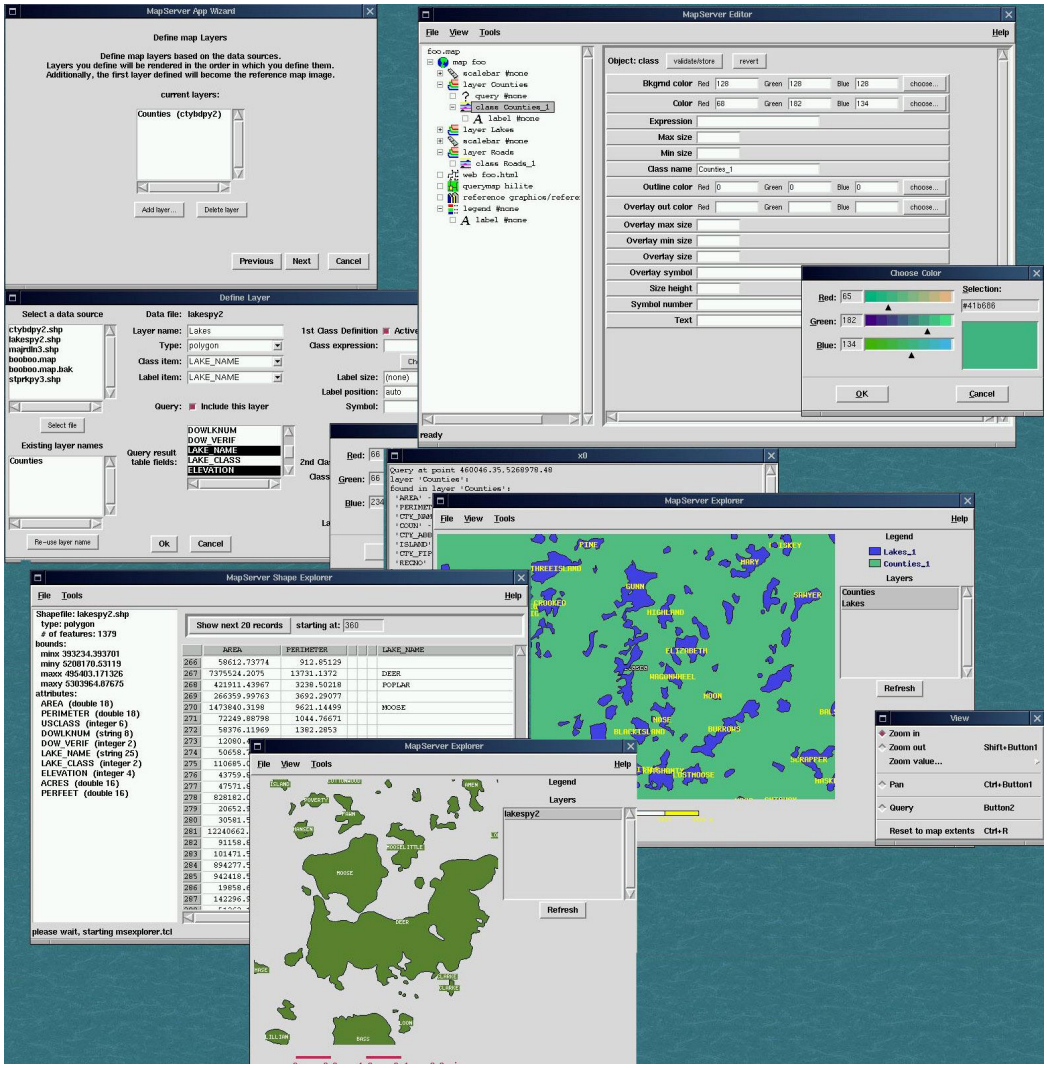

Figure 8 MapServer Workbench

MapServer templates – The HTML template

The HTML "basic" template as described in figure 5 is the most important html template used in MapServer applications as it is the interface of the data defined in the map file. In this template we define what data we want to represent in our browser and how we are going to explore the data, like e.g. zooming, panning, querying etc. The simplest way of doing the HTML template is by using the HTML language. However, if we want fancy buttons or functions in our browser, there exist also other possibilities, like e.g. using Javascript or Java. It is though important for MapServer that the HTML templates names end with "*.html/*.html/*.xml/*.wml"

Other templates that we need to define or create are the query templates, where we structure results from queries, and error/empty templates that appear if failure occurs in a query.

Like mentioned before, MapServer uses CGI to create dynamic HTML templates. This is done by making standard template that does not have any fixed variables, but dynamic which are simultaneously on the fly retrieved from the map file. This is done by using CGI variable inside brackets in the HTML template. These variables are later exchanged by the CGI software, in this example the MapServer, and a new static sub-HTML template created and sent to the client browser. These dynamic CGI variables are listed as described by the MapServer developers in the table 3 below and further explained in text box 7.

MapServer CGI variables to use in HTML templates	
General	
[version]	The MapServer version number.
[id]	Unique session id.
[host]	Hostname of the web server.
[port]	Port the web server is listening to.
[post or get variable name], [post or get variable name_esc]	The contents of any variables passed to the MapServer, whether they were used or not, can be echoed this way. One use might be to have the user set a map title or north arrow style in an interactive map composer. The system doesn't care about the values, but they might be real important in creating the final output, e.g. if you specified a CGI parameter like myvalue=.... you can access this in the template file with [myvalue].
File Reference	
[img]	Path (relative to document root) of the new image, just the image name if IMAGE_URL is not set in the map file.
[ref]	Path (relative to document root) of the new reference image.
[legend]	Path (relative to document root) of new legend image rendered by the MapServer.
[scalebar]	Path (relative to document root) of new scalebar image.
[queryfile]	Path to the query file (if savequery was set as a CGI Parameter).
[map]	Path to the map file (if savemap was set as a CGI Parameter).
Image Geometry	
[center]	Computed image center in pixels. Useful for setting imgxy form variable when map sizes change.
[center_x], [center_y]	Computed image center X or Y coordinate in pixels.
[mapsize], [mapsize_esc]	Current image size in cols and rows (separated by spaces).
[mapwidth], [mapheight]	Current image width or height.
[scale]	Current image scale. The exact value is not appropriate for user information but essential for some applications. The value can be rounded e.g. using JavaScript or server side post processing.
[cellsize]	Size of a pixel in the current image in map units. Useful for distance

	measurement tools in user interfaces.
Map Geometry	
[mapx], [mapy]	X and Y coordinate of mouse click.
[mapext], [mapext_esc]	Full mapextent (separated by spaces).
[minx], [miny], [maxx], [maxy]	Minimum / maximum X or Y coordinates of new map extent.
[dx], [dy]	The differences of minimum / maximum X or Y coordinate of new map extent.
[rawext], [rawext_esc]	Raw mapextent, that is the extent before fitting to a window size (separated by spaces). In cases where input came from imgbox (via Java or whatever) rawext refers to imgbox coordinates transformed to map units.
[rawminx], [rawminy], [rawmaxx], [rawmaxy]	Minimum / maximum X or Y coordinates of a raw map/search extent.
[maplon], [maplat]	Longitude / latitude value of mouse click. Available only when projection enabled.
[mapext_latlon], [mapext_latlon_esc]	Full mapextent (separated by spaces). Available only when projection enabled.
[minlon], [minlat], [maxlon] [maxlat]	Minimum / maximum longitude or latitude value of mapextent. Available only when projection enabled.
Layer	
[get_layers]	All active layers formatted for inclusion in a URL. (i.e. layer=layer1&layer=layer2 and so on...). Used for a "GET" request.
[layers] | [layers_esc]	All active layers space delimited. Used for a "POST" request.
[toggle_layers] | [toggle_layers_esc]	List of all layers that can be toggled, i.e. all layers defined in the map file) which status is currently not default.
[layername_check | select]	Used for making layers persistent across a map creation session. String is replaced with the keyword "checked", "selected" or "" if layername is on. Layername is the name of a layer as it appears in the map file. Does not work for default layers.
[layername_meta data key]	Layer meta data access (e.g. [streets_build] the underscore is essential).
Zoom	
[zoom_minzoom to maxzoom_check|select]	Used for making the zoom factor persistent. Zoom values can range from -25 to 25 by default. The string is replaced with the HTML keyword "checked", "selected" or "" depending on the current zoom value. E.g. if the zoom is 12, a [zoom_12_select] is replaced with "selected", while a [zoom_13_select] in the same HTML template file is not.
[zoomdir_-1|0|1_check|select]	Used for making the zoom direction persistent. Use check with a radio control or select with a selection list. See the demo for an example. The string is replaced with the HTML keyword "checked", "selected" or "" depending on the current value of zoomdir.
Query	
[shpext], [shpext_esc]	Extent of current shape plus a 5 percent buffer. Available only when processing query results.
[shpminx], [shpminy], [shpmaxx], [shpmaxy]	Minimum / maximum X or Y coordinates of shape extent. Available only when processing query results.
[shpmid]	Middle of the extent of current shape. Available only when processing query results.
[shpmidx], [shpmidy]	X or Y coordinates of middle of the extent of the current shape. Available only when processing query results.
[shpidx]	Index value of the current shape. Available only when processing query results.
[tileindex]	Index value of the current tile. If no tiles used for the current shape this is replaced by "-1". Available only when processing query results.
[DBASE item name], [DBASE item name_esc], [DBASE item name_raw]	Item name from the attribute table of a queried layer. Only attributes for the active query layers are accessible. Case must be the same as what is stored in the DBASE file. ArcView, for example, uses all caps for shapefile field names. Available only when processing query results.

[Join name_DBASE item name], [Join name_DBASE item name_esc], [Join name_DBASE item name_raw]	One-to-one joins: First the join name (as specified in the map file) has to be given, second the tables fields can be accessed similar to the layers attribute data. Available only when processing query results.
[join_Join name]	One-to-many joins: The more complex variant. If the join type is multiple (one-to-many) the template is replaced by the set of header, template file and footer specified in the map file.
[nr]	Total number of results. Useful in web header and footers. Available only when processing query results.
[nl]	Number of layers returning results. Useful in web header and footers. Available only when processing query results.
[nlr]	Total number of results within the current layer. Useful in web header and footers. Available only when processing query results.
[rn]	Result number within all layers. Starts at 1. Useful in web header and footers. Available only when processing query results.
[lrn]	Result number within the current layer. Starts at 1. Useful in query templates. Available only when processing query results.
[cl]	Current layer name. Useful in layer headers and footers. Available only when processing query results.
References: http://mapserver.gis.umn.edu/doc40/template-reference.html	

Table 3 Variables used in MapServer templates.

```
<!-- DISPLAY THE MAPSERVER-CREATED MAP IMAGE -->
   <td align="center" valign="top">
   <input type="image" name="img" src="[img]" width="[mapwidth]" height="[mapheight]">
   <p>Map Scale 1:[scale] </p>
</td>
<!-- DISPLAY THE MAPSERVER-CREATED MAP IMAGE -->
   <td align="center" valign="top">
   <input type="image" name="img" src="http://81.69.54.149/canada/tmp/
   Canada10781024863044.png" width="400" height="300">
   <p>Map Scale 1:73233381.960000 </p>
</td>
[img]: Path of the new map image. Usually the URL is defined in the WEB object inside the map file.
[mapwidth]: Returns the map width as defined in the map object inside the map file.
[mapheight]: Returns the map width as defined in the map object inside the map file.
[scale]: Returns the active scale of the viewed map.
```

Text Box 7 Example of using CGI variables inside MapServer HTML Template.

After invoking “mapserver.exe” and the map file, the HTML template should be open and running. Behind the HTML template is the map file (the connections between them is somehow in a loop, see figure 3) that retains the path to the spatial information to be available for the application. The HTML template is just an access point and an interface to that information. It is possible to customise this interface in many ways and with diverse methods. Like mentioned before, HTML is the most basic method to write this template, but by making use of e.g. Javascript the interface can be made more fancy, and maybe more useable for users and in divers applications. To be able to use Javascript, one has to have basic knowledge of the script language although it is possible to copy existing scripts found on the web and employ in ones own application. Moreover it is possible to employ “out of the box” software and scripts that developers of MapServer have contributed.

"J-box", "MapClient" and "Rubber-band-zoombox" are all contributions to help MapServer users to make interesting and fancy MapServer applications. These scripts can make functions like "click-on-buttons", "panning" and "rubber band box" to name a few. The last function mentioned has the most apparent function in zooming but can also be used in spatial queries. For example when someone wants to now all elements/objects in particular area, he or she can draw the rubber band box over that area and the computer returns all objects that are inside the box (or overlap, or outside etc. it depends on the definition of the rubber band box) These scripts can be accessed through MapServer utility page: http://mapserver.gis.umn.edu/contributed.html.

Now we should return again to the fundamentals of the HTML template. Text box 8 shows simple example of a HTML template without any appended Javascript functions. The text that is marked with red is descriptive and has nothing to do with the function of the template. The text in bold is fundamental HTML commands and the text in blue is the CGI connection to the MapServer as described earlier. E.g. the following line in the template is the "cities" part of the check box that can be seen in bottom right part of figure 9:

<**input type**="checkbox" **name**="layer" **value**="cities" [cities_check]><**b**>*Cities* <**br**>

The "input type" and "name" commands define that following variables are part of a checkbox called "layer". The value of this particular feature in the checkbox is "cities" and the text in the brackets is a straight reference to a layer with the same name in the map file (see also the [layername_check | select] command in table 3). In the same way are functions like "Zoom In" and "Zoom Out" defined and etc.

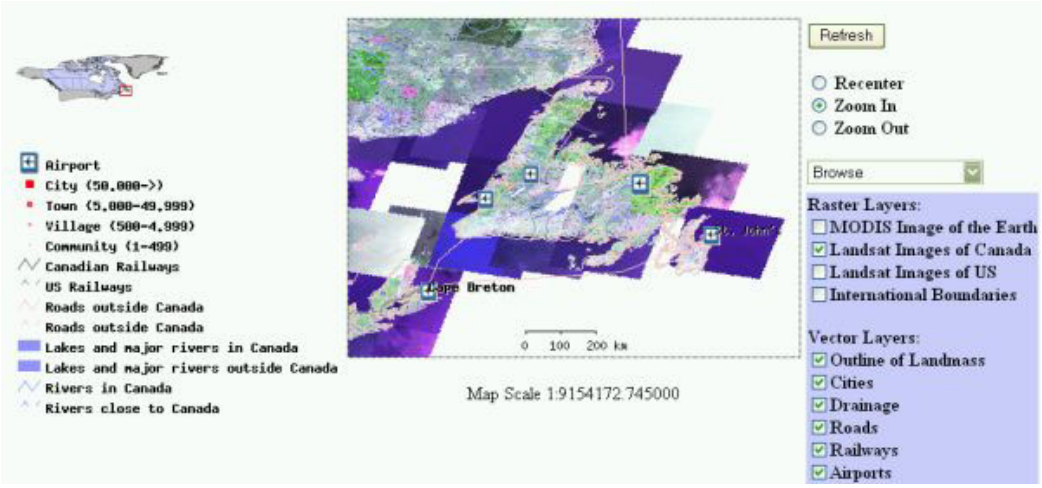

Figure 9 The HTML interface – Checkboxes

It is also possible to do the HTML template completely automatically with the assistance of readymade software like MapLab that has already be mentioned in connections to editing and viewing map files. The "GMapFactory" feature of MapLab enables MapServer user to create map application in a wizard form (see figure 10).

```
<html><head><title>The name of the Web-Map</title></head><body>
<!-- START OF MAPSERVER FORM -->
<form name="mapserv" method="GET" action="[program]">

<!-- START OF HTML INTERFACE TO WEB-BROWSER -->
<!--REFERENCE AND LEGENDS-->
<table border="0"><tr><td align="left" valign="top" >
    <input type="image" name="ref" src="[ref]" border="0"></p>
    <img name="legend" src="[legend]"></p>
</td>

<!-- DISPLAY THE MAPSERVER-CREATED MAP IMAGE -->
<td align="center" valign="top">
    <input type="image" name="img" src="[img]" width="[mapwidth]" height="[mapheight]"
    border="0">
    <p>Map Scale 1:[scale] </p>
</td>

<!-- ZOOM/PAN - REFRESH -->
<td align="left" valign="top">
    <table align="left" valign="top" ><tr><td height="150" align="left" valign="top">
    <p><input type="submit" name="submit" value="Refresh"></p>
    <p><input type="radio" name="zoom" value="1" [zoom_1_check] > <b>Recenter</b><br>
    <input type="radio" name="zoom" value="2" [zoom_2_check]> <b>Zoom In</b><br>
    <input type="radio" name="zoom" value="-2" [zoom_-2_check] > <b>Zoom Out</b>
    </p>

<!-- SPECIFY MAP MODE -->
    <div><select name="mode">
    <option value="browse" [browse_select] checked>Browse </option>
    <option value="query" [query_select]>Query Single Layer</option>
    <option value="nquery" [nquery_select]>Query Multiple Layers</option>
    <option value="map" [map_select]>Map</option>
    </select></div>
    </td>
    </tr>

<!—SELECT LAYERS -->
    <tr><td align="left" valign="top">
    <b>Raster Layers:</b><br>
    <input type="checkbox" name="layer" value="modis" [modis_check]><b>MODIS</b><br>
    <p><b>Vector Layers:</b><br>
    <input type="checkbox" name="layer" value="cities" [cities_check]><b>Cities</b> <br>
    <input type="checkbox" name="layer" value="roads" [roads_check]><b>Roads</b> <br>
    </td></tr></table>
    </td>
</tr>
</table>

<!-- HIDDEN MAPSERVER CGI VARIABLES -->
<input type="hidden" name="map" value="[map]">
<input type="hidden" name="imgext" value="[mapext]">
<input type="hidden" name="imgxy" value="[center]">
<input type="hidden" name="savequery" value="true">
</body>
</html>
```

Text Box 8 Example of a simple HTML template.

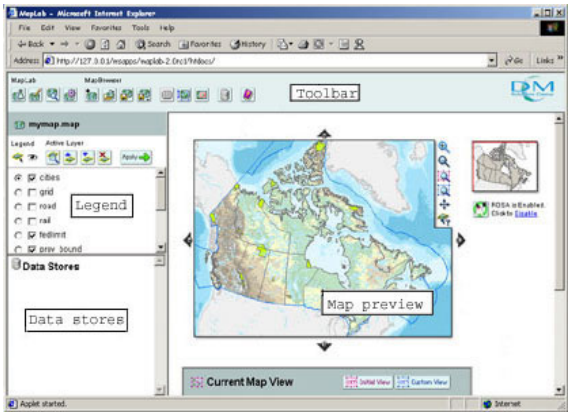

Figure 10 The standardised look/form of MapLab browser made by GMapFactory

MapServer templates – The Querymap templates

One important feature of MapServer is its capability of graphically displaying queries that are made by mouse dragging/clicking or by using simple textual applications. The results of the queries are displayed in so called queries templates, e.g. by highlighting the features that are resulting from the query. Although it seems that just one template appears when making a query it default consists of five templates that are somehow summed together.

SEARCH WINDOW: -71.582541 46.537154 -70.855979 47.082389

MAP COORDINATES:
min X (-71.628886) and min Y (46.537154)
max X (-70.809634) and max Y (47.082389)

YOUR MOUSE CLICK COORDINATES: -71.215627, 46.811589

NUMBER OF LAYERS QUERIED: 1

NUMBER OF RESULTS: 1

Name of Layer: pop4

Nr	Name	Category	District
1	Québec Ville (2)		Quebec/Québec

Map made by T.M.Ingvarsson

header.html

```
<html>
<head>
  <title>Canada - Query Results</title>
</head>
<body bgcolor="#ffffff">
<img border="2" src="[img]">
<table border="0">
 <tr>
  <td>
   <p><b>SEARCH WINDOW</b>: [imgext]</p>
   <p><b>MAP COORDINATES</b>:<br>
   min X ([minx]) and min Y ([miny])<br>
   max X ([maxx]) and max Y ([maxy])</p>
   <p><b>YOUR MOUSE CLICK COORDINATES</b>:
      [mapx], [mapy] </p>
   <p><b>NUMBER OF LAYERS QUERIED</b>: [nl]</p>
   <p><b>NUMBER OF RESULTS</B>: [nr]</p>
  </td>
 </tr>
</table>
<br>
```

[layername]_header.html

```
<b>Name of Layer: [cl]</b><br>
<br><table align="left"> <b><tr><th>Nr</th>
<th>Name</th><th>Category</th><th>District</th>
</tr></b>
```

[layername]_query.html

```
<tr><td>[lrn]</td><td>[GEONAME]</td>
<td>[GENERIC]</td><td>[REGIONNAME]</td></tr>
```

[layername]_footer.html

```
</table><p>
<br>
<br>
```

footer.html

```
<h4>Map made by <A HREF="mailto:t.m.ingvarsson
@student.tudelft.nl">T.M.Ingvarsson</A></body>
</html>
```

Figure 11 An example of a Querymap results in MapServer can be seen on the left side of this figure. Hence that this "interface" is a composition of five files that are demonstrated on the right side of the figure. The italic-bolded text in the middle is the source name of the HMTL files that are shown on the right side.

When making a query (single or multiple) in MapServer the program returns the results in a predetermined way. It splits the query into five parts that together composite in one template. First to mention is the header and the footer files that mark both the start and the end of the query template. It gives the query result a standardised look. The header and footer files are defined as a parameter of the Web Object, see also figure 11.

Between the header and the footer are three files: “layer_header”, “layer_query” and “layer_footer”. These files are created specifically for each layer in the map file that is to be queried. The layer header and footer are defined as parameters of the Layer Object as they are constant for this particular layer. The “layer_query” is however defined inside the Class Object as it is variable, considering what element is queried each time.

WMS

MapServer is capable to act simultaneously as a WMS client and a WMS server. When MapServer connects to an external WMS server he retrieves the information that is kept in a XML document, acting as an access point to the external datasets. He uses specific protocols according to OGC specification to establish the connection and the retrieved “web map” is placed in the map file as a regular layer object (as it was hosted on local computer). Normally, the connection is defined as a metadata in the LAYER object but can alternatively also be defined as a sub-parameter of the CONNECTION (see further text box 9).

```
LAYER
  NAME                    "[your given name of this specific layer]"
  TYPE                    RASTER
  STATUS                  on
  CONNECTION              "[the source of the WMS server without any parameters e.g.
                          http://atlas.gc.ca:80/cgi-bin/atlaswms_en?"
  CONNECTIONTYPE          WMS
  METADATA
    "wms_srs"             "[Projection of the WMS maps e.g. EPSG:4326]"
    "wms_name"            "[name of the layer on the WMS server]"
    "wms_abstract"        "[description of the layer]"
    "wms_server_version"  "[vrs. of WMS OGC specification e.g. 1.1.0.]"
    "wms_formatlist"      "[list of available formats]"
    "wms_format"          "[the format to be used in your GetMap request]"
  END
END
```

Canadian WMS servers:
http://atlas.gc.ca/site/english/data_services/web_mapping_service.html
http://dev.geographynetwork.ca/data/freedata.html

American WMS servers:
http://wms.jpl.nasa.gov/
http://terraserver.microsoft.com/

WMS and WFS servers with global data, provided by OGC (however they still tend to concentrate mostly on North America):
http://ogc.compusult.nf.ca/cgi-bin/OWS/phtml/service_manager/services/services.phtml
http://ogc.compusult.nf.ca/cgi-bin/MMI/phtml/service_manager/services/services.phtml
http://ogc.compusult.nf.ca/cgi-bin/english/phtml/service_manager/services/services.phtml

Text Box 9 Example of a simple WMS connection in MapServer map file.

Each WMS server that is implemented by employing MapServer WMS server capabilities must have own map file, but can have several sub layers. This file is just a regular map file as used in MapServer web applications, but with mandatory metadata and parameters defined to enable and standardise external queries according to OGC specification.

Examples of WMS client/server are shown in text box 9 and text box 10 respectively. Good documentation can however be found at "http://mapserver.gis.umn.edu/".

```
MAP
  NAME                      The name of the WMS server
  PROJECTION
  …                         The available OUTPUT projections
  END
  WEB
    METADATA
      "wms_title"           The name of the WMS server
      "wms_onlineresource"  URL that is used to access your server
      "wms_srs"             Can be skipped if PROJECTION Object is defined.
    END
  END
  LAYER
    NAME                    The name of each layer used by external web client to extract.
    PROJECTION              The projection of this layer, recommended to be same as the map file.
    METADATA
      "wms_title"           The name of each layer used by external web client to extract.
      "wms_srs"             (optional since the layers inherit the map's SRS value)
    END
  END
END
```

Text Box 10 Example of a WMS server map file.

WFS

Connections to WFS services have simpler structure of connection in the MapServer map file than WMS. The only change that has to make to the Layer Object to connect it to WFS server is to delete the "DATA" parameter and place instead the parameters: "CONNECTION" and "CONNECTIONTYPE" as similar to WMS. The difference there between (WMS and WFS) is that it not important to fill in parameters for Metadata for displaying the WFS layer. A simple example of a WFS layer in a map file is shown in text box 11.

MapServer can also act as a WFS server. According MapServer "how-to" (2004):

> A WFS (Web Feature Service) publishes feature-level geospatial data to the web. This means that instead of returning an image, as MapServer has traditionally done, the client now obtains fine-grained information about specific geospatial features of the underlying data, at both the geometry AND attribute levels. As with other OpenGIS specifications, this interface uses XML over HTTP as it's delivery mechanism, and, more precisely, GML (Geography Markup Language), which is a subset of XML.

I did not do any experiments with WFS services and according to documentation (MapServer, 2004) there are many limitations of WFS compared to WMS in terms of feasibility, simplicity and efficiency. It is therefore essential to do more research on

this subject as WFS services are theoretically much better option in sharing spatial vector data than WMS.

```
LAYER
  NAME                      "[your name of this specific layer]"
  TYPE                      POINT|LINE|POLYGON
  CONNECTIONTYPE            WFS
  CONNECTION                "http://[WFS_URL_datalayer]"

  PROJECTION
    "init=epsg:4326"        #epsg is projection code
  END

  METADATA
    "wfs_connectiontimeout" "[time in seconds, by default 30 sec.]"
  END

  CLASSITEM                 "[Name of class]"
  CLASS
    SYMBOL                  "[Symbol type]"
    COLOR                   R G B
  END
END
```

Text Box 11 Example of a simple WFS connection in MapServer map file.

CONCLUSION OF OBJECTIVES

Following conclusions are the answers to the objectives that were put forward in the beginning of the report. They are reasoned from my research and experiment on the MapServer software and from various references mostly found on MapServer homepage (documentation, FAO and "messageboards"). There are not found any published literature specifically about MapServer (at least not on Amazon or Oxford Press) apart some articles on the internet and many of them out of date. As result I had to base my research first and foremost on my experiments and by monitoring the developer's messageboards.

Let start with answering the first objective:

1. *The availability of OpenSource software's on the internet to use in interactive web-mapping applications and compatibility with standard GIS applications like SQL.*

There is great availability of OpenSource software found on the internet that can be used for free in GIS applications or to share geospatial data over the internet trough web map interfaces. I did not test all of them but one has only has to browse "www.freegis.org" to explore many of the available OpenSource GIS software's. Examples of products are e.g.:

- MapServer, the main topic of this report, impressive software to make web map applications and is even developing into being fully equipped GIS enabled web application where it will hopefully be possible to make complex queries into diverse spatial data. That is however not easily implemented with current version. Presently its purpose is to make "geographic image maps, that is, maps that can direct users to content" through web map applications. It is also serviced with

multiple utility programs that enable both novices and experts to use the server in an efficient way. MapServer is server sided (thin client approach) software that means that potential users do not need any supplemental software to operate the web map as the "centralised server" performs all tasks and delivers them process finished to the client in form of HTML templates and images (MapServer, 2004; Trinidad et all, 2000).

- GRASS (Geographic Resources Analysis Support System), an OpenSource GIS software that can be downloaded through "http://grass.itc.it". It is compatible with UNIX, LINUX and Windows platforms. In many ways similar to the commercial GIS applications, like ESRI, Intergraph and Bentley products, but can be obtained for free! However, some user would think that GRASS is not very "user-friendly" as many of the commands are typed in a command window instead of the graphical solution that the commercial software offers. The latest stable version of GRASS is 5.0 (Grass, 2004).
- Geo-Tools, is a Java based GNU LGPL licensed software for developing interactive web maps (GNU http://www.gnu.org/copyleft/lesser.html stands for "General Public Licenses/Lesser General Public License" that is part of the OpenSource licence definition). It is developed by The University of Leeds and its main file format is SHP. However, unlike MapServer, Geo-Tools is client based software (fat client approach) where as it extracts and uses subsets of the spatial data that is otherwise kept on the web-map server. It enables that it is possible to do multiple and complex queries to the data. However the drawback is that it does not necessarily work the same way on all platforms or in all browsers. The reason is that the client has to make the queries himself and download supplementary software's to use for this purpose (Geo-Tool, 2004; Trinidad et all, 2000). I did not do any further research on this program as I just discovered it when I was finishing this report. That confirms my idea that no report on web software's can be fully comprehensive as the development is too fast and implemented by number of diverse producers.
- Apache HTTP server is an OpenSource server to share data over the internet. It is a good alternative to use with applications like MapServer and MySQL.
- MySQL is an OpenSource relational database server that can be used as a source for data in e.g. MapServer applications. The weight of MySQL in MapServer application will probably grow in the future with new versions of MapServer.
- PostGIS ads support for geographic objects to the PostgreSQL object-relational database. In effect, PostGIS "spatially enables" the PostgreSQL server, allowing it to be used as a backend for GIS. No further research was done in this subject.

I could not find any composition of OpenSource software that could be used to build up fully equipped GIS applications in a web-map interface, although I think that it will be possible in near future and then has MapServer the greatest potential to do so, alone or in combination with other software vendor. Important to notice here is that I have not tested GEO-Tools and I could not find any proper documentation or snapshots that would give me answer to this question.

2. *To research the widely spread OpenSource software; MapServer and check its compatibility with OGC specifications and protocols (WMS and WFS), Apache server, MySQL, and applicability for practical uses in GIS applications.*

MapServer is "not a full-featured GIS system" but its potentials are enormous and one has just to look at the examples that are provided on "http://mapserver.gis.umn.edu" to get grab on its potentials. MapServer is developed according to specifications made

by OGC and can both act as a client or as a server for WMS and WFS. It is also compatible to GML standards set by OGC but my knowledge of GML is too restricted to be able to explain that further. MapServer can work on all common platforms like Mac/OS, UNIX, LINUX and Win XP/00/NT//98/95 and it works also with most common servers like Apache, MS IIS and Netscape Servers to name a few.

3. *To check the capability of MapServer to use different file formats, both for vector and raster. Emphasise was on vector file formats used in software's from ESRI, AutoCad, Intergraph and Bentley.*

It is possible to use variety of diverse data at once in MapServer applications without any problem. In table 1 is a list of supported formats, both raster and vector. Moreover it is possible to keep spatial data in relational databases like provided by Oracle, MySQL and PostgreSQL/PostGis. However, like mentioned before, limitation of the software prevent that it is capable to make complex spatial queries to the data as it is not possible to create "new" data from combined dataset and represent it on the fly. One of the reasons is that MapServer works as a thin client and there for are all queries and commands processed by the web server that afterwards sends completed image maps and HTML templates to the web client. If complex queries would be enabled, one could see that it would slow down the web server computer and make MapServer subsequently less feasible option in terms of efficiency and simplicity.

There are however limited queries possible like e.g. by using the mouse to query available datasets of specific area, or by typing simple questions in a form that results in simple answer like: "highlight all roads that are over 50 km in length" etc. (it is not possible to make queries like "make a new point where a road cross a river" etc). I can moreover not rule out the possibility to use MapServer for routing service as that kind of services are theoretically and practically mostly done by thin clients. As MapServer can solve simple queries I can imagine that it is possible to simplify complex queries for MapServer to solve and reveal. For instance by showing the shortest path between A and B. By using MySQL address/road infrastructure relational database it would connect A to road 1 and B to road 10. The MySQL database processes the query and returns that road 2, 3, 4, 5, 6, 7, 8 and 9 as the shortest route. MapServer would afterwards display these results as a simple object query (highlight lines: 1, 2 … 10). I think this is just matter of solving the MySQL part of the process (how are we going to build up such a database?) as the MapServer part would be relatively easy. This subject needs of course clearly more research of my behalf, because in my view is that it would open new doorways in utilising MapServer.

4. *To check the possibility to use OpenSource software for commercial uses.*

The OpenSource licence not only allows a person to modify the program, but it also allows people to use it for commercial purposes like in business to make profit from its use. Although OpenSource is the commonly referred name of this kind of licence it is more like of synonym for many different license, with different function, but abide common rules defined in the OpenSource definition, presented in text box 12. Example of these licenses are e.g.: Academic Free License, Apache Software License, Common Public License, GNU General Public License (GPL), GNU Lesser General Public License (LGPL), Mozilla Public License (MPL) etc.

According to the 6th paragraph of the OpenSource definition, version 1.9, presented in text box 12, it can be interpreted that anybody can make commercial use of MapServer, no matter in what purpose. MapServer is very efficient software to

redistribute spatial data over the internet and act as a clearinghouse to spatial data infrastructures. What makes it so good is its compatibility to OGC standards and specifications, and its simple structure that enables it to be both quick and reliant.

1. Free Redistribution
The license shall not restrict any party from selling or giving away the software as a component of an aggregate software distribution containing programs from several different sources. The license shall not require a royalty or other fee for such sale.

2. Source Code
The program must include source code, and must allow distribution in source code as well as compiled form. Where some form of a product is not distributed with source code, there must be a well-publicized means of obtaining the source code for no more than a reasonable reproduction cost–preferably, downloading via the Internet without charge. The source code must be the preferred form in which a programmer would modify the program. Deliberately obfuscated source code is not allowed. Intermediate forms such as the output of a preprocessor or translator are not allowed.

3. Derived Works
The license must allow modifications and derived works, and must allow them to be distributed under the same terms as the license of the original software.

4. Integrity of The Author's Source Code
The license may restrict source-code from being distributed in modified form only if the license allows the distribution of "patch files" with the source code for the purpose of modifying the program at build time. The license must explicitly permit distribution of software built from modified source code. The license may require derived works to carry a different name or version number from the original software.

5. No Discrimination Against Persons or Groups
The license must not discriminate against any person or group of persons.

6. No Discrimination Against Fields of Endeavor
The license must not restrict anyone from making use of the program in a specific field of endeavor. For example, it may not restrict the program from being used in a business, or from being used for genetic research.

7. Distribution of License
The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.

8. License Must Not Be Specific to a Product
The rights attached to the program must not depend on the program's being part of a particular software distribution. If the program is extracted from that distribution and used or distributed within the terms of the program's license, all parties to whom the program is redistributed should have the same rights as those that are granted in conjunction with the original software distribution.

9. License Must Not Restrict Other Software
The license must not place restrictions on other software that is distributed along with the licensed software. For example, the license must not insist that all other programs distributed on the same medium must be open-source software.

10. License Must Be Technology-Neutral
No provision of the license may be predicated on any individual technology or style of interface.

Text Box 12 The Open Source Definition - Version 1.9

5. *To make a little web server project by utilising OpenSource software's like MapServer, Apache and MySQL.*

See http://81.69.54.149/ and cross your fingers… my Apache server is not always running. It is possible to ask me to start the server by send me an e-mail at t.m.ingvarsson@student.tudelft.nl.

Total estimated time

- Setting up the software's: It takes several hours to find and download the right software's and install them on your computer. I think that I spent a half day doing that… twice. Moreover I spent considerable time trying to find suitable OpenSource software to test and integrate with my project.
- Configuring MapServer can be time consuming, especially when configuring the DEMO. I think that I spent at least 1 day in both occasions, totally before it was ready to run!
- Tutorials can be done either fast or slow. I spent around 2-3 days going through the MySQL tutorial without using it later on in my project. But I see the potentials; maybe that is in a specific way both an important discovery for me and a conclusion to the project! Even though I did not implement any MapServer project by utilising MySQL.
- In the beginning there was no official MapServer tutorial so I spent several hours trying to implement something with the references from the official site. Finally I gave up and searched for better source and found in the end a good tutorial that is now also accessible on the MapServer official site. My way of doing the tutorial was to use different data than the tutorial used so I could learn more by implementing the examples that the tutorial described. That put me into situations where I could not use the tutorial to help me! Therefore it gave me better insight and understanding of the software. In total I spent around 3-4 days on the MapServer tutorial.
- Relation model of functions inside MapServer. I spent 1 day trying to understand MapServer in depth by doing some kind of UML model of the different MapServer functions. This can be seen in figure 12 in Appendix B.
- The architectural model of interactions of the web client, web server and MapServer. Spent around 2 days finalising the schema that can be seen in figure 5. This is my understanding of the structure but one can criticise it for not defining the CGI segment as a specific box that worked as a kind of hub between the Apache web server at one side, and map file and templates at the other. My view is that the CGI program is underlying and drives the commands from the web server along the MapServer infrastructure, and returns them back fulfilled. It is then the web server function to interact with the web client.
- I spent around 3 days reading documentations and references about MapServer and explored interesting MapServer sites to get hand on its capabilities.
- I spent at least 3 days in looking for sufficient data to use in my applications. First I was trying to get data from home with restricted success. Then I started to look for data provided by countries that allow free access to governmental data. M y choice was to download data from Canada. I spent a lot of time exploring this data, and must admit that I somehow lost sight of my objectives whereas I downloaded probably gigabytes of diverse spatial data that would eventually have no use for my project.
- Making my own project took much time. I did it twice as my computer down in the university lost connection during the implementation of my assignment and I

thought it would be easer to do the project again at home instead of doing it on computer with no network in the school. I would like to estimate that the total time spent on implementing the MapServer web map application were at least a week as I have done many experiments… and experienced a lot of tricky obstacles. For example, I did spend time working on the interface by trying to make it more fashion by using Javascript but my restricted knowledge got it all wrong. At the end I gave up as I thought it did not serve my final objectives to have fancy interface!

- Writing… a lot of time spent on sitting in front of the computer, but the actual writing was done in "few" days.
- In total I estimate the total time spent on the project to be at least 150 working hours, maybe over 200. At least I think that my workload is according to the project objectives. However I am not sure of how wisely some hours were spent as I explored many other fields of interest than served my objectives.

WHAT HAVE I LEARNT WITH THIS ASSIGNMENT?

- To start my own Apache server.
- To build up MapServer application on both Windows XP and 2000.
- I am now familiar with MapServer scripting and somewhat Javascript.
- I am now familiar with standards and specification made by OGC and I am capable to use WMS and WFS in my web-map applications.
- I am much wiser on HTML.
- I am almost an "expert" on finding spatial data and free software on the internet.
- I am capable to build up my own MySQL relational database and work with data in it.
- Experience of analysing software functionality by distinguishing particular component step by step and search for relations of these components. Finally, interpret it by means of graphical diagrams and UML (see e.g. figures 2 & 10)
- I am like a volcano of ideas that have been erupting parallel with my work on the assignment. Most of them are however probably not very profitable as I would like to implement them according to the concept of Wiki databases. Examples are:
    - Hiking tracks database (prototype can now be accessed at http://81.69.54.149/).
    - Winter off-road driving database.
    - Photo's with position database.
    - Abandoned farm database.
    - Service to farmhouse accommodations in Iceland.
- That theoretical material is much "easier" foundation for an essay than practical experiments… but hopefully I have learnt something :)

INTERESTING MAPSERVER SITES

Two impressive MapServer applications made for Swiss cantons:
http://sitn.ne.ch
http://www.so.ch/de/pub/departemente/bjd/gis/mapserver.htm#

Interesting site using MySQL with MapServer:
http://www.nrm.qld.gov.au/lris/webgis/burnett/index.html

Basic MapServer application that gives access to information about French telecommunication:
http://www.cartoradio.fr/

A page provided by DM Solution and Mapsherpa and shows example of MapServer applications powered with Java and Flash.
http://www.mapsherpa.com/make_a_map/

MapServer, users and online applications:
http://www.moximedia.com:8080/imf-ows/imf.jsp?site=ms_users

REFERENCES

Apache Software Foundation (visited 1.3.2004):
http://www.apache.org

Burn All GIFs. A project of the League for Programming Freedom (visited 1.3.2004):
http://burnallgifs.org/

DM Solution Group. Is the world largest single contributor to the MapServer, especially when considering the utility programs they offer. Homapage (last visited 1.3.04):
http://www.dmsolutions.ca

FreeGIS (visited 1.3.2004):
http://www.freegis.org

GeoConnections – Canadian GeoSpatial Data Infrastructure (visited 1.3.2004):
http://www.geoconnections.ca

Geo-Tools (visited 1.3.2004):
http://www.geotools.org

GISdevelopment (visited 1.3.2004):
http://www.gisdevelopment.net

GNU Lesser General Public License (visited 1.3.2004):
http://www.gnu.org/copyleft/lesser.html

GRASS Geographic Resources Analysis Support System (visited 1.3.2004):
http://grass.itc.it

OGR Simple Feature Library (visited 3.3.2004):
http://ogr.maptools.org/

OpenSource (visited 1.3.2004):
http://www.opensource.org

Lime, Stephen (2003): *GIS and the internet*. Lectures/PowerPoint slides. Homepage (visited 1.3.2004):
http://www.geog.umn.edu/courses/5574/

MapServer homepage (visited 1.3.2004):
http://mapserver.gis.umn.edu/

MapServer 4.0 Tutorial (visited 1.3.2004):
http://hypnos.cbs.umn.edu/projects/tutorial/

MySQL (visited 1.3.2004):
http://www.mysql.com

Open GIS Consortium (visited 1.3.2004):
http://www.opengis.org/

PostGIS Geographic Objects for PostgreSQL (visited 3.3.2004):
http://www.postgis.org/

Santitamnont, Dr.Phisan (2001): *A Comparative Assessment of Internet GIS Server Systems.* Survey Engineering Department, Chula Longkorn University. Homepage (last visited 1.3.2004):
http://www.gisdevelopment.net/technology/gis/techgi071pf.htm

The Common Gateway Interface (visited 1.3.2004):
http://hoohoo.ncsa.uiuc.edu/cgi/

Trinidad, Gerardo; Cole, Ivan and Chan, Wan-Yee (2000): *Developing Internet-based GIS Application*. Commonwealth Scientific and Industrial Research Organisation (CSIRO), Victoria Australia. Homepage (visited 1.3.2004):
http://www.ccs.dlsu.edu.ph/csp/docs/proceedings/posters/developing.pdf

MapServer-Wiki. Homepage (visited 1.3.2004):
http://mapserver.gis.umn.edu/cgi-bin/wiki.pl

Unisys. The Patent holder of GIF (visited 1.3.2004):
http://www.unisys.com

Wiki (visited 3.3.2004):
http://www.wiki.org

APPENDIX A: CHECK LIST

#	
#Start of MapFile	
#	
MAP	
DATAPATTERN	#[regular expression]
EXTENT	#[minx] [miny] [maxx] [maxy]
FONTSET	#[filename]
IMAGECOLOR	#[r] [g] [b]
IMAGEQUALITY	#[int]
IMAGETYPE	#[gif|png|jpeg|wbmp|gtiff|swf|userdefined]
INTERLACE	#[on|off]
NAME	#[name]
RESOLUTION	#[int]
SCALE	#[double]
SHAPEPATH	#[filename]
SIZE	#[x][y]
STATUS	#[on|off]
SYMBOLSET	#[filename]
TEMPLATEPATTERN	#[regular expression]
TRANSPARENT	#[on|off]
UNITS	#[feet|inches|kilometers|meters|miles|dd]

#	
#Projection output	
#	
PROJECTION	
"proj=utm"	
"ellps=GRS80"	
"zone=15"	
"north"	
"no_defs"	
END	

#	
#Web Object	
#	
WEB	
EMPTY	#[url]
ERROR	#[url]
FOOTER	#[filename]
HEADER	#[filename]
IMAGEPATH	#[path]
IMAGEURL	#[path]
LOG	#[filename]
MAXSCALE	#[double]
MAXTEMPLATE	#[file|url]
METADATA	
title "My layer title"	
author "Me!"	
END	
MINSCALE	#[double]
MINTEMPLATE	#[file|url]
TEMPLATE	#[filename|url]

END	
#	
#Query Map Object	
#	
QUERYMAP	
COLOR	#[r] [g] [b]
SIZE	#[x][y]
STATUS	#[on|off]
STYLE	#[normal|hilite|selected]
END	

#	
#Reference Map Object	
#	
REFERENCE	
COLOR	#[r] [g] [b]
EXTENT	#[minx][miny][maxx][maxy]
IMAGE	# [filename]
MARKER	#[integer|string]
MARKERSIZE	#[integer]
MINBOXSIZE	#[integer]
MAXBOXSIZE	#[integer]
OUTLINECOLOR	#[r] [g] [b]
SIZE	#[x][y]
STATUS	#[on|off]
END	

#	
#Legend Object	
#	
LEGEND	
IMAGECOLOR	#[r] [g] [b]
INTERLACE	#[on|off]
OUTLINECOLOR	#[r] [g] [b]
POSITION	#[ul|uc|ur|ll|lc|lr]
KEYSIZE	#[x][y]
KEYSPACING	#[x][y]
POSTLABELCACHE	#[true|false]
STATUS	#[on|off]
TRANSPARENT	#[on|off|embed]
END	

#	
#Scalebar Object	
#	
SCALEBAR	
BACKGROUNDCOLOR	#[r] [g] [b]
COLOR	#[r] [g] [b]
IMAGECOLOR	#[r] [g] [b]
INTERLACE	#[true|false]
INTERVALS	#[integer]
LABEL	
END	

OUTLINECOLOR	#[r] [g] [b]
POSITION	#[ul|uc|ur|ll|lc|lr]
POSTLABELCACHE	#[true|false]
SIZE	#[x][y]
STATUS	#[on|off]
STYLE	#[integer]
TRANSPARENT	#[on|off|embed]
UNITS	#[feet|inches|kilometers|meters|miles]
END	

#	
#Layer Object	
#	
LAYER	
CLASS	
BACKGROUNDCOLOR	#[r] [g] [b]
COLOR	#[r] [g] [b]
DEBUG	
EXPRESSION	#[string]
JOIN	#Signals the start of a JOIN object.
LABEL	
ANGLE	#[double]
ANTIALIAS	#[true|false]
BACKGROUNDCOLOR	#[r] [g] [b]
BACKGROUNDSHADOWCOLOR	#[r] [g] [b]
BACKGROUNDSHADOWSIZE	#[x][y]
BUFFER	#[integer]
COLOR	#[r] [g] [b]
FONT	#[name]
FORCE	#[true|false]
MAXSIZE	#[integer]
MINDISTANCE	#[integer|auto]
MINFEATURESIZE	#[integer]
MINSIZE	#[integer]
OFFSET	#[x][y]
OUTLINECOLOR	#[r] [g] [b]
PARTIALS	#[true|false]
POSITION	#[ul|uc|ur|cl|cc|cr|ll|lc|lr|auto]
SHADOWCOLOR	#[r] [g] [b]
SHADOWSIZE	#[x][y]
SIZE	#[integer]|[tiny|small|medium|large|giant]
TYPE	#[bitmap|truetype]
WRAP	#[character]
END	
MAXSIZE	#[integer]
MINSIZE	#[integer]
NAME	#[string]
OUTLINECOLOR	#[r] [g] [b]
SIZE	#[integer]
STYLE	
ANTIALIAS	#[true|false]
BACKGROUNDCOLOR	#[r] [g] [b]
COLOR	#[r] [g] [b]
MAXSIZE	#[integer]

MINSIZE	#[integer]
OFFSET	#[x][y]
OUTLINECOLOR	#[r] [g] [b]
SIZE	#[integer]
SYMBOL	#[integer|string]
END	
SYMBOL	#[integer|string]
TEMPLATE	#[filename]
TEXT	#[string]
END	
CLASSITEM	#[attribute]
CONNECTION	#[string]
CONNECTIONTYPE	#[local|sde|ogr|postgis|oraclespatial|wms]
DATA	#[filename]|[sde parameters][postgis table/column][oracle table/column]
DEBUG	
DUMP	#[true|false]
FEATURE	#Signals the start of a FEATURE object.
FILTER	#[string]
FILTERITEM	#[attribute]
FOOTER	#[filename]
GROUP	#[name]
HEADER	#[filename]
LABELANGLEITEM	#[attribute]
LABELCACHE	#[on|off]
LABELITEM	#[attribute]
LABELMAXSCALE	#[double]
LABELMINSCALE	#[double]
LABELREQUIRES	#[expression]
LABELSIZEITEM	#[attribute]
MAXFEATURES	#[integer]
MAXSCALE	#[double]
METADATA	
title "My layer title"	
author "Me!"	
END	
MINSCALE	#[double]
NAME	#[string]
OFFSITE	#[r] [g] [b]
POSTLABELCACHE	#[true|false]
PROCESSING	#[string]
PROJECTION	#input projection
"proj=utm"	
"ellps=GRS80"	
"zone=15"	
"north"	
"no_defs"	
END	
REQUIRES	#[expression]
SIZEUNITS	#[pixels|feet|inches|kilometers|meters|miles|
STATUS	#[on|off|default]
STYLEITEM	#[attribute]
SYMBOLSCALE	#[double]
TEMPLATE	#[file|url]
TILEINDEX	#[filename]
TILEITEM	#[attribute]
TOLERANCE	#[double]

TOLERANCEUNITS	#[pixels|feet|inches|kilometers|meters|miles|dd]
TRANSPARENCY	#[integer]
TRANSFORM	#[true|false]
TYPE	#[point|line|polygon|circle|annotation|raster|query]
END	

END #End of MapFile

APPENDIX B: THE ARCHITECTURE OF MAPSERVER

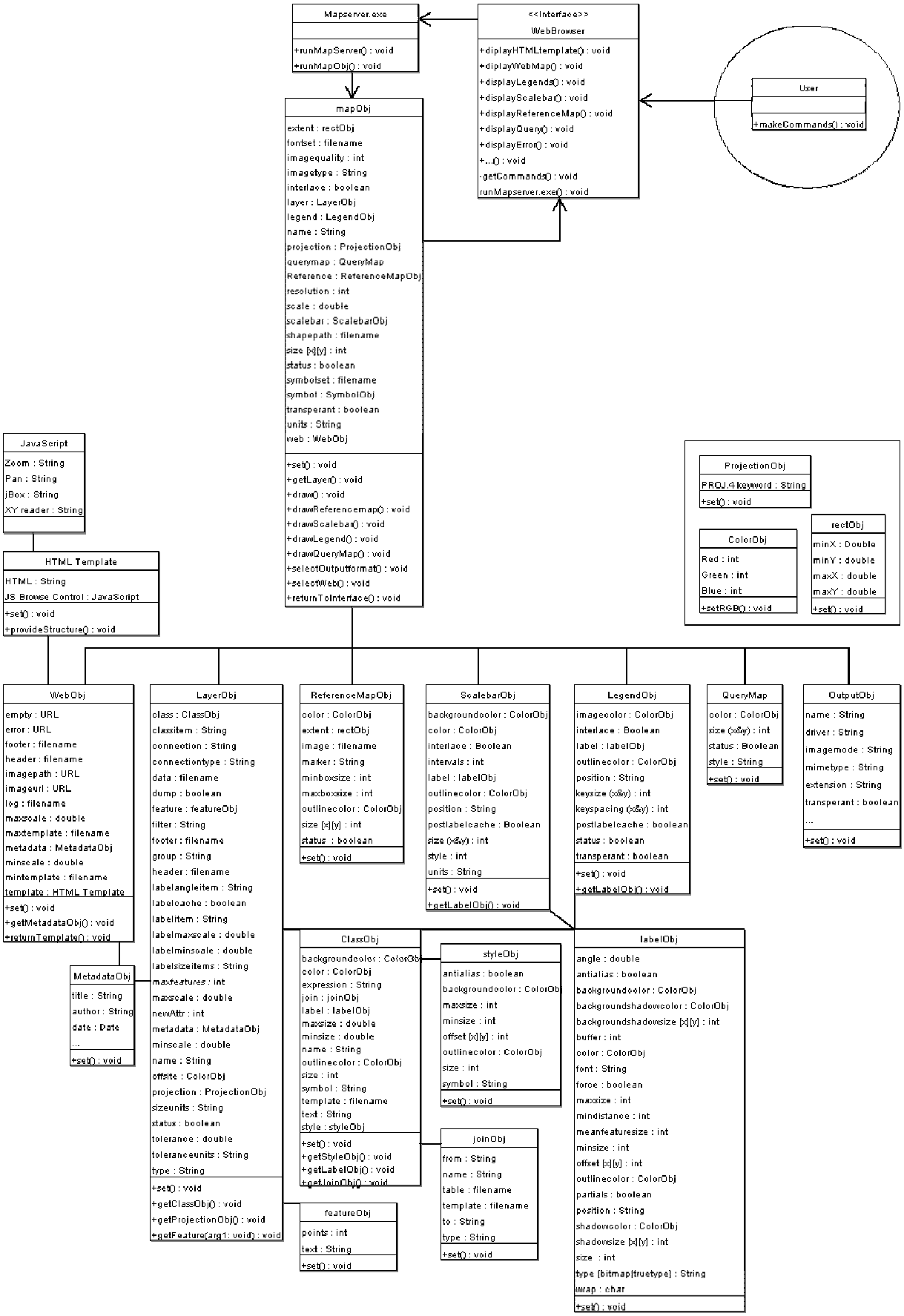

Figure 12 The structure and relationships of objects and parameters in MapServer